INSTRUCTIONS FOR USE

PLC Automation

Automation Builder, AC500

Automation Builder 2.5.0, AC500 V3, AC500-eCo V3, AC500-XC V3

Table of contents

1 PLC Automation with V3 CPUs

1.1 About this document

1.1.1 Documentation structure

↳ *Chapter 1.1.4 “Use the "magic button" to display your current position in the table of contents” on page 11.*

↳ *See also chapter " Your tasks - documentation from the user's point of view" on page 9.*

Table 1: Guidance for this documentation: Main chapters

Getting started Basic information to start with Automation Builder and AC500 PLC, e.g., licensing, GUI explanations, example projects.	↳ *Chapter 1.2 “Getting started” on page 11*
Automation Builder installation manager Add, remove or modify software packages in Automation Builder.	↳ *Chapter 1.3 “Automation Builder installation manager” on page 169*
Programming with CODESYS Information about IEC programming in Automation Builder, including description of CODESYS libraries.	↳ *Chapter 1.4 “Programming with CODESYS” on page 176*
Libraries and solutions ABB libraries. Overview and description of integrated standard libraries and solution libraries available as library packages. Explanation of the concept of solution libraries ("system technology"). Description of the library elements, like function blocks and functions.	↳ *AC500 V3 library descriptions: Chapter 1.5 " Libraries and solutions" on page 2146* ↳ *Chapter 1.10 “Reference, function blocks” on page 4292*
PLC integration (hardware) Hardware description and specifications. Overview on module variants, connections, technical data, order data, assembly of modules. Device configuration in Automation Builder. Explanation of system behavior ("system technology"), interaction between PLC behavior (firmware), configuration, programming and use cases.	↳ *Chapter 1.6 “PLC integration (hardware)” on page 2384*
Diagnosis and debugging Explanation of the diagnosis system in the PLC, the display of error messages at the CPU and in IEC applications. Online diagnosis in Automation Builder. List of diagnosis and error messages.	↳ *Chapter 1.7 “Diagnosis and debugging for AC500 V3 products” on page 4011*
Engineering interfaces and tools Information on add-on packages, e.g., for security static analysis or Project Version Control. Mainly for advanced users.	↳ *Chapter 1.8 “Engineering interfaces and tools” on page 4112*
Human machine interface (HMI) Information on HMI with Automation Builder. Configuration of HMI devices in Automation Builder.	↳ *Chapter 1.9 “Human machine interface” on page 4281*
Contact ABB Contact information about our sales and support teams.	↳ *Chapter 1.11 “Contact ABB” on page 4408*

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

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

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

↳ *Chapter 1.1.4 “Use the "magic button" to display your current position in the table of contents” on page 11.*

↳ *See also chapter " Documentation structure" on page 8.*

As a mechanical/electrical designer

PLC system description	↳ *Chapter 1.6.2.7 “PLC system description” on page 2421*
Hardware descriptions	↳ *Chapter 1.6.3 “Device specifications” on page 2430*
Comparison of product features	● ↳ *Chapter 1.6.1 “Product overview and comparison” on page 2384* ● *via product catalog (online)*

As a switchgear cabinet manufacturer

Assembly of modules	↳ *Chapter 1.6.4 “System assembly, construction and connection” on page 3333*
Connection of modules	In the device specifications, select the desired product to access the connection for this device ↳ *Chapter 1.6.3 “Device specifications” on page 2430*. *“Device specifications ➔ Product group ➔ Product type ➔ Electrical connection”*
Installation instructions	*AC500 V2 + V3 (online)*

As a programming engineer

Getting started with Automation Builder	↳ *Chapter 1.2 “Getting started” on page 11*
Installation of Automation Builder	*AC500 V2 + V3 (online)*
License management for Automation Builder	↳ *Chapter 1.2.4 “Managing your licenses” on page 20*
Getting started with example projects	↳ *Chapter 1.2.18 “Example projects” on page 61*
Firmware update	↳ *Chapter 1.6.6.1.4 “Firmware identification and update” on page 3652*
Configuration of PLC hardware in Automation Builder	↳ *Chapter 1.6.6 “Configuration in Automation Builder for AC500 V3 products” on page 3631*
Programming with CODESYS	↳ *Chapter 1.4 “Programming with CODESYS” on page 176*
Function block libraries	Libraries by ABB ↳ *Chapter 1.5 “Libraries and solutions” on page 2146* CODESYS libraries by 3S ↳ *Chapter 1.4.4 “Libraries” on page 1248*
System behavior ("system technology"), interaction between PLC (firmware), configuration, programming and use cases.	↳ *Chapter 1.6.5 “System technology for AC500 V3 products” on page 3455*

Visualization and web visualization: Example projects	*Chapter 1.2.18.1.2 “Example project for central I/O expansion” on page 63*
Visualization and web visualization	*Chapter 1.4.5 “CODESYS Visualization” on page 1249*
Add, remove or modify software packages in Automation Builder via installation manager	*Chapter 1.3 “Automation Builder installation manager” on page 169*
Add-on software packages	*Chapter 1.8 “Engineering interfaces and tools” on page 4112*
HMI, e.g., interface to control panels and SCADA systems	*Chapter 1.9 “Human machine interface” on page 4281*

As a commissioning engineer

Function block libraries	Libraries by ABB *Chapter 1.5 “Libraries and solutions” on page 2146* CODESYS libraries by 3S *Chapter 1.4.4 “Libraries” on page 1248*
Hardware descriptions	*Chapter 1.6.3 “Device specifications” on page 2430*
Diagnosis and debugging	*Chapter 1.7 “Diagnosis and debugging for AC500 V3 products” on page 4011*

As a service engineer

Diagnosis and debugging	*Chapter 1.7 “Diagnosis and debugging for AC500 V3 products” on page 4011*
List of diagnosis and error messages	*Chapter 1.7.3 “Diagnosis messages” on page 4062*
Contact the PLC support team	*Chapter 1.11 “Contact ABB” on page 4408*

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

AC500 V3 CPU specifications	*Chapter 1.6.3.3 “Processor modules” on page 2440*
Comparison of product features	● *Chapter 1.6.1 “Product overview and comparison” on page 2384* ● *via product catalog (online)*
Convert an AC500 V2 project to an AC500 V3 project	*Chapter 1.2.17 “Converting an AC500 V2 project to an AC500 V3 project” on page 61*
Compatible modules with AC500 CPUs	*Chapter 1.6.3 “Device specifications” on page 2430*
Documentation for AC500 V2	*AC500 V2 (online)*

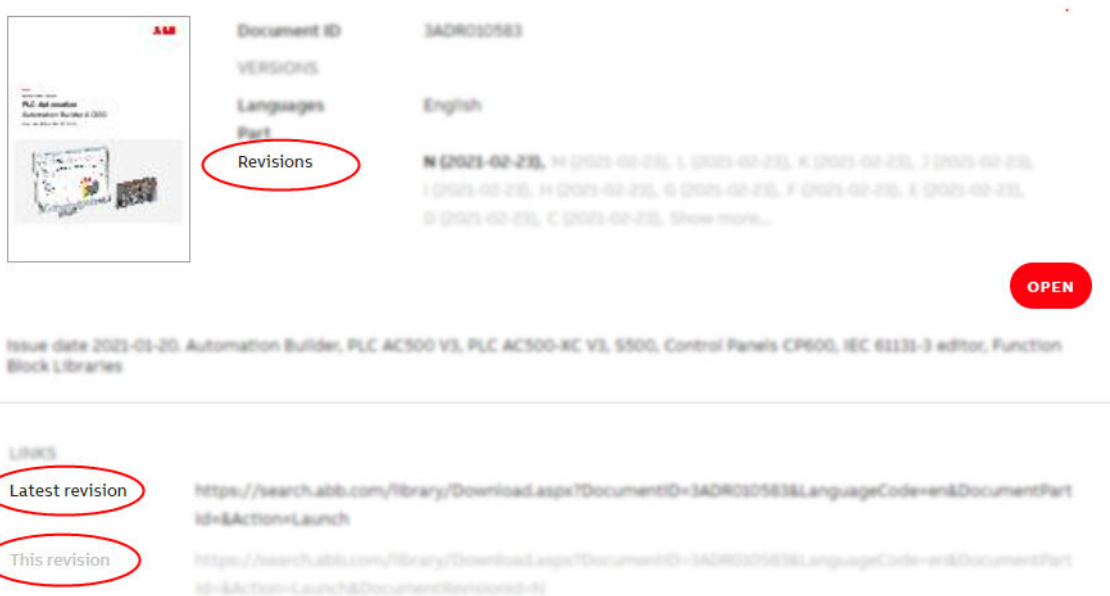
As a specialist for PLCs, new to AC500 PLC

Getting started with engineering suite Automation Builder	*Chapter 1.2 “Getting started” on page 11*
PLC system description	*Chapter 1.6.2.7 “PLC system description” on page 2421*
Hardware descriptions	*Chapter 1.6.3 “Device specifications” on page 2430*
System technology: System behavior, interaction between PLC behavior (firmware), configuration, programming and use cases.	*Chapter 1.6.5 “System technology for AC500 V3 products” on page 3455*

1.1.3 Older revisions of this document

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

AC500 V3 (online)



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

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

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

▷ Select the "magic button".

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

1.2 Getting started

ABB Automation Builder - One holistic suite

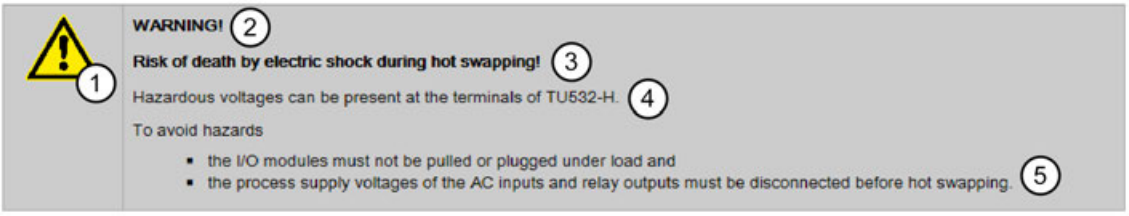
ABB Automation Builder is the integrated software suite for machine builders and system integrators wanting to automate their machines and systems in a productive way. Combining the tools required for configuring, programming, debugging and maintaining automation projects from a common intuitive interface,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

Automation Builder ReadMe

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

1.2.1 Structure of safety notices

Throughout the documentation we use the following types of safety and information notices. They make you aware of safety considerations or give advice on AC500 products usage.



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

Signal words

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

Ensure to take measures to prevent the described impending danger.

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

Ensure to take measures to prevent the described dangerous situation.

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

Ensure to take measures to prevent the described dangerous situation.

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

Ensure to take measures to prevent the described dangerous situation.

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

1.2.2 Cyber security

Cyber security disclaimer

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

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

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

Security related deployment guidelines for industrial automation

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

Signed firmware updates

The firmware update files for the AC500 V3 PLC are digitally signed releases by ABB. During the update process, these signatures are validated by a hardware security component in the PLC. This way, the AC500 V3 PLC will only update with valid, authentic firmware, signed by ABB.

Open ports and services

As part of the ABB security concept the AC500 V3 PLC comes with minimal services opened by default. Only the services needed for initial setup and programming are open before any user application is downloaded *Chapter 1.6.1.3 “Ethernet protocols and ports for AC500 V3 products” on page 2389*.

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

Secure communication

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

Secure shell access for ABB service

The AC500 V3 PLC contains a secure shell service to access core logging data in case of problems which need a deeper analysis. This service is inactive by default, which means that no one can access this privileged shell in the normal operating state.

To activate this service, local access to the PLC is necessary and activation is only valid until the next power cycle of the PLC. Once activated, the service run on TCP port 22. Each PLC also protects the secure shell access by an individual password.

Frequently asked questions

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

1.2.2.1 Defense in depth

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

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

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

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

References: *CODESYS Security Whitepaper*

Security zones

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

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

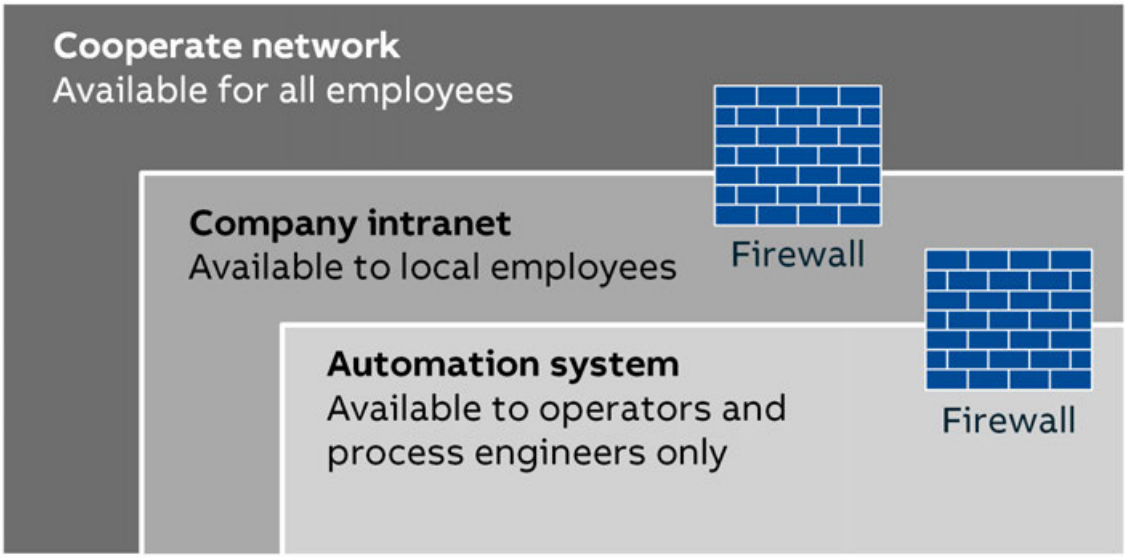


Fig. 1: Security zones

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

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

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

References: *Security for Industrial Automation and Control Systems*

1.2.2.2 Secure operation

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

A protected environment can be:

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

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

Follow these rules for the protected environment:

- Keep the trusted network as small as possible and independent from other networks.
- Protect the cross-communication of controllers and the communication between controllers and field devices via standard communication protocols (fieldbus systems) using appropriate measures.
- Protect such networks from unauthorized physical access.
- Use fieldbus systems only in protected environments. They are not protected by additional measures, such as encryption. Open physical or data access to fieldbus systems and their components is a serious security risk.

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

A protected local control cabinet could look like in figure 2, page 16. This network is not connected to any external network. Security is primarily a matter of physically protecting the automation system and preventing unauthorized users from accessing the system and from connecting or installing unauthorized hardware and software.

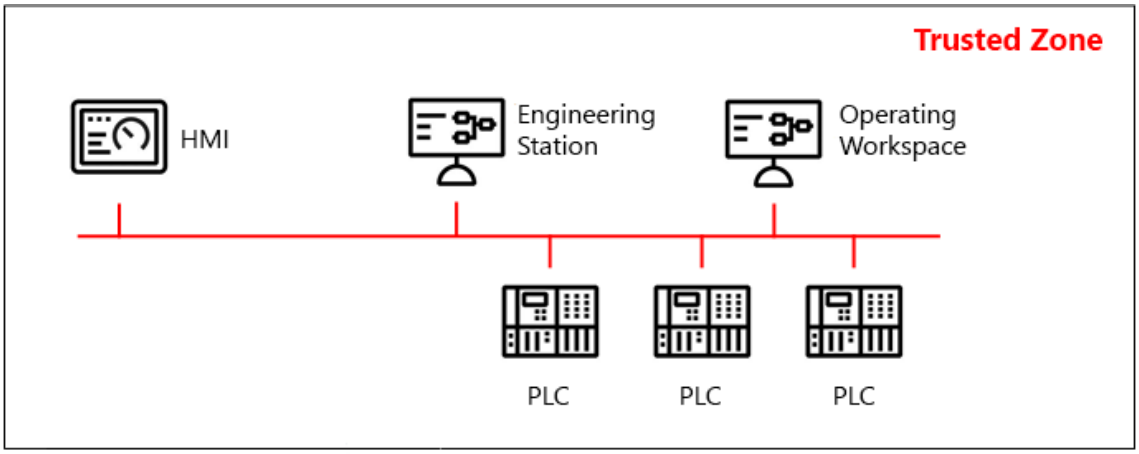


Fig. 2: Isolated automation system

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

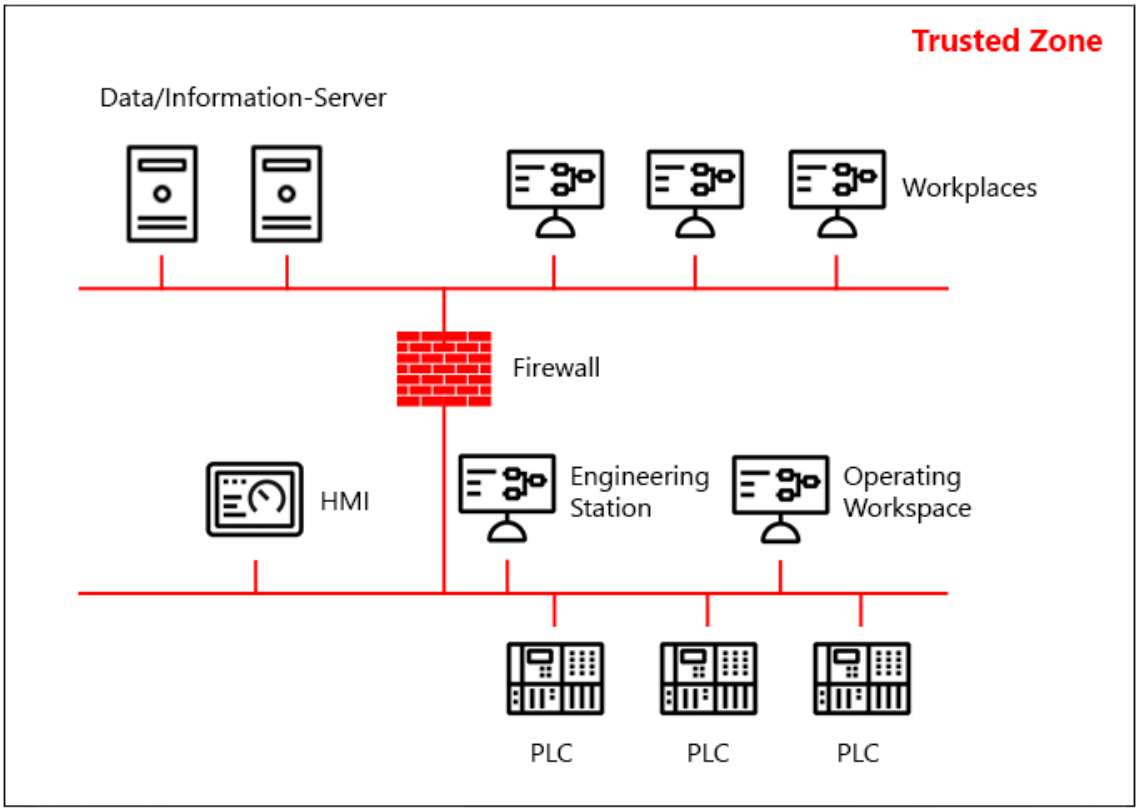


Fig. 3: Plant information network connected to an automation system

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

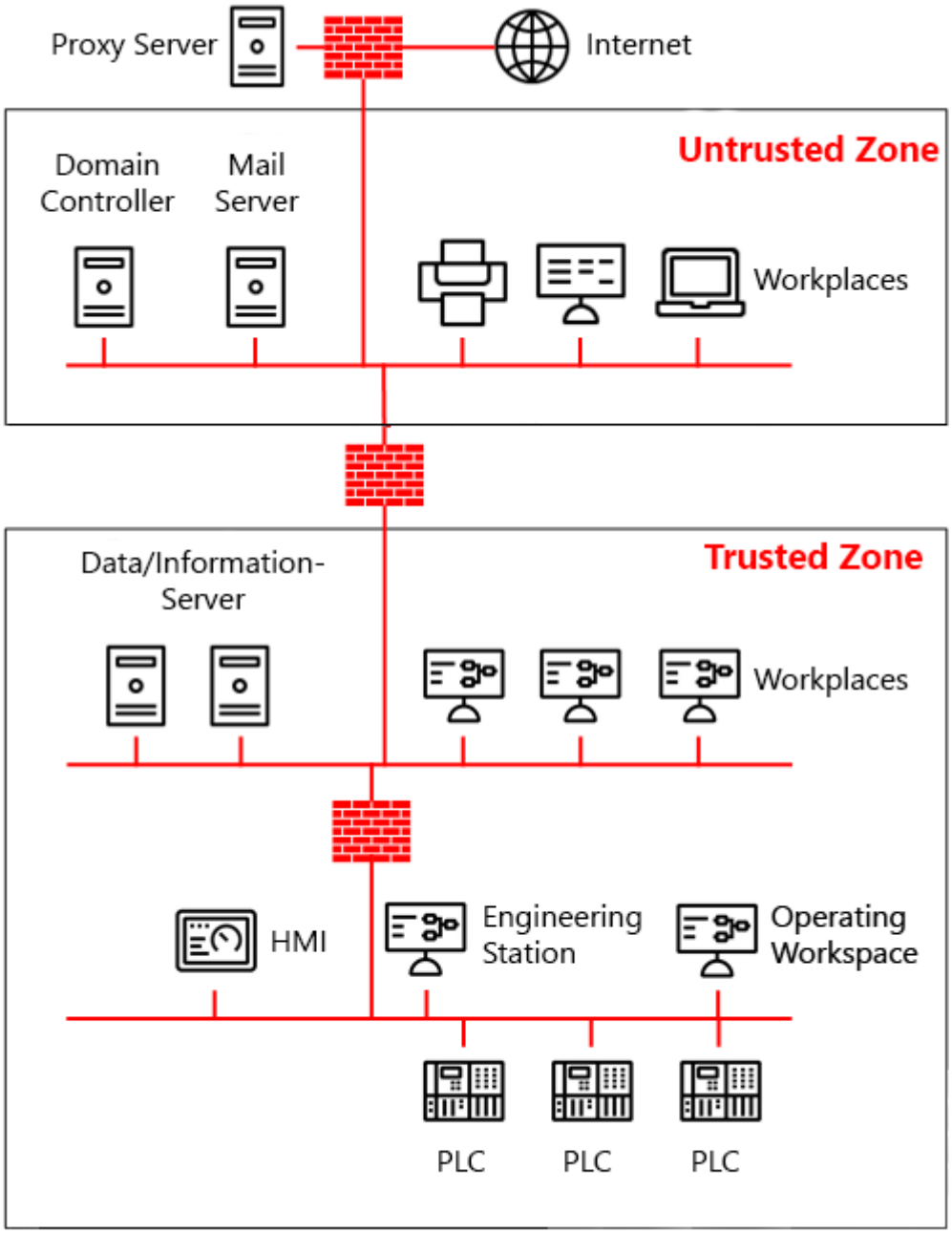


Fig. 4: Automation system and IS network

1.2.2.3 Hardening

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

Commissioning phase

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

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

Verification phase

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

Operation phase

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

Decommissioning phase

- Delete all credentials stored in the device like certificates and user data *Chapter 1.6.4.4.6 “Decommissioning” on page 3351*.

References: *Hardening in Wikipedia (2021)*

1.2.2.4 Open Ports and Services

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

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

1.2.3 Automation Builder update notification

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

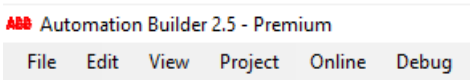
Chapter 1.3.3 “Automation Builder update notification” on page 172

1.2.4 Managing your licenses

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

1.2.4.1 Identifying the installed license

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



Be aware of the following rule for this information:

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

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

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

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

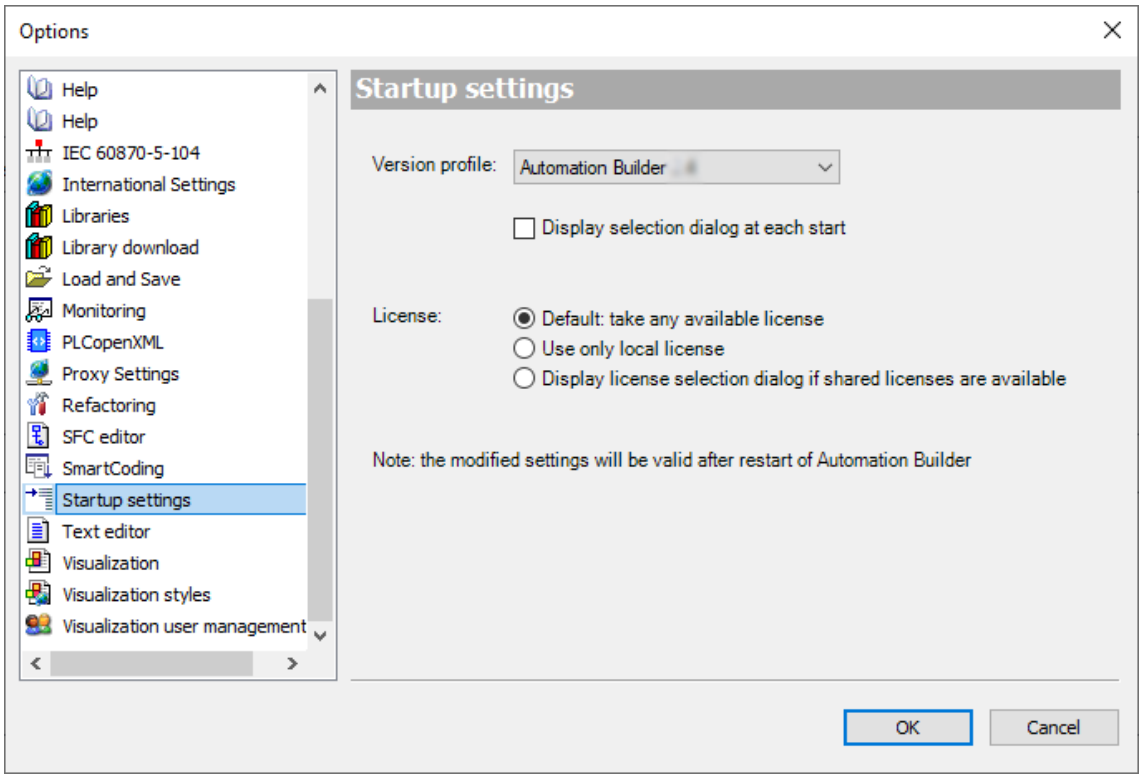
To check your installed licenses, the CodeMeter Control Center tool can be used ↳ *Chapter 1.2.4.3 “Checking licenses with CodeMeter control center” on page 22*.

1.2.4.2 Selecting the license used on Automation Builder startup

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

To select which license should be used:

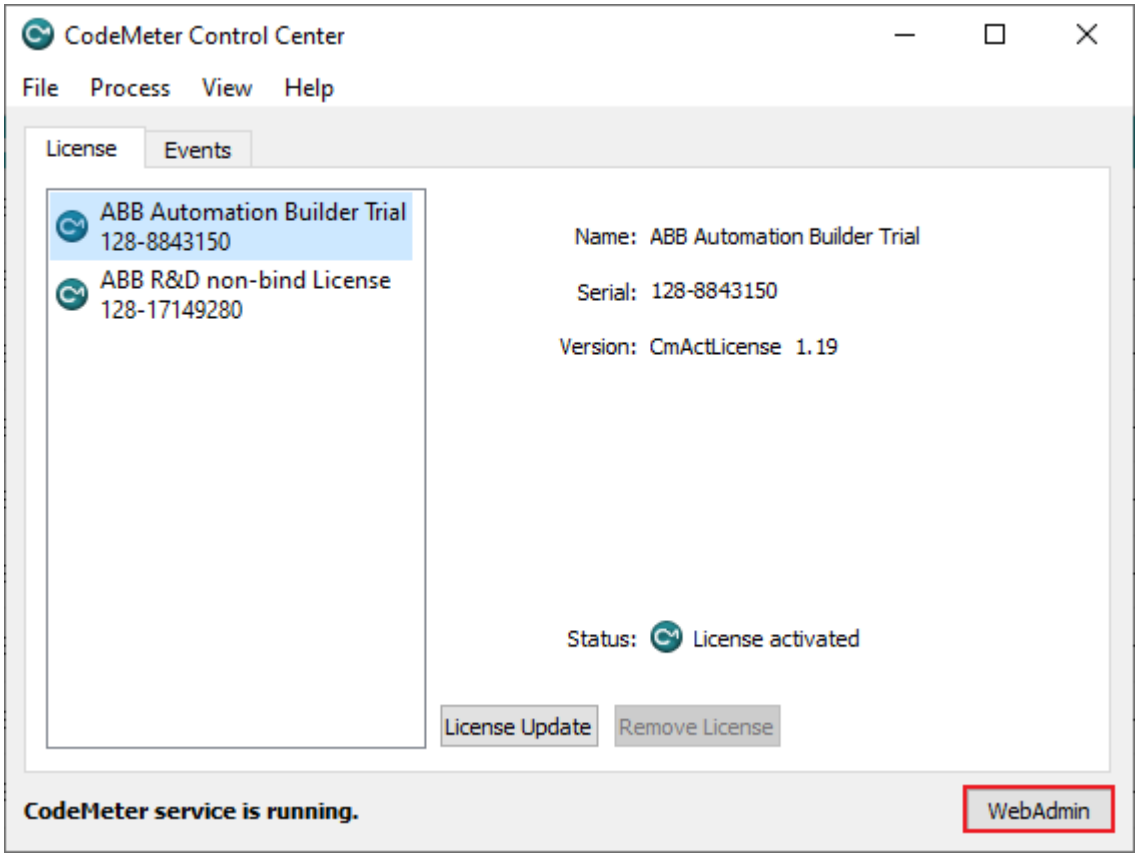
1. In the Automation Builder menu select *“Tools ➔ Options”*.
   ⇨ The Options window is opened.
2. In *“Startup settings”* under *“License”* select which license should be used.
   - Default: The most comprehensive available license will be selected
   - Use only local license: Network licenses will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



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

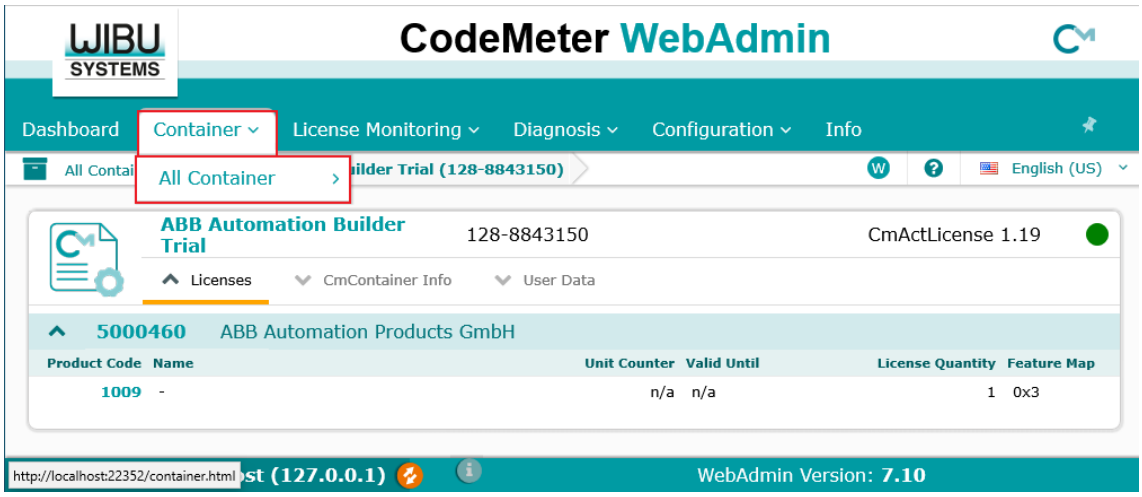
1.2.4.3 Checking licenses with CodeMeter control center

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



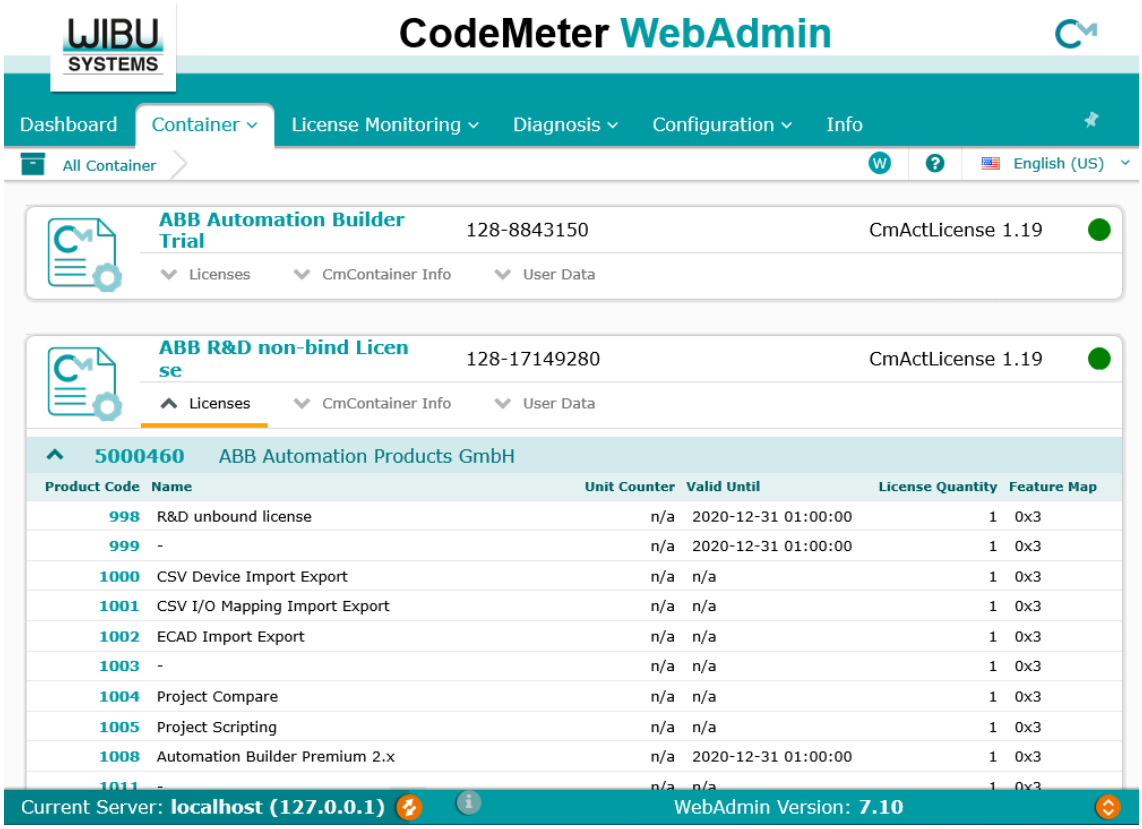
2. In the CodeMeter Control Center window you can see the different license “tickets” / “CmContainers” that are installed on your PC.

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



3. Select *“Container ➔ All Container”*

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

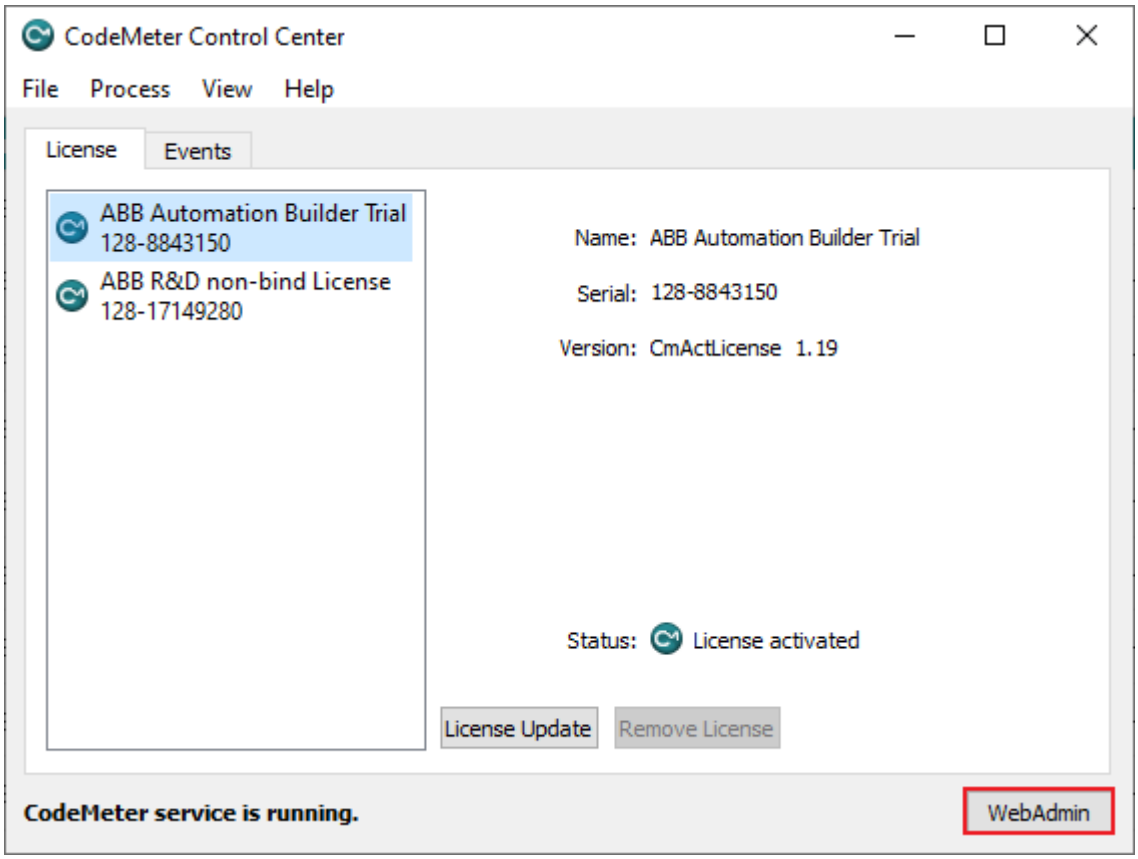


1.2.4.4 Setting dedicated network servers in search list

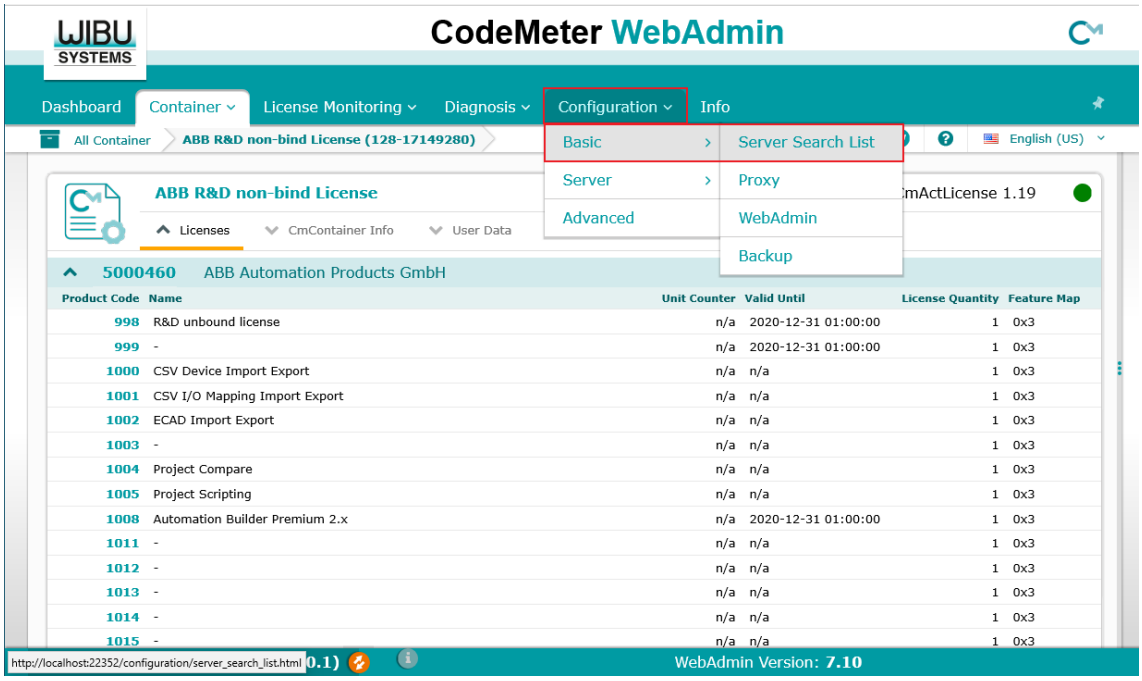
In case of a new installation CodeMeter will check for licenses also in the network. If there is a run-out or wrongly installed license found, the service is closed without any further hint. This looks like Automation Builder starts and closes after a few moments.

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

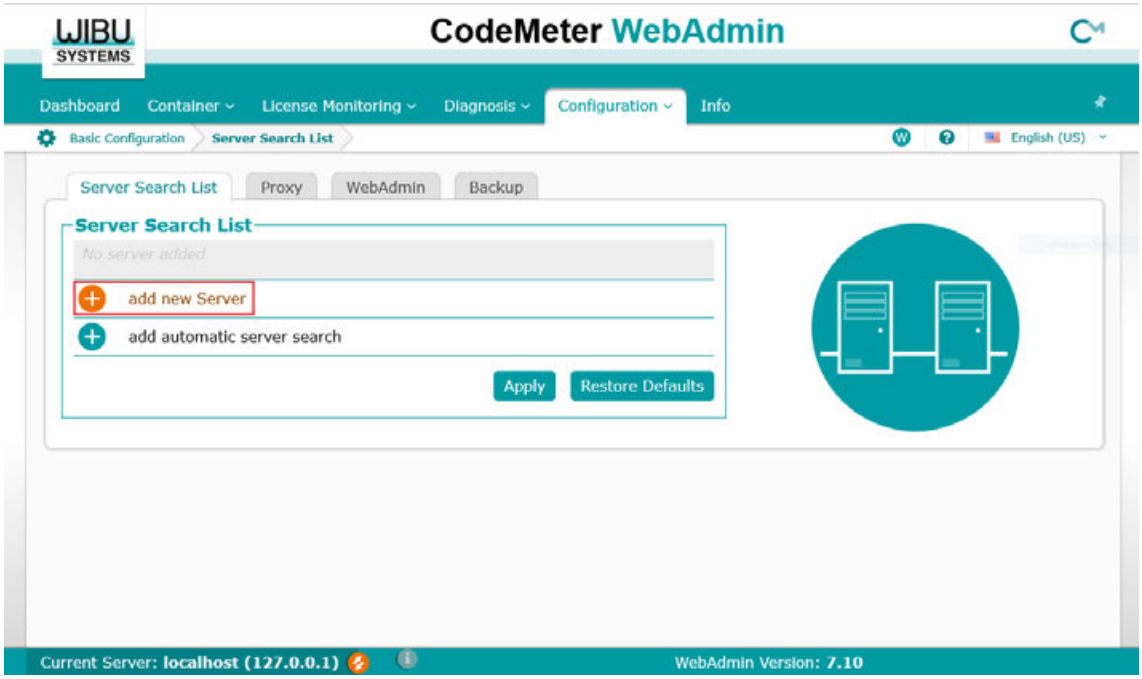
1. Open the CodeMeter Control Center. See ↬ *Chapter 1.2.4.3 “Checking licenses with CodeMeter control center” on page 22*
2. Open the CodeMeter WebAdmin by selecting *“WebAdmin”*



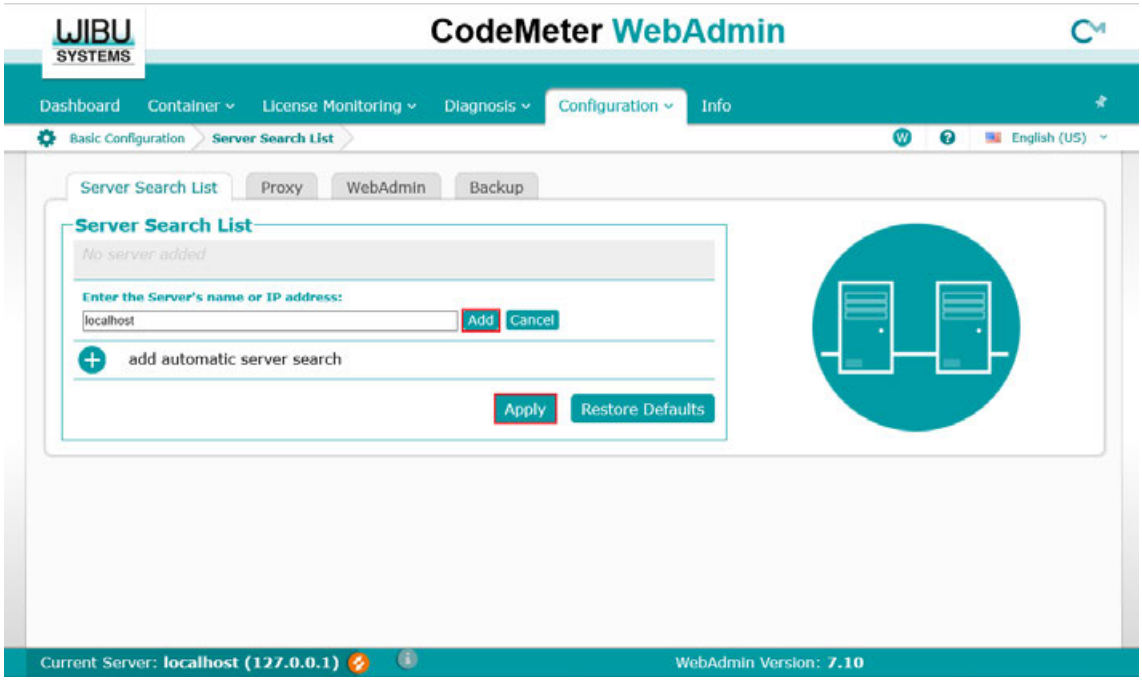
3. Select *"Configuration ➔ Basic ➔ Server Search List"*



4. Select *"add new Server"*



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

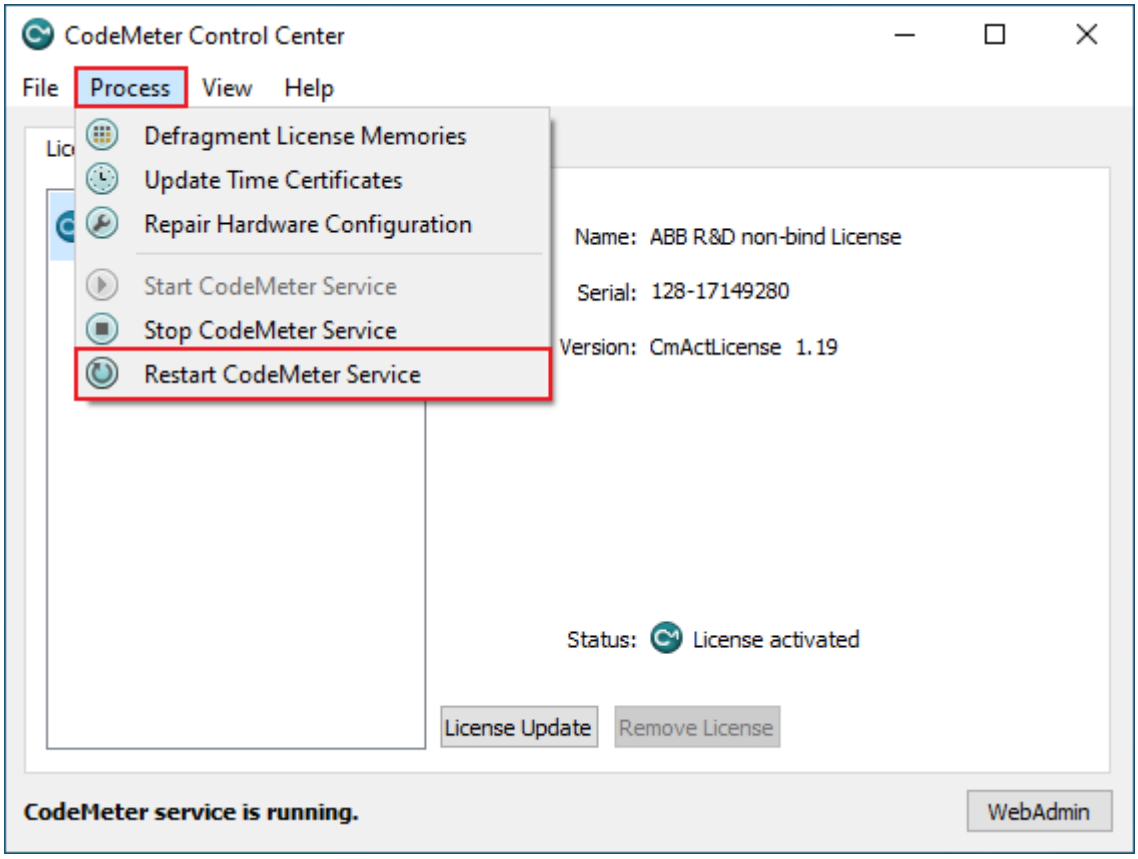


8. Restart the license check
9. Add more servers to the search list by entering the IP-Adress or name of the license servers you know.

1.2.4.5 Restarting license check with a dongle bound license

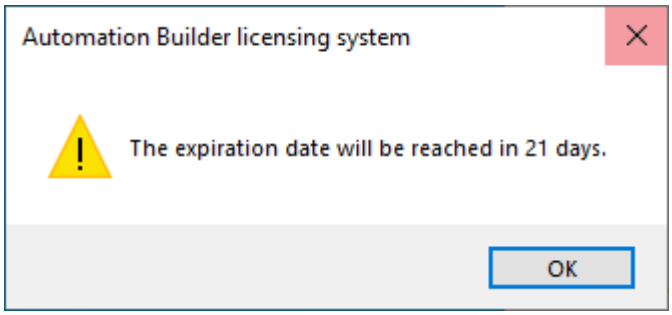
In case of using a dongle bound license it might be necessary to restart the check for license on the PC, e.g. if the dongle was removed and reinserted.

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



1.2.4.6 Removing trial license to remove expiring message

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

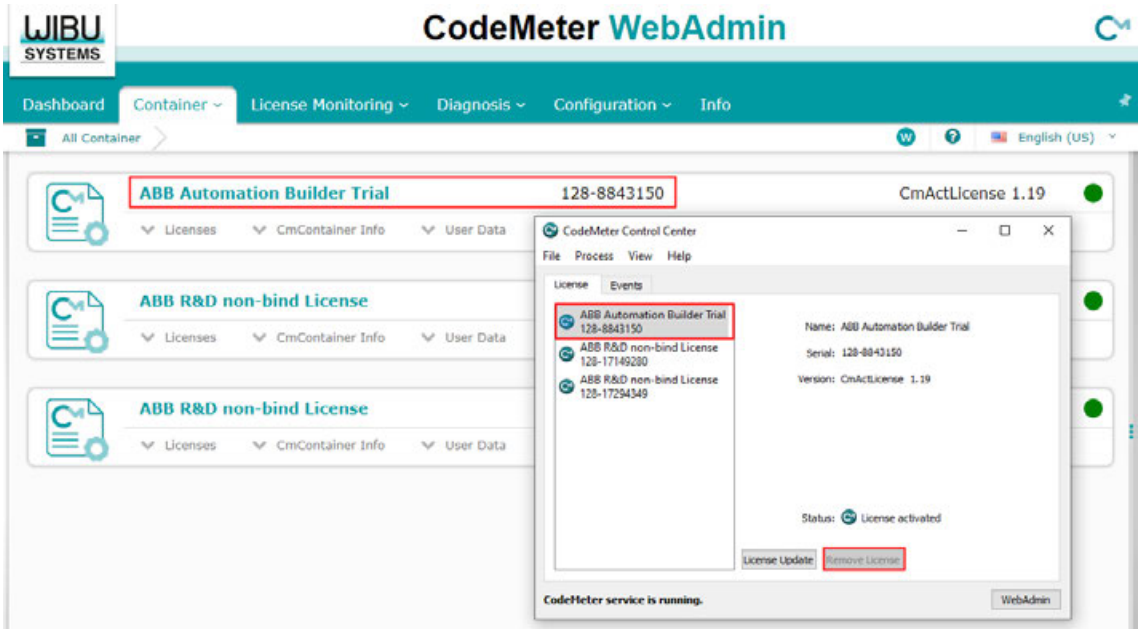


To avoid this message the trial license can be removed.

CAUTION!

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

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



3. Remove this selected license in CodeMeter Control Center

1.2.4.7 Network licenses

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

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

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

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

1.2.4.7.1 Setting up a network license

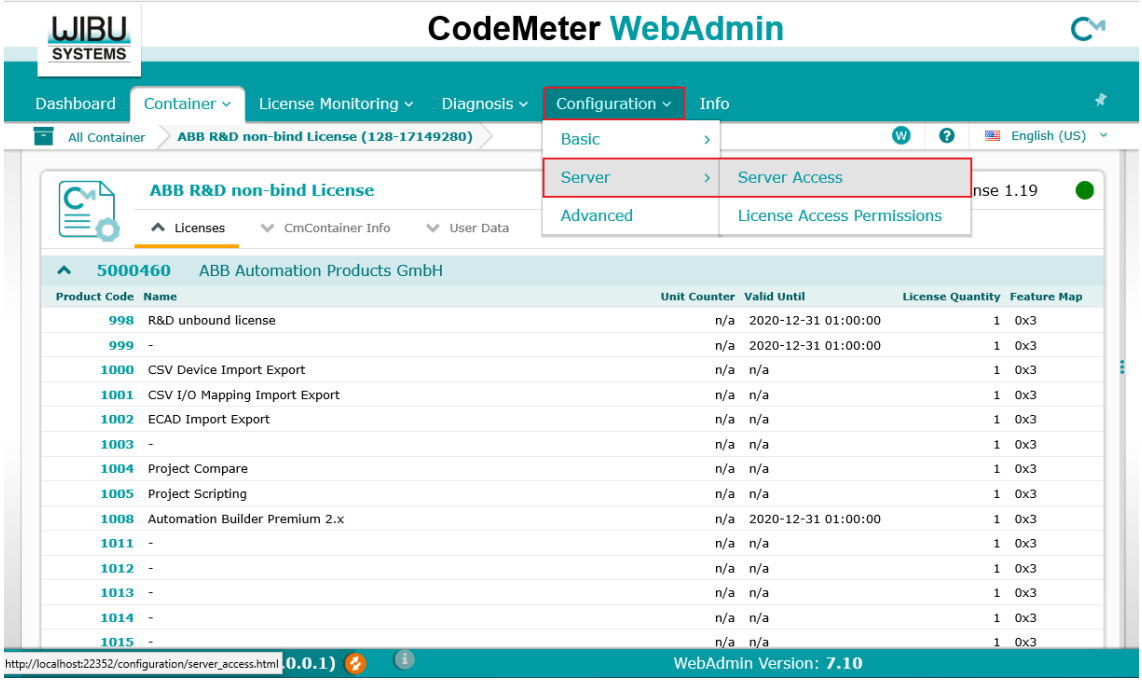
The following setup works in typical environments.

Configuring a network license server

☑ Network license must be registered.

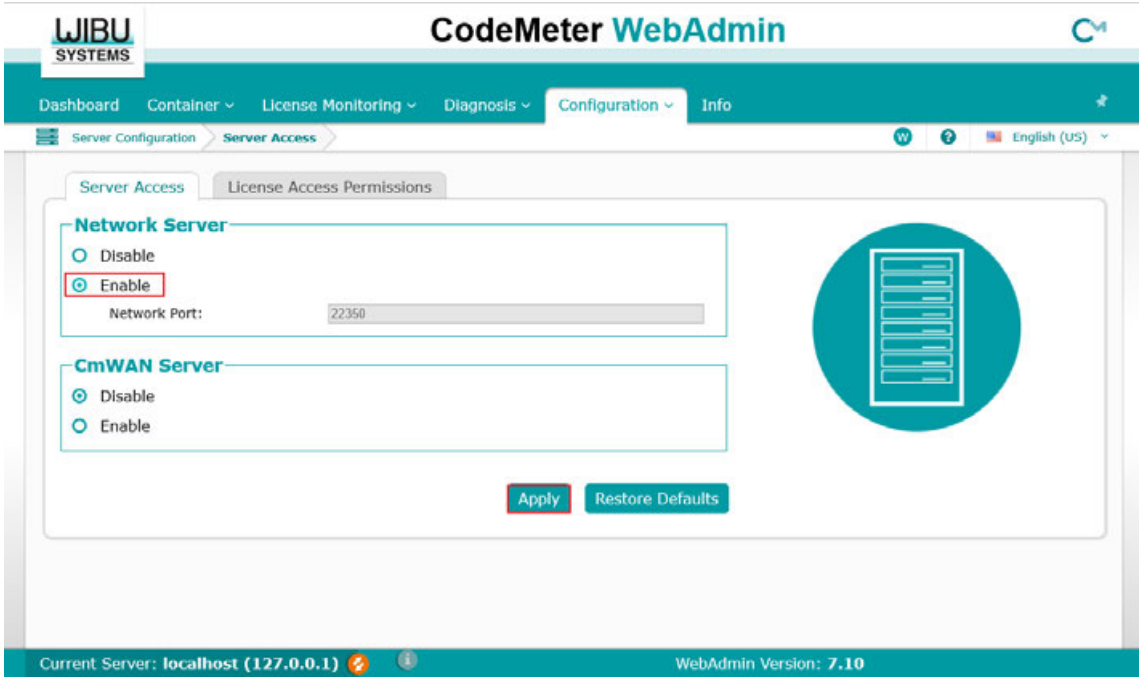
☑ Automation Builder license must be activated.

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



3. Enable Network Server

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



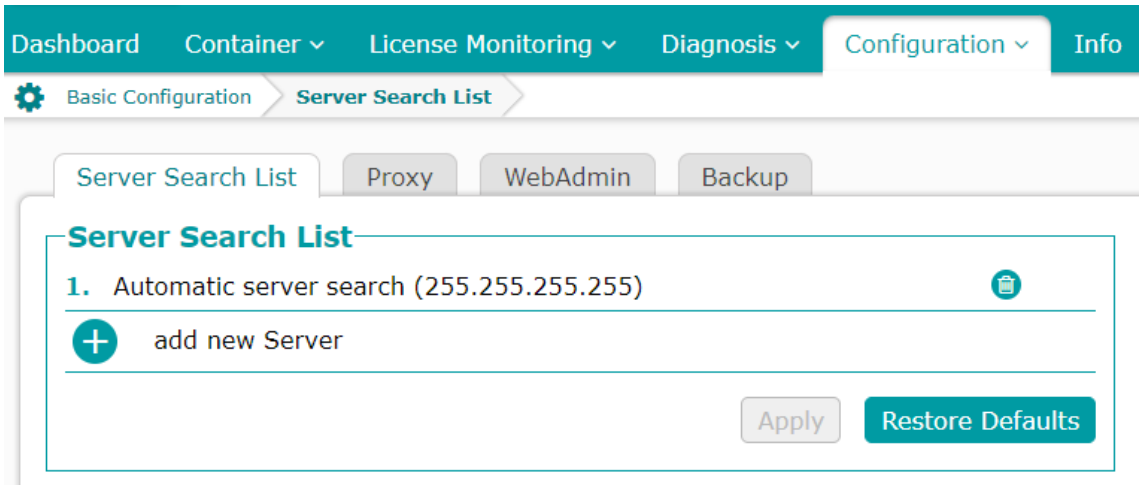
5. For the changes to take effect, restart CodeMeter Control Center see *Chapter 1.2.4.5 “Restarting license check with a dongle bound license” on page 25*

NOTICE!

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

Configuring the client side

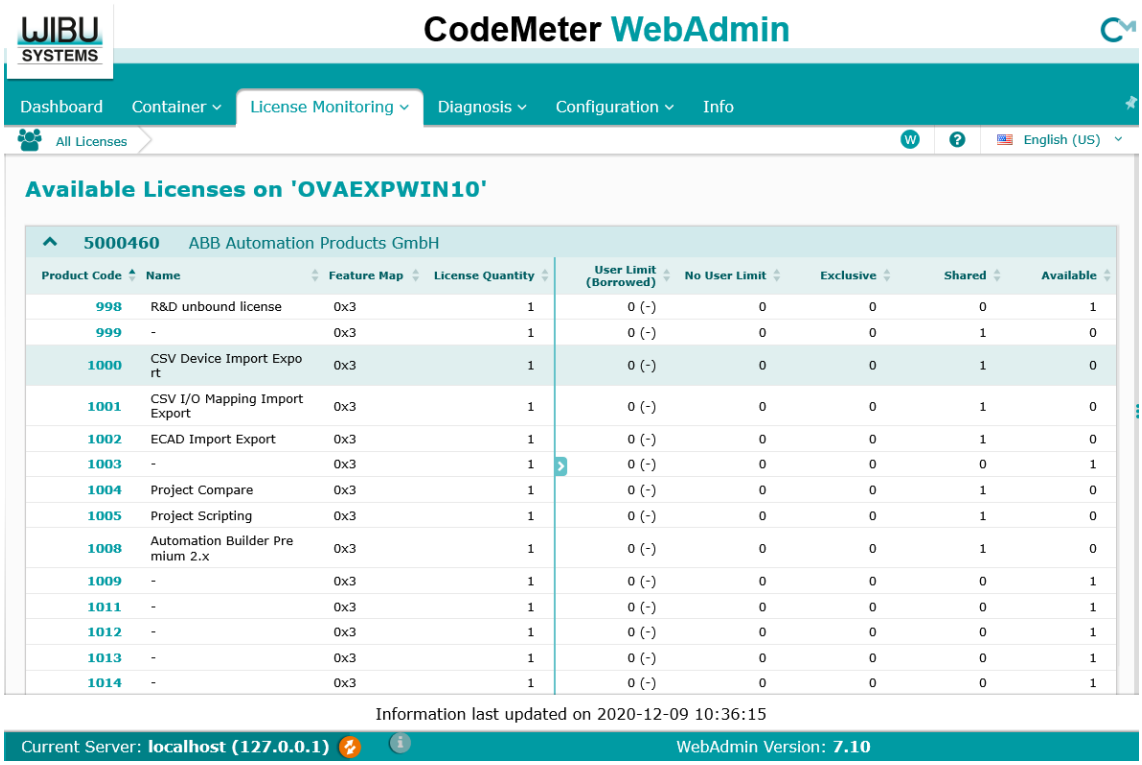
The default settings of Automation Builder and the CodeMeter (on client side) are sufficient in most cases to get the network licenses working. In case of problems accessing the network license, please set the server search list on the client side.



1.2.4.7.2 View network server licenses

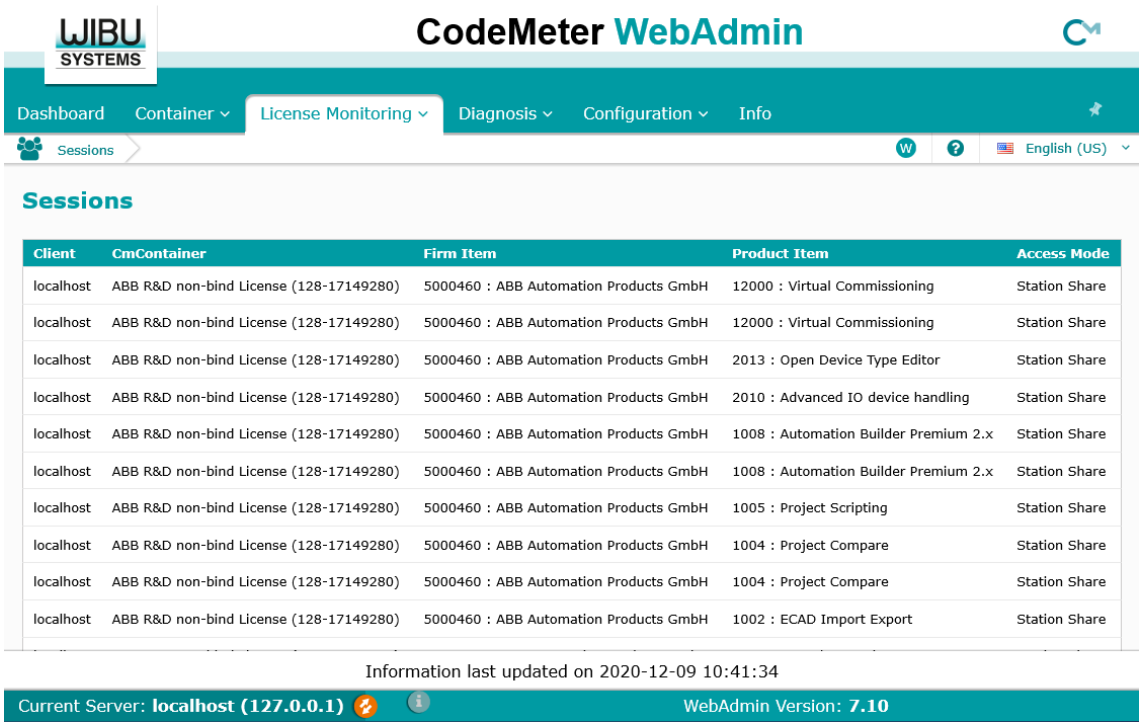
On the Network Server side you can find information on existing network licenses and their current allocation.

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



1.2.4.7.3 View network server license usage

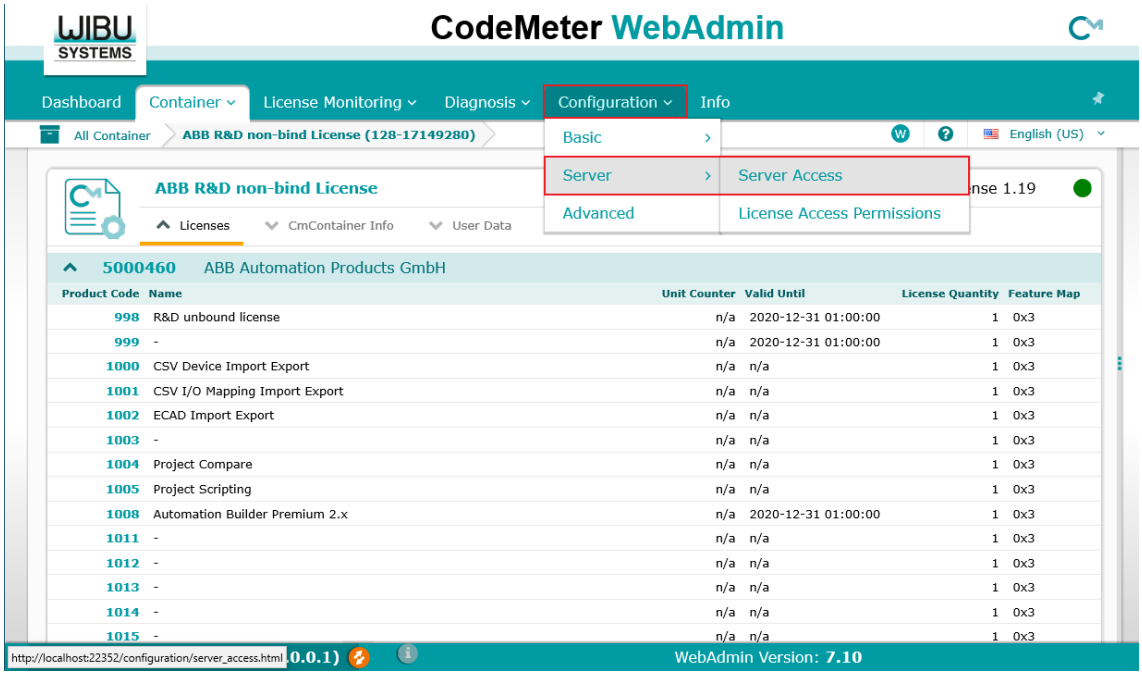
1. Launch CodeMeter WebAdmin. See ↳ *Chapter 1.2.4.3 "Checking licenses with CodeMeter control center" on page 22*
2. Select *"License Monitoring ➔ Sessions "*



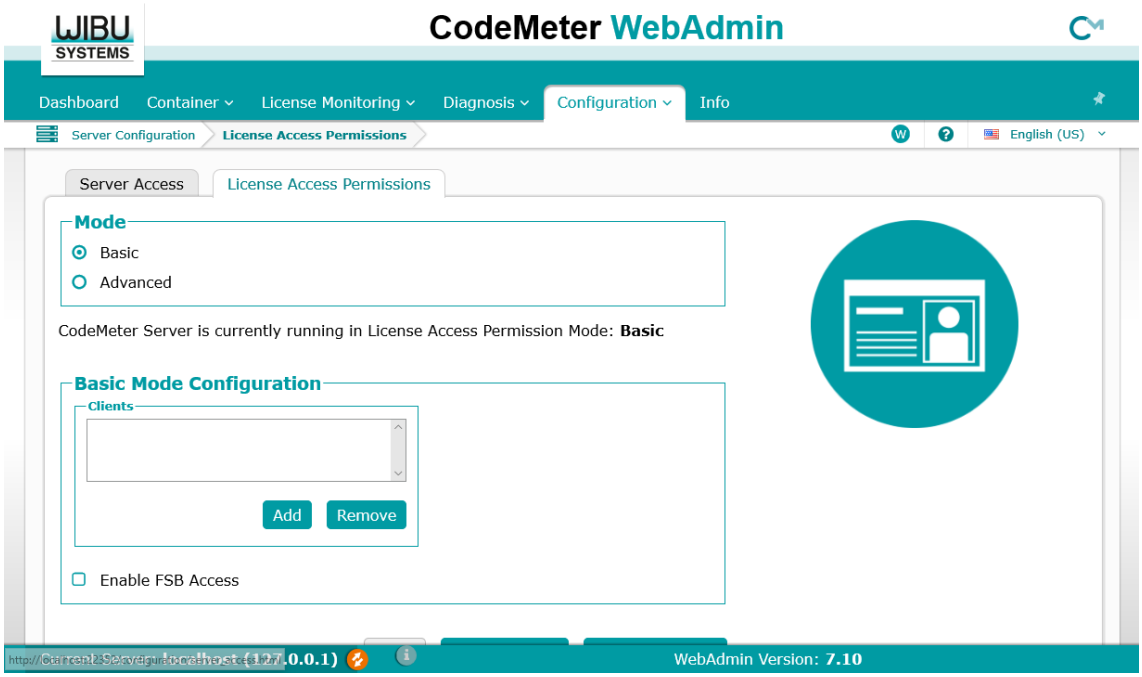
1.2.4.7.4 Controlling network server license usage

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

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



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



1.2.4.8 License borrowing manager

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

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

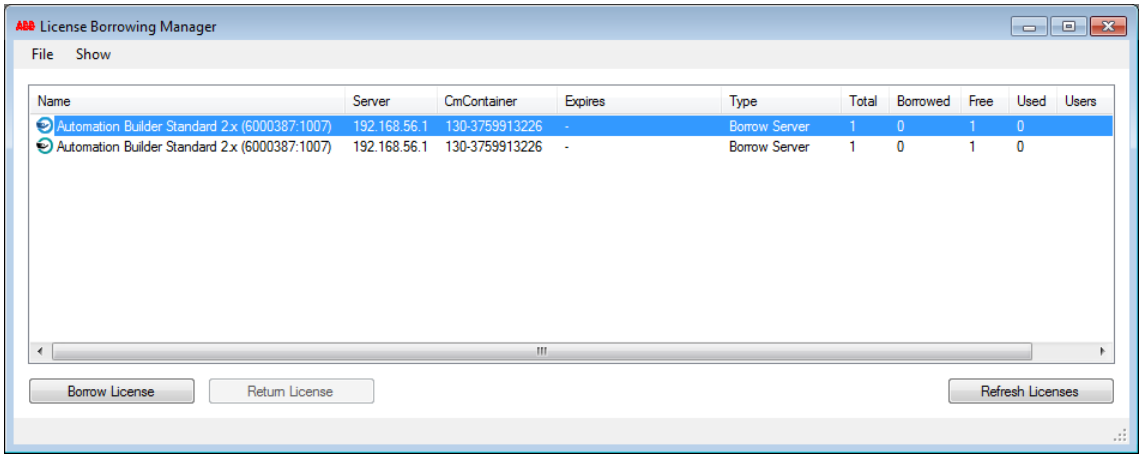
> *The license borrowing manager is only supported by Automation Builder 2.2.3 and later.*

1.2.4.8.1 Borrowing a network license

☑ Network access to the license server required.

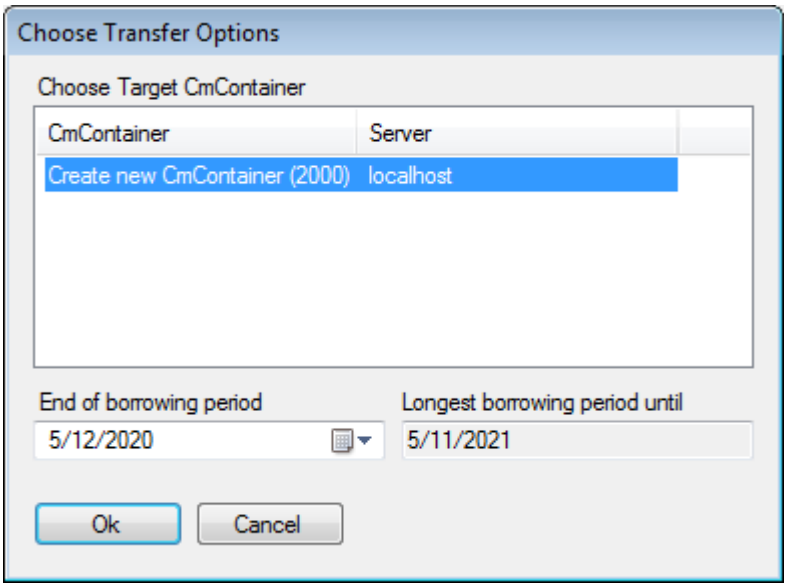
☑ Opened the license borrowing manager.

1. Select the license you want to borrow.



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

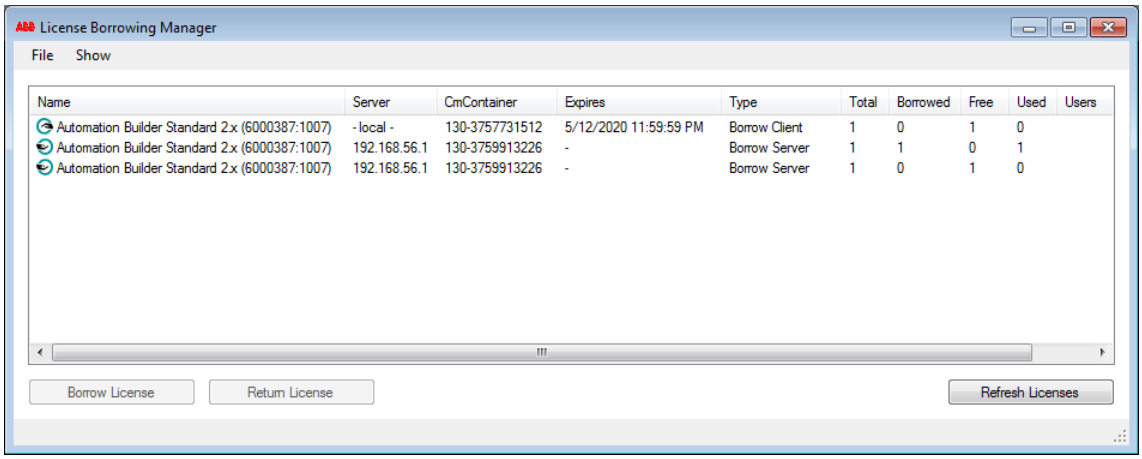
   Alternatively a new CmContainer can be created.
4. Select the end of the borrowing period.



5. Select *“OK”*.

   ⇨ The license has sucessfully been borrowed.

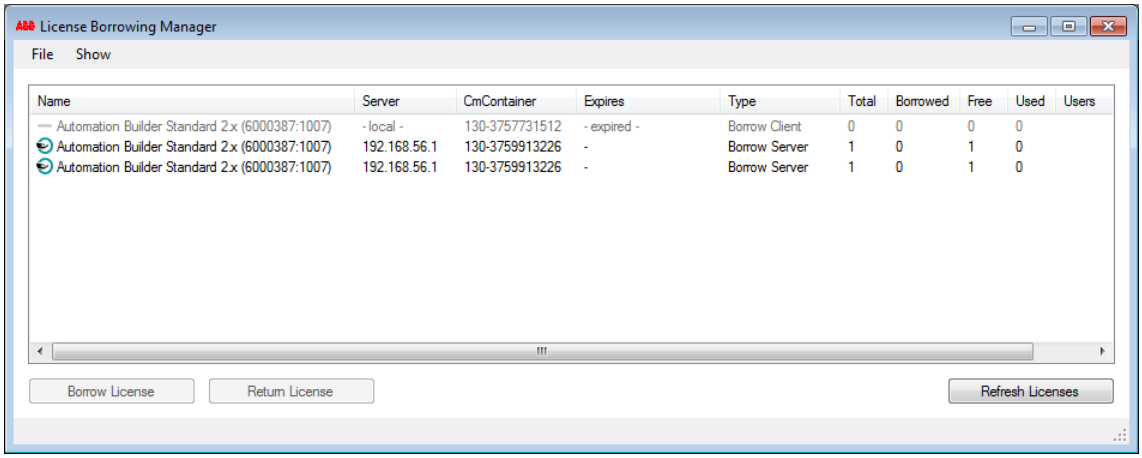
   The list of available licenses has been updated.



1.2.4.8.2 Returning a network license

Automatical return of a license

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



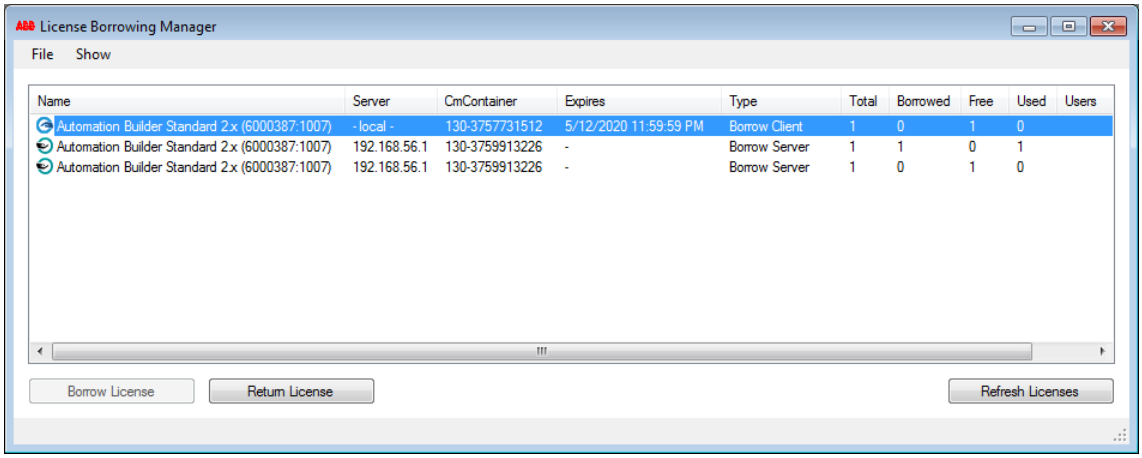
Manual return of a network license

Network licenses can be returned anytime manually.

☑ Network access to the license server required.

☑ Opened the license borrowing manager.

1. Select a borrowed license.



2. Select *“Return License”*

   ⇨ The license has sucessfully been returned.

1.2.4.9 Transfering an Automation Builder license

1.2.4.9.1 General

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

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

The process consists of two main steps:

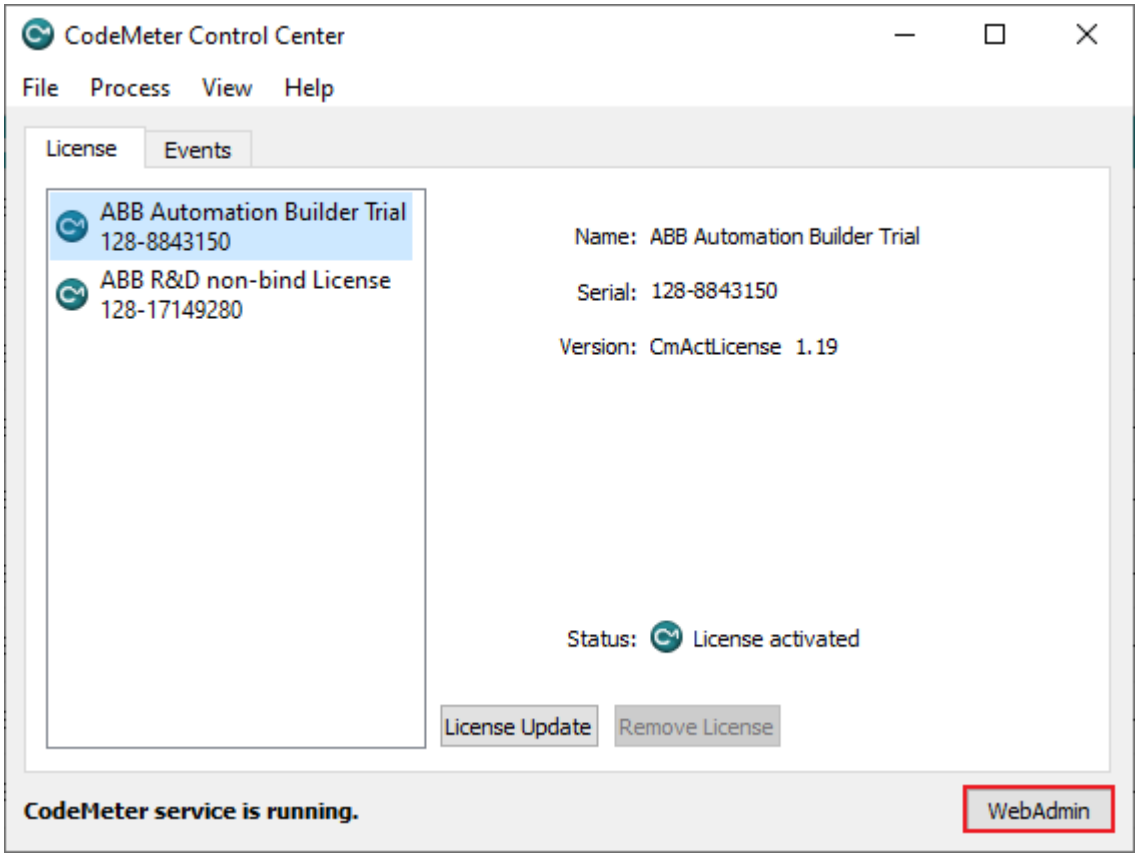
1. Return the actual license from the actual PC
2. Reactivate the license on the new PC

1.2.4.9.2 Getting activation code

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

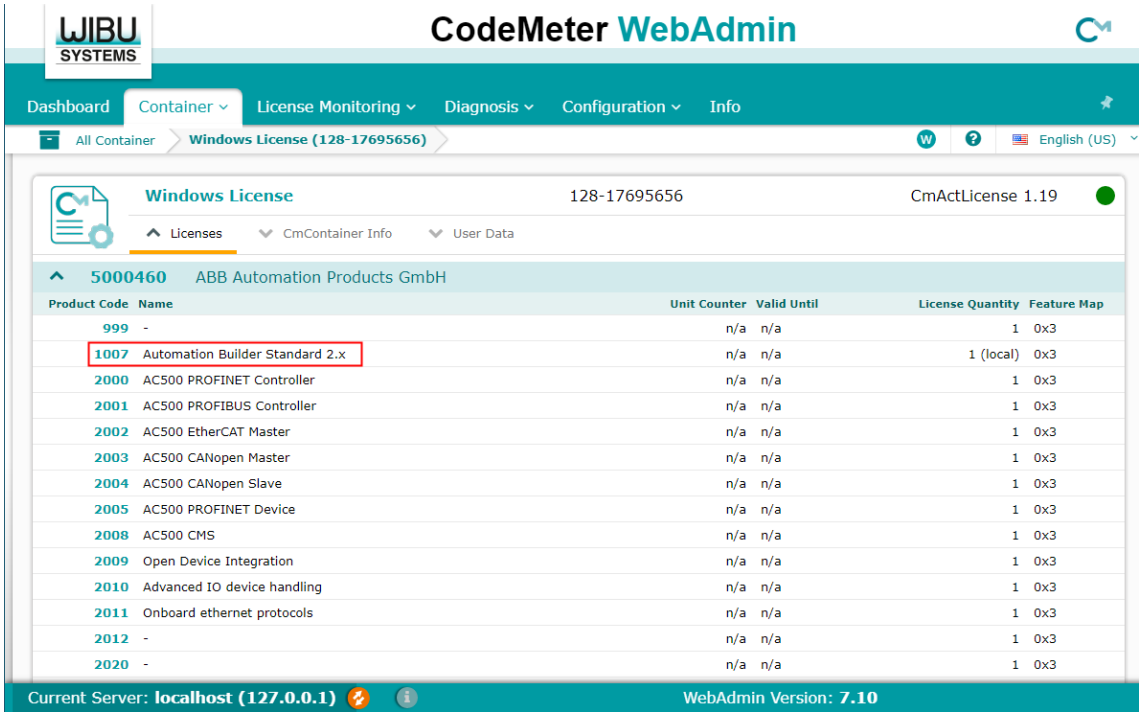
For Automation Builder licenses purchased April 2020 or later, the activation code is available from the activated license:

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

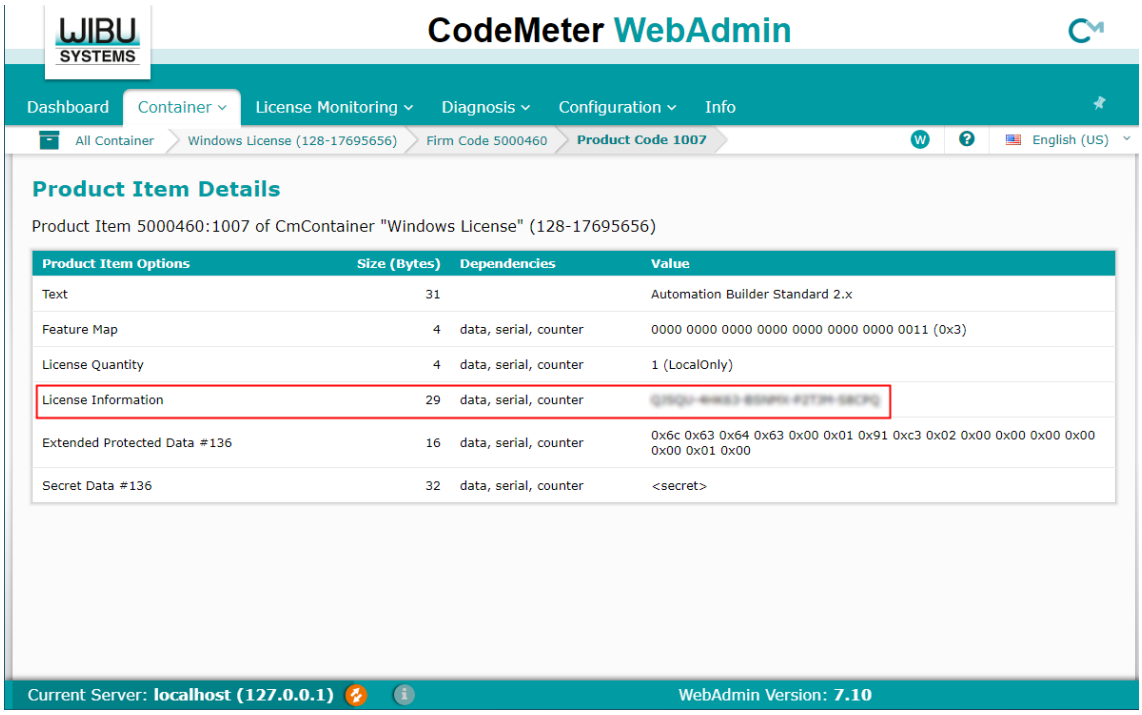


2. Identify the right product code.

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



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



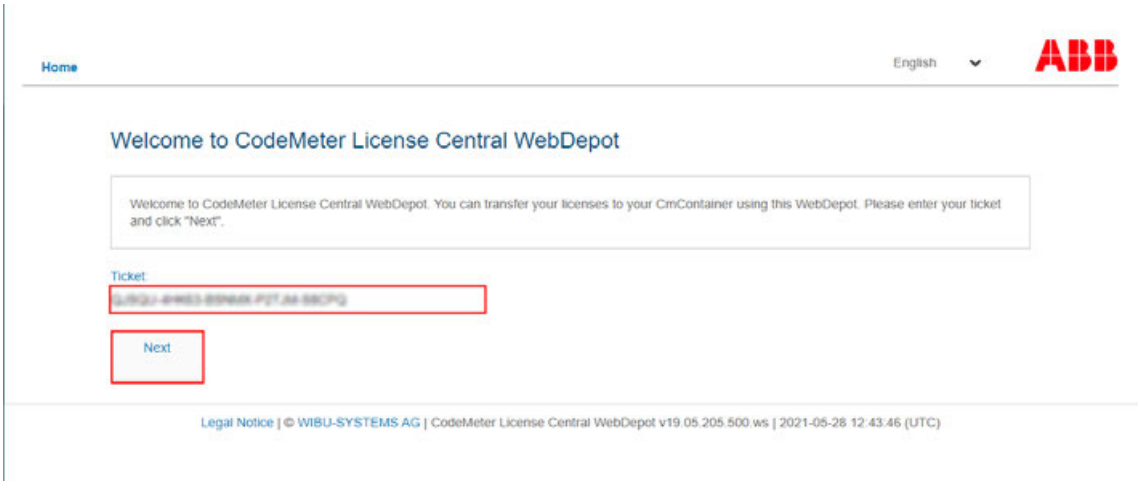
1.2.4.9.3 Returning an Automation Builder license

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

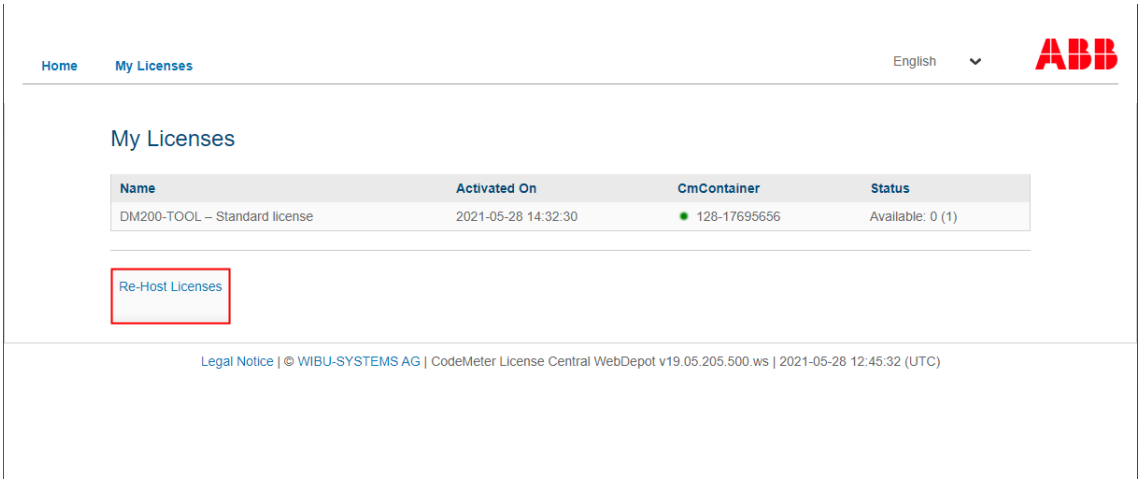
1. Go to the following website: *http://lc.codemeter.com/32838/depot-return/index.php*

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

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



4. Select *“Re-Host License”*

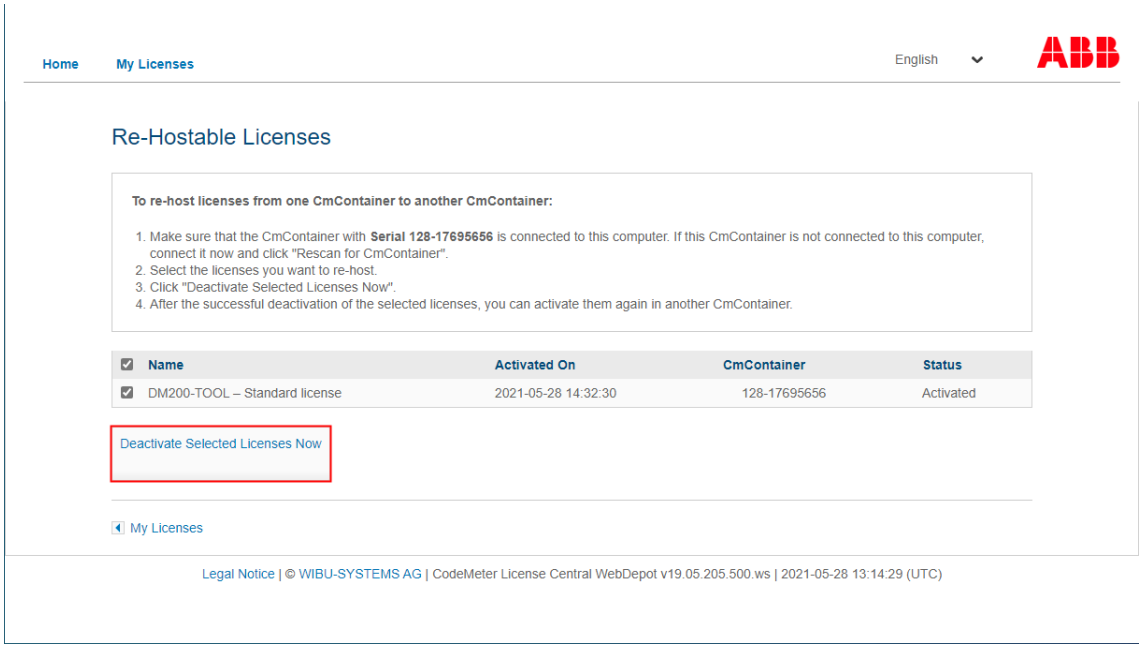


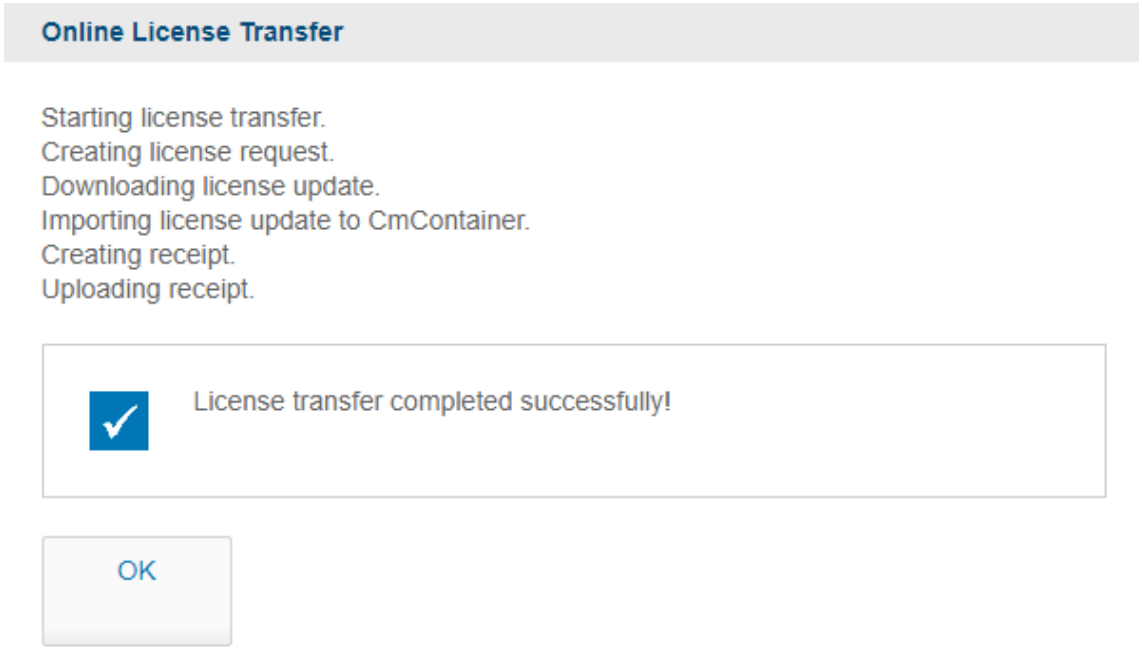
⇨ If the CmContainer is found, continue with Online licenses transfer ↯ *Chapter 1.2.4.9.3.1 “Online license transfer” on page 37*

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

Online license transfer

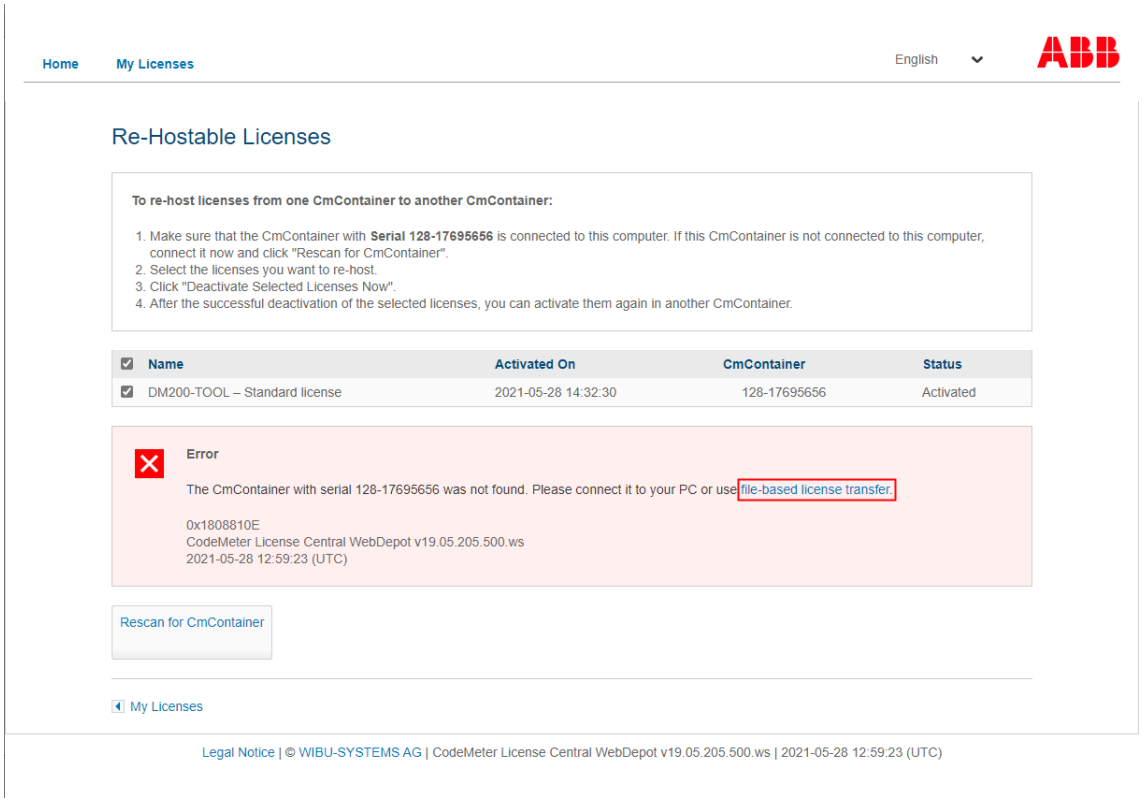
▷ Wait till the CmContainer is found, then select *“Deactivate Selected License Now”*


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

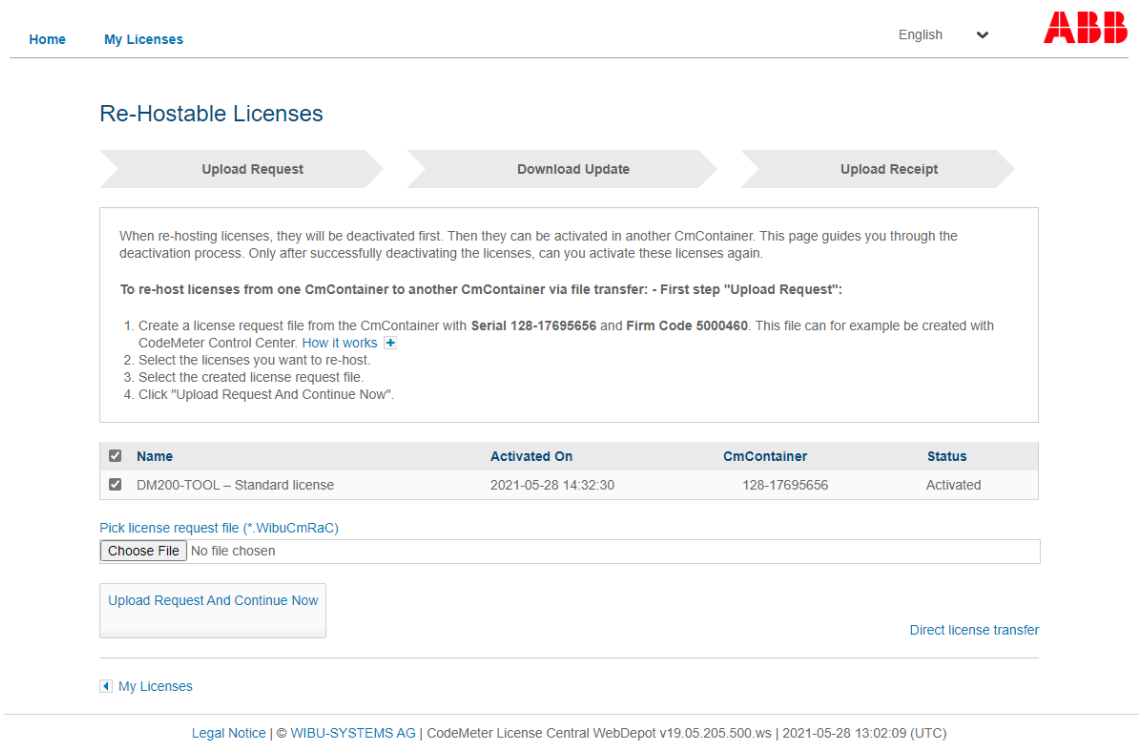

Online License Transfer
Starting license transfer.
Creating license request.
Downloading license update.
Importing license update to CmContainer.
Creating receipt.
Uploading receipt.
License transfer completed successfully!
OK

Offline license transfer

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

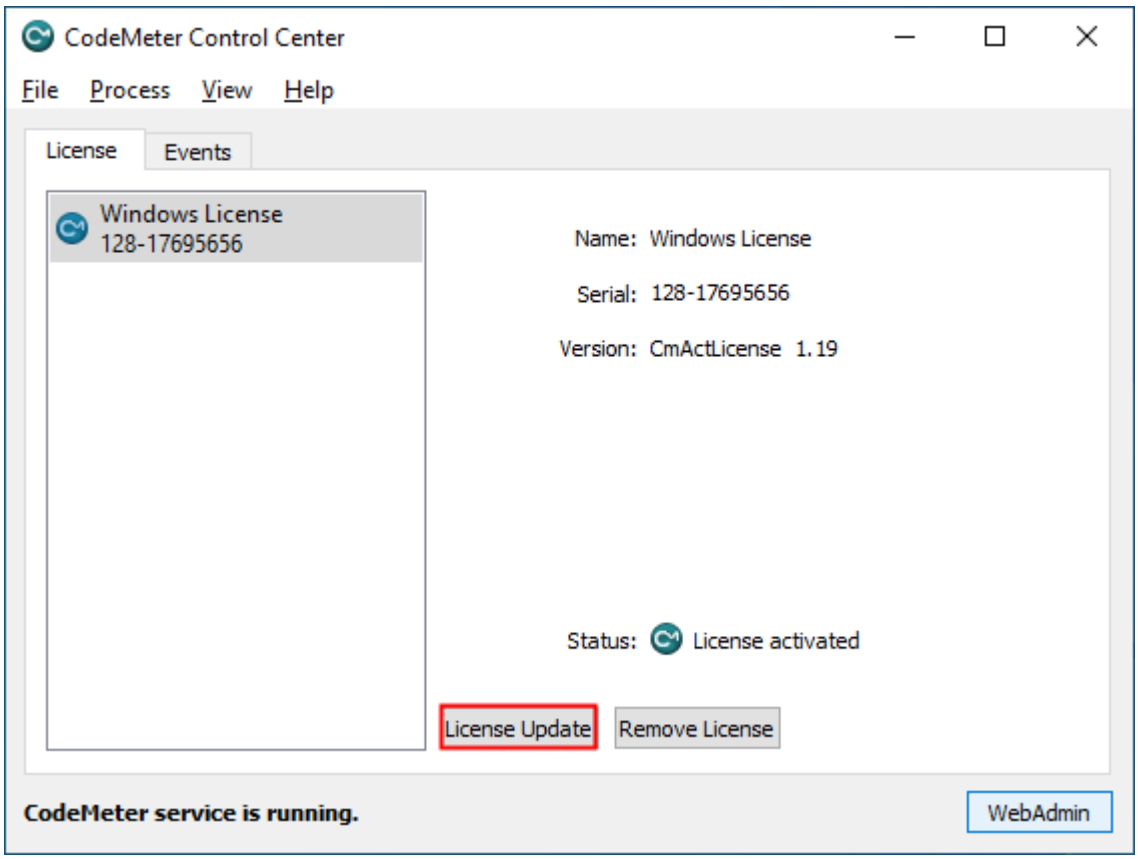


⇨ The following dialog opens

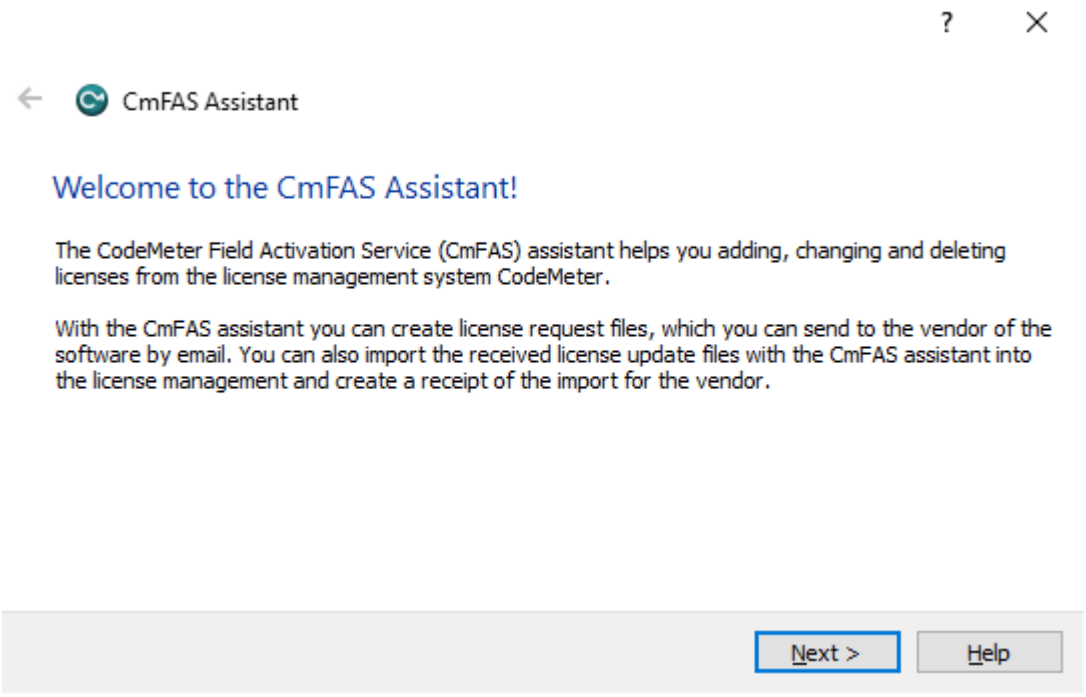


The instructions will lead you through the main steps of the offline license transfer:

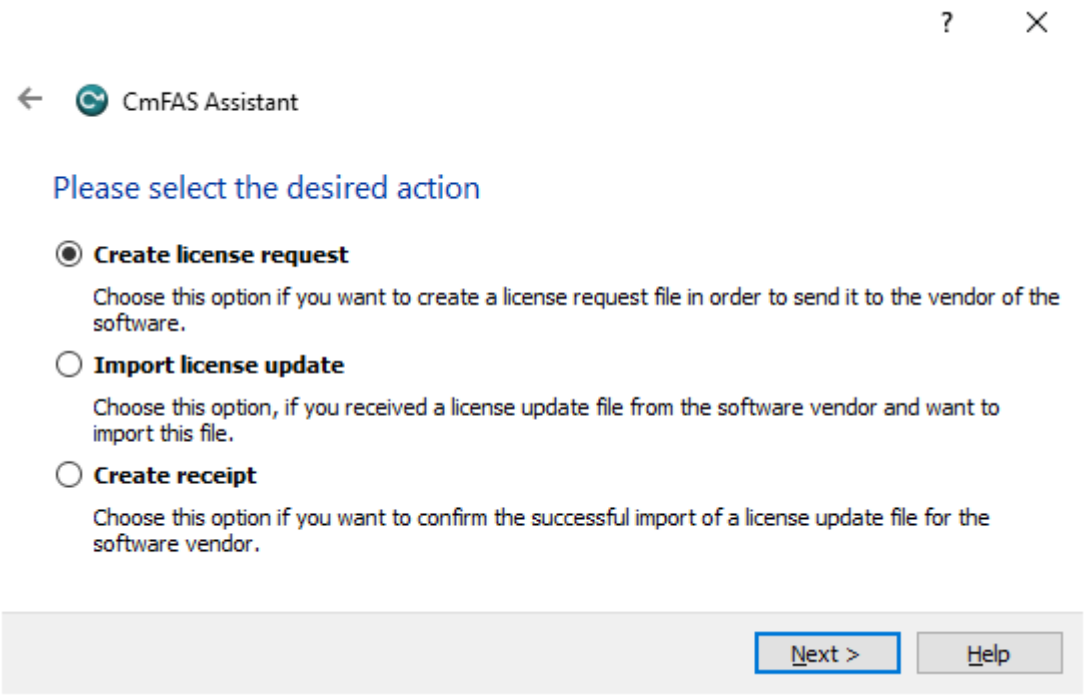
1. On the offline PC open the CodeMeter Control Center.
2. Select *“License Update”*.



⇨ The CmFAS Assistant opens.

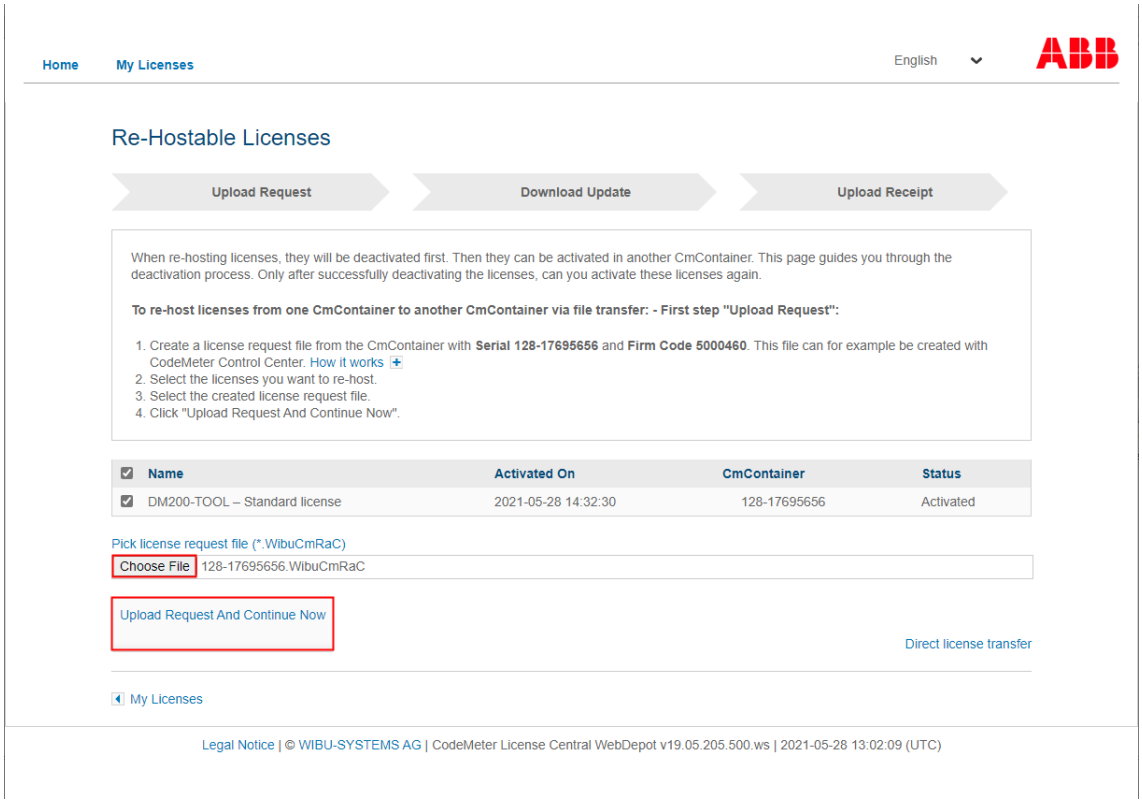


3. Select *“Create a license request file”*.

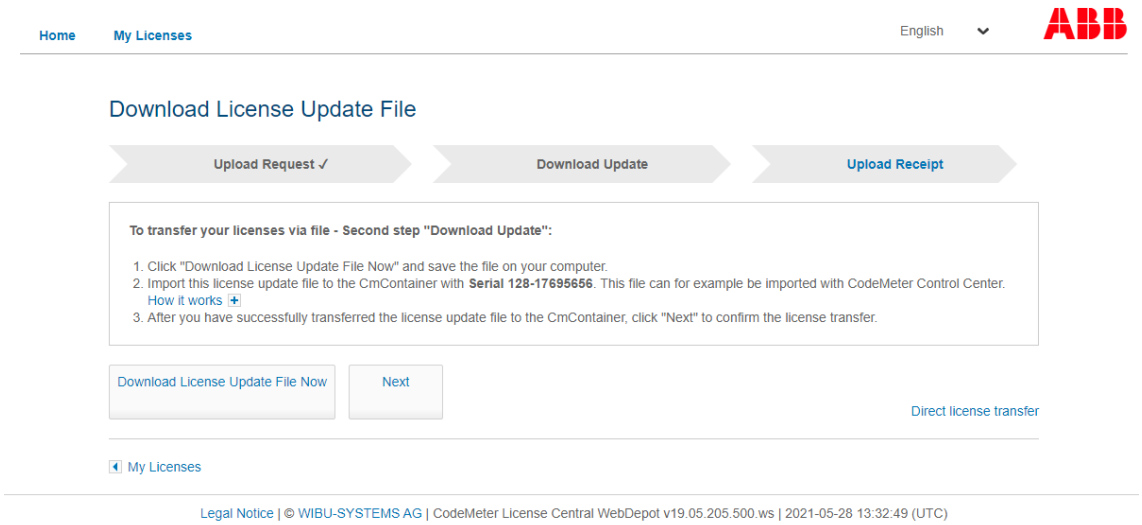


4. Select a location to store the license request file.
5. Transfer the license request file from the offline PC to an online PC.

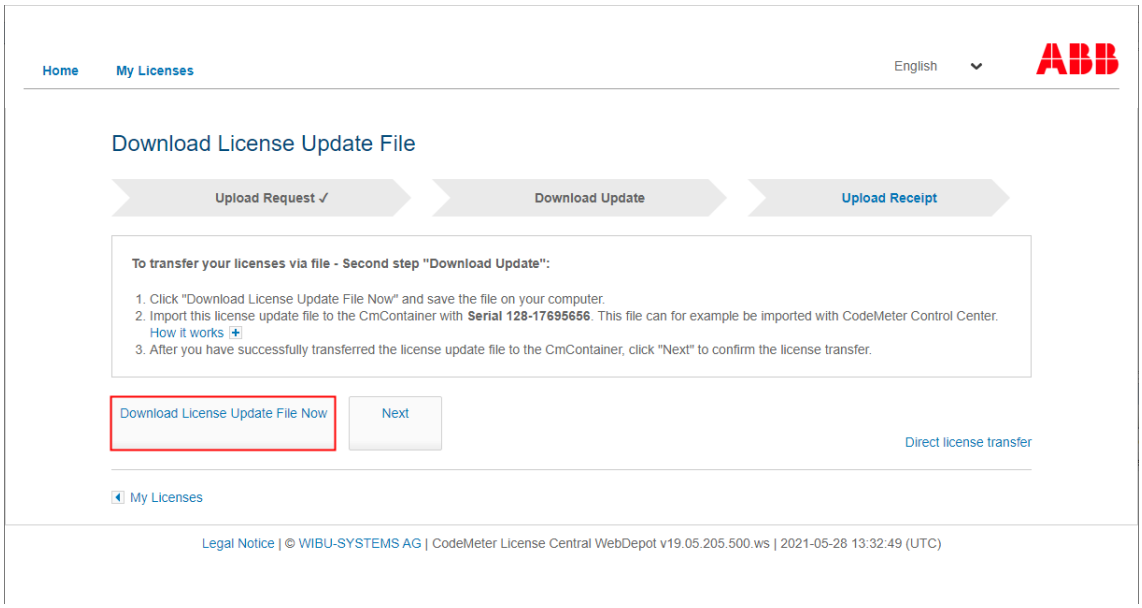
6. On the online PC choose the license request file and select *“Upload Request And Continue Now”*.



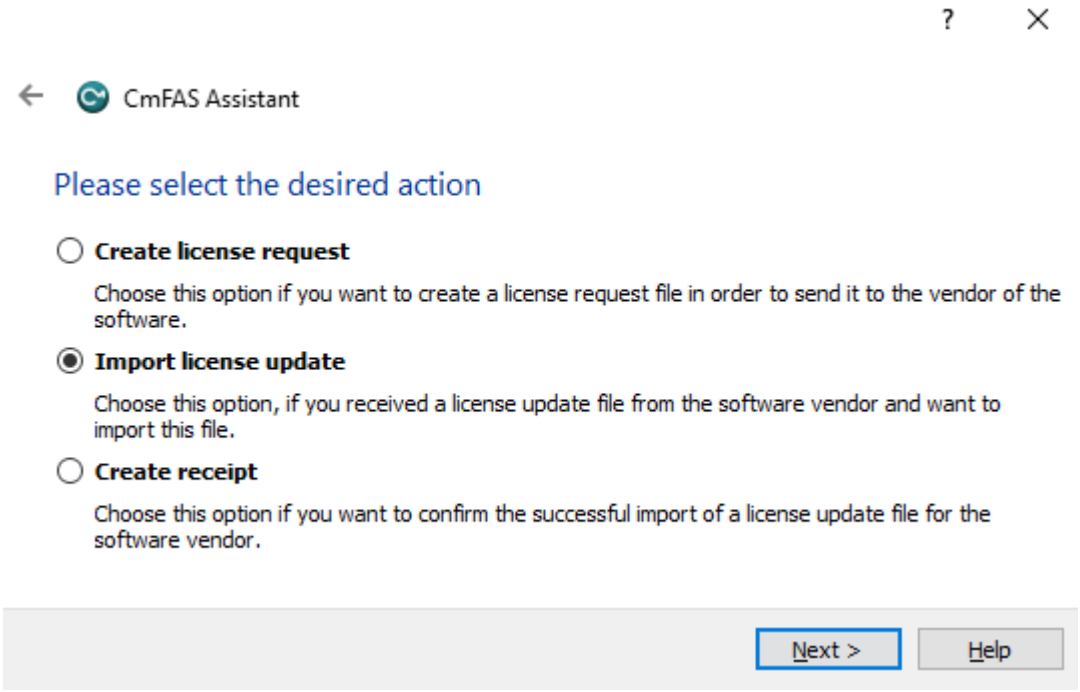
⇨ The next dialog is opened



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

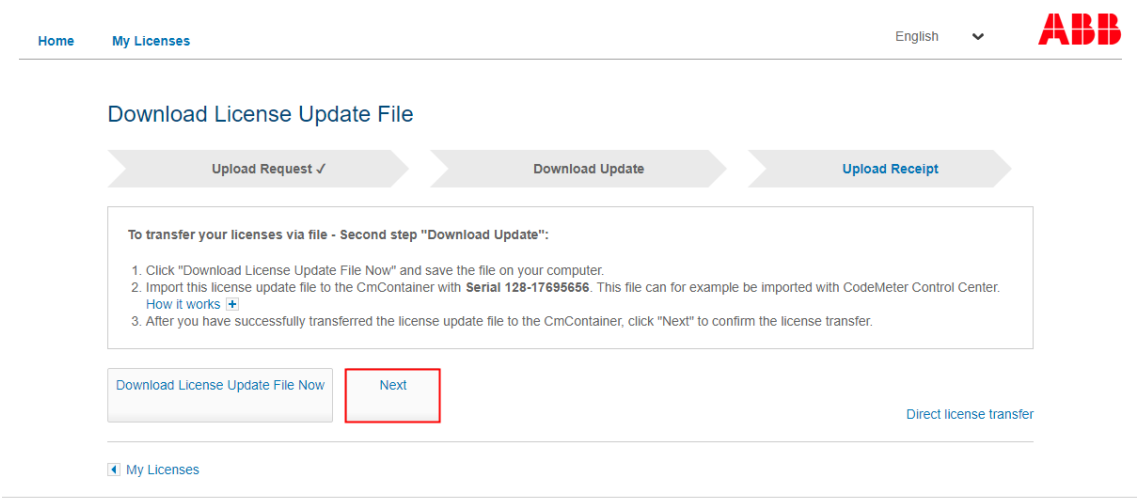


8. Save the license update file to a location on your computer.
9. Transfer the license upate file from the online PC to the offline PC.
10. On the offline PC open the CmFAS Assistant.
11. Select *“Import license update”*.

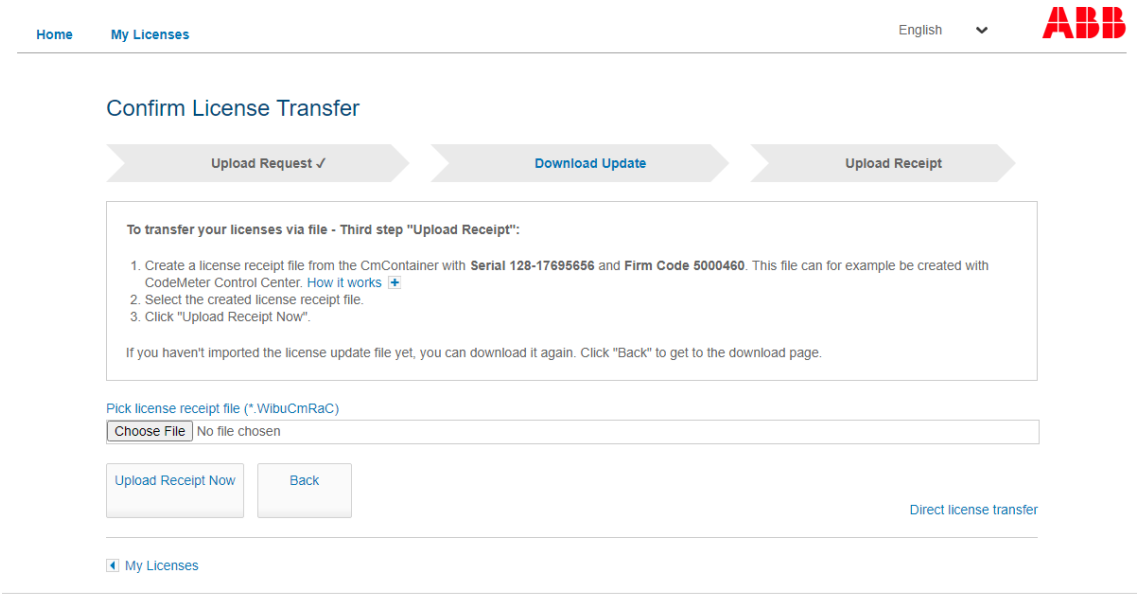


12. Select the license update file, to import the new license to the offline PC

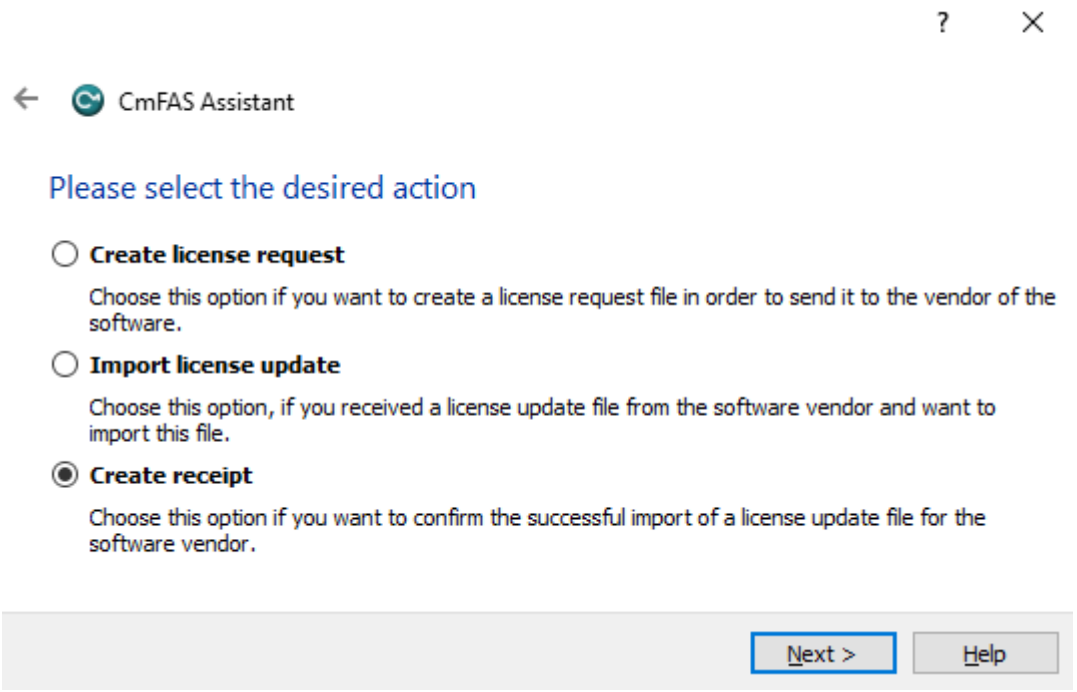
13. To confirm a succesful license transfer return to the online PC and select *“Next”*.



⇨ The last dialog is opened

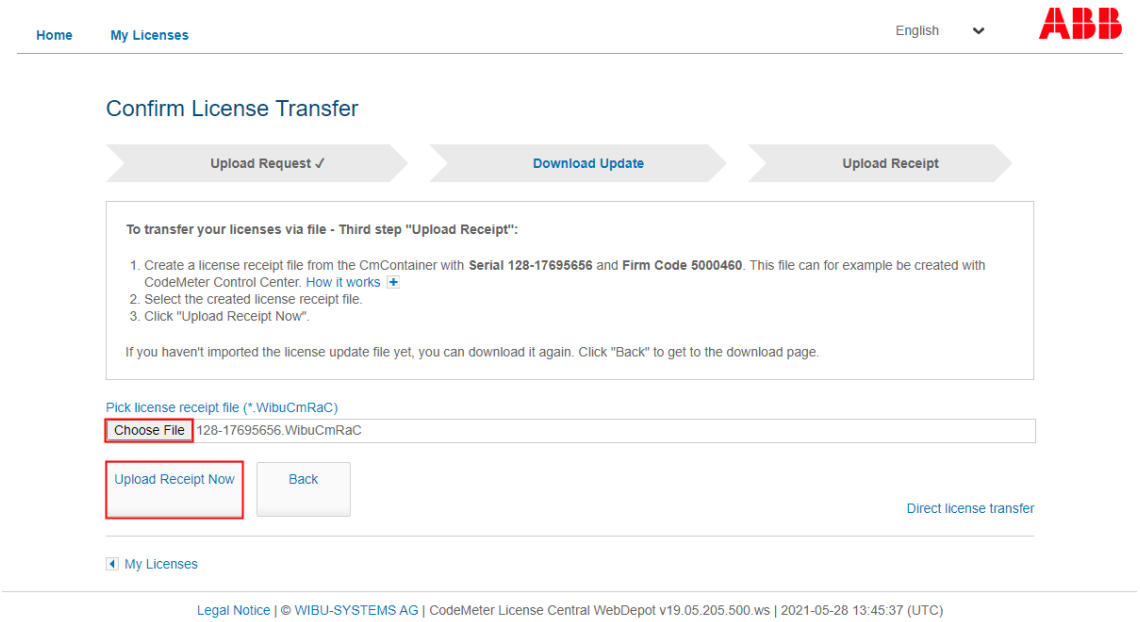


14. On the offline PC open the CmFAS Assistant.
15. Select *“Create receipt”*.

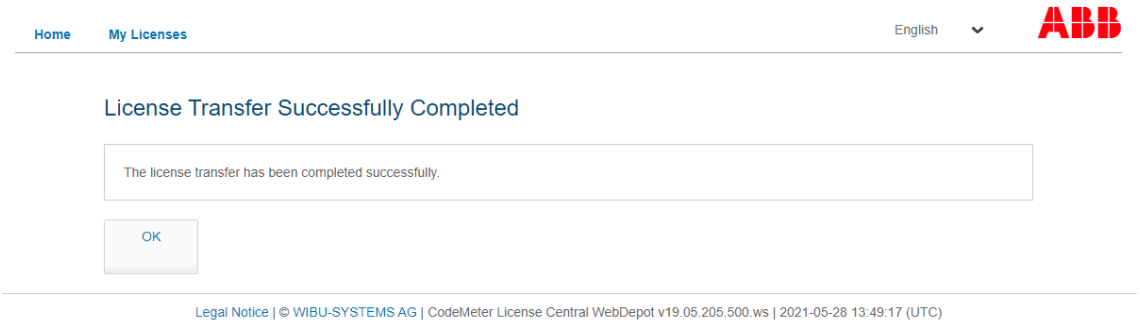


16. Choose a location to save the license receipt file.
17. Transfer the license receipt file from the offline PC to the online PC.

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



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



1.2.4.10 Generating license information file for support

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

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

1.2.4.10.1 Log files

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

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

C:\ProgramData\CodeMeter\Logs

This folder includes

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

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

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

1.2.5 Set-up communication parameters in Windows

Set-up communication parameters

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

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

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

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

Subnet mask should be 255.255.255.0.

Change the IP address

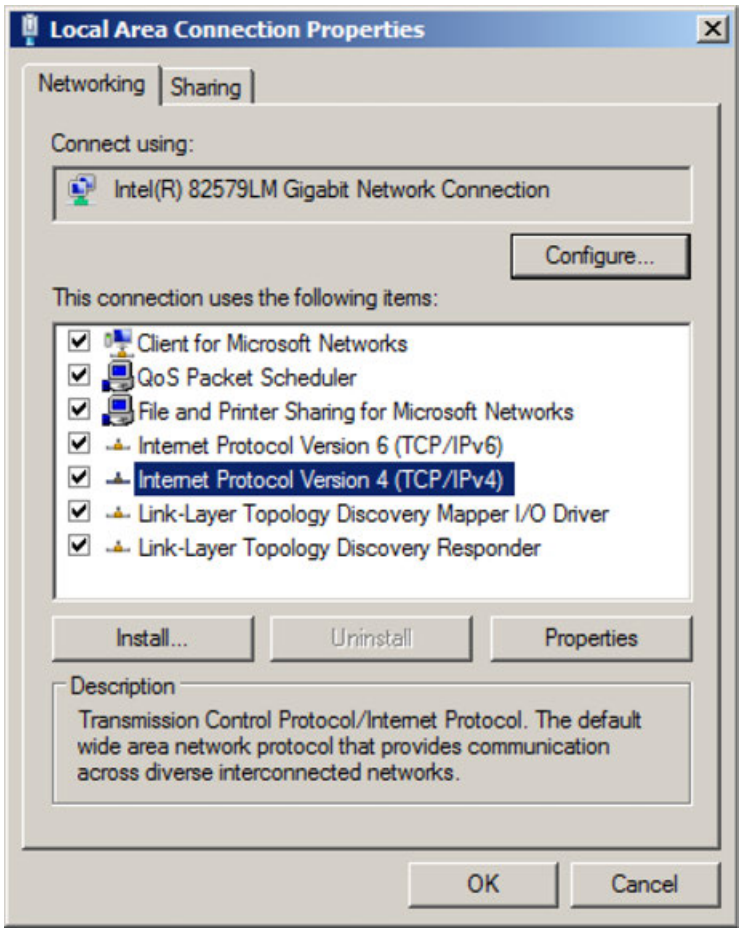
1. Open Windows **Control Panel**. Click *“Network and Internet ➔ Network and Sharing Center”*.
2. Click **Change adapter settings**.

   ⇨

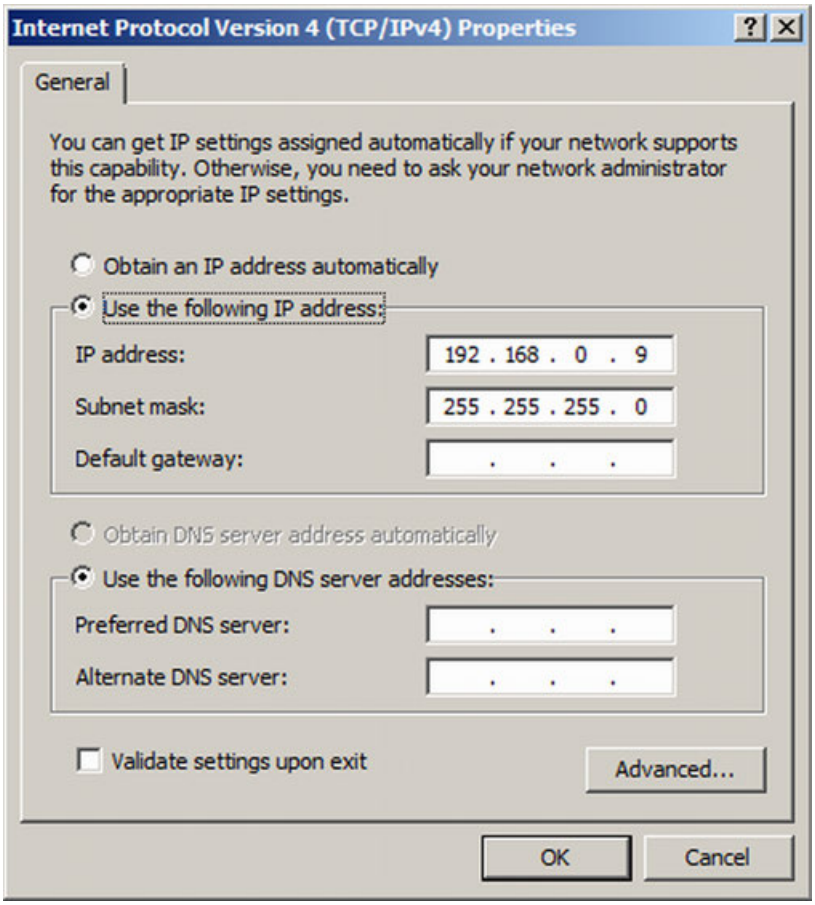
   

   If using existing network with several devices, please pay attention on given network rules or contact your system administrator.

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



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



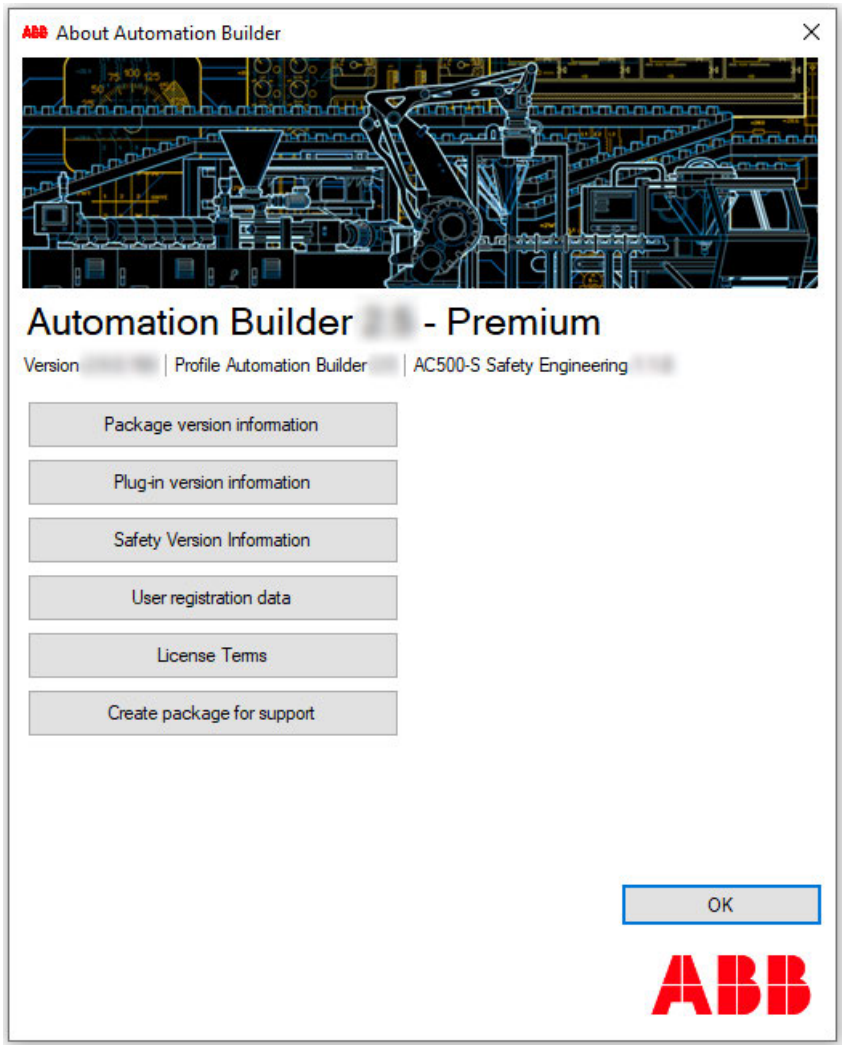
5. Enter your desired IP address and subnet mask.

1.2.6 Further information

Further information on the installed Automation Builder version such as installed packages or license terms can be found on the "About" page (help menu) and in ↳ *Chapter 1.4.1.20.4.13 “Dialog 'Options'” on page 1186*.

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

- Package version information: Further information about all installed package versions is shown.
- Plug-in version infromation: Further information about all installed plug-in versions is shown.
- Safety version infromation: Further information about all safety component versions is shown.
- User registration data: Enter or change your registration data.
- License Terms: Information about the license terms.
- Create package for support: Creates a package which can be saved or sent to support ↳ *Chapter 1.2.8 “Create log files for support” on page 50*.



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

See

- *Chapter 1.4.1.20.4.13 “Dialog 'Options'” on page 1186*

1.2.7 PLC runtime and demo licensing

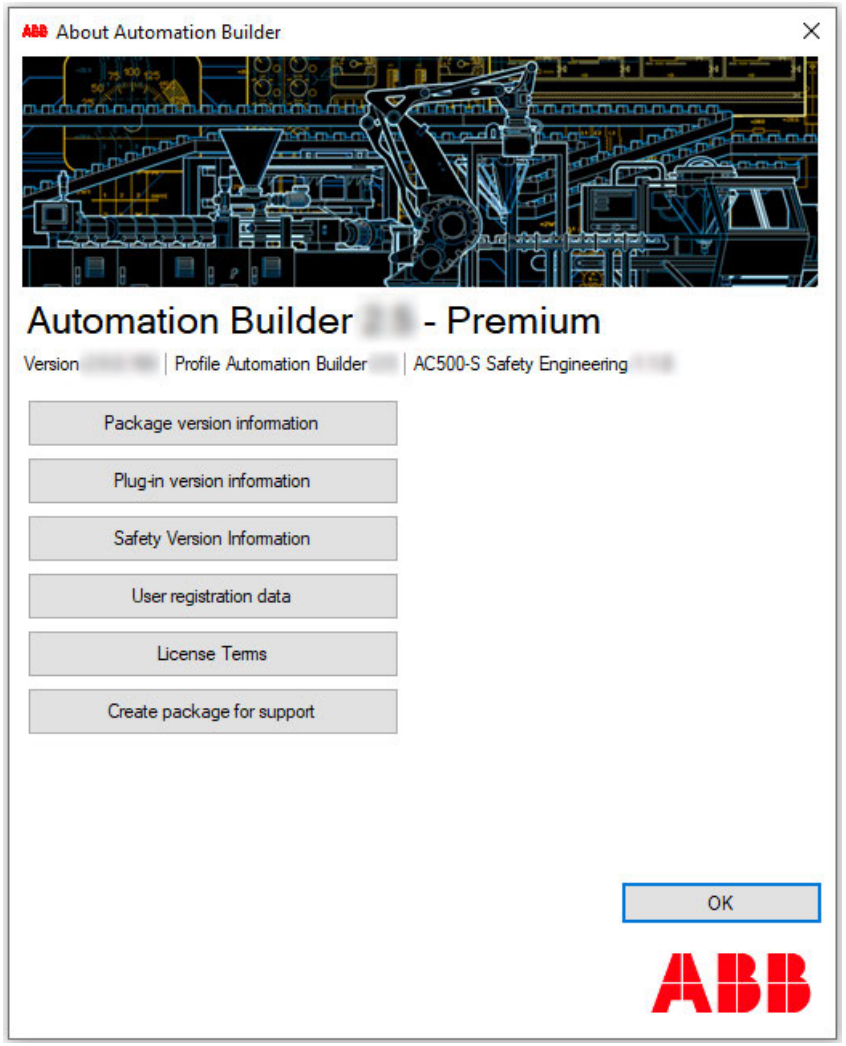
The use of some libraries and devices require the PLC to have a runtime license. Further it is possible to try out device features or library features by using a demo license *Chapter 1.6.6.2.2.2 “PLC runtime licensing” on page 3665*. .

1.2.8 Create log files for support

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

To collect this information proceed as follows:

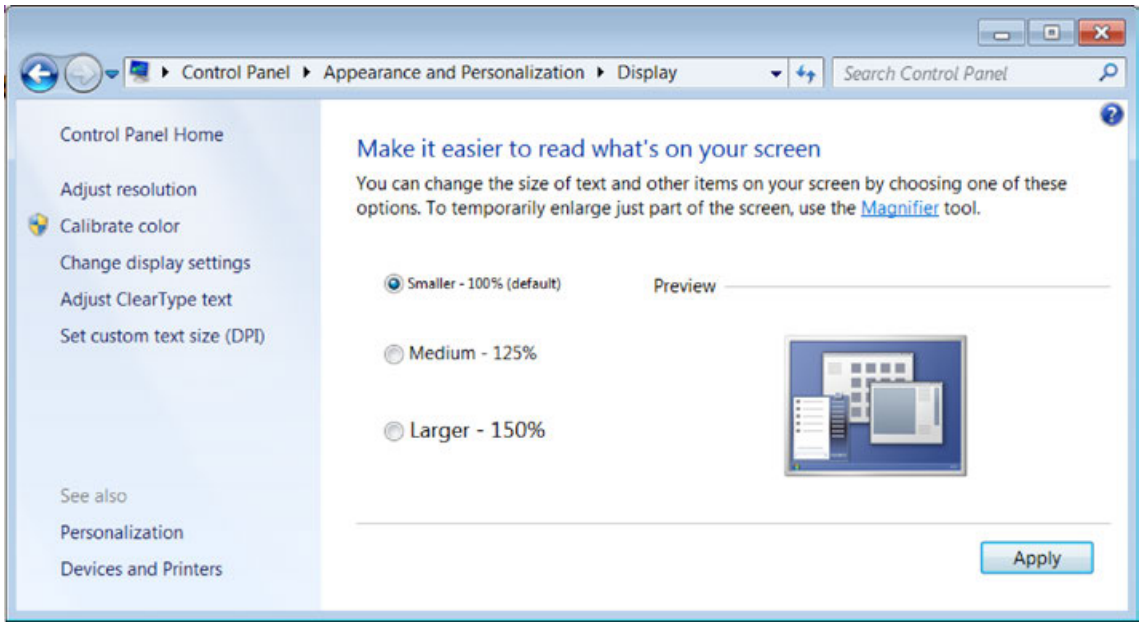
1. Click *"Help ➔ About"* in the main menu of Automation Builder.



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

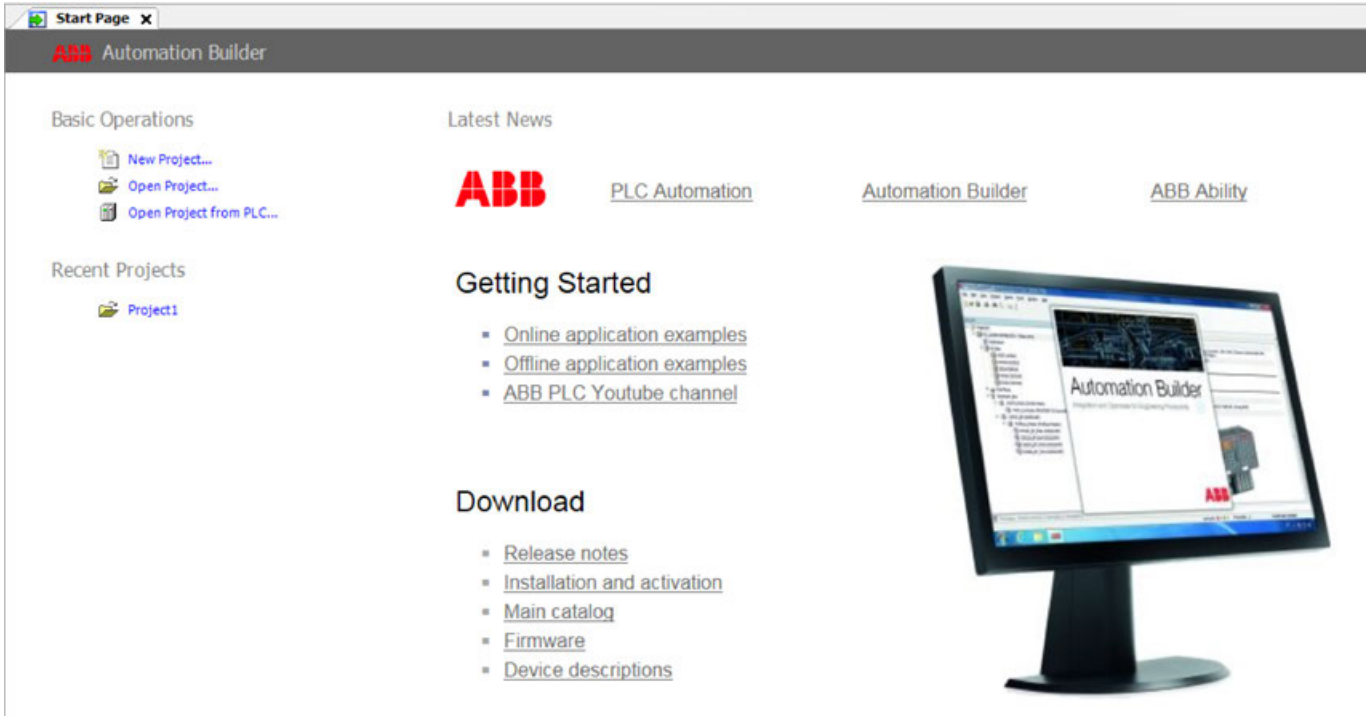
1.2.9 Menues, views, windows

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



1.2.9.1 Start page and menus

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



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

- *Chapter 1.4.1.20.3 “Menu Commands” on page 955*
- *Chapter 1.4.1.20.2 “Objects” on page 818*
- *Chapter 1.4.1.20.4 “Dialogs” on page 1149*

1.2.9.2 'All Messages' window

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

Chapter 1.4.1.20.3.3.5 “Command 'Messages'” on page 986

1.2.10 Device repository

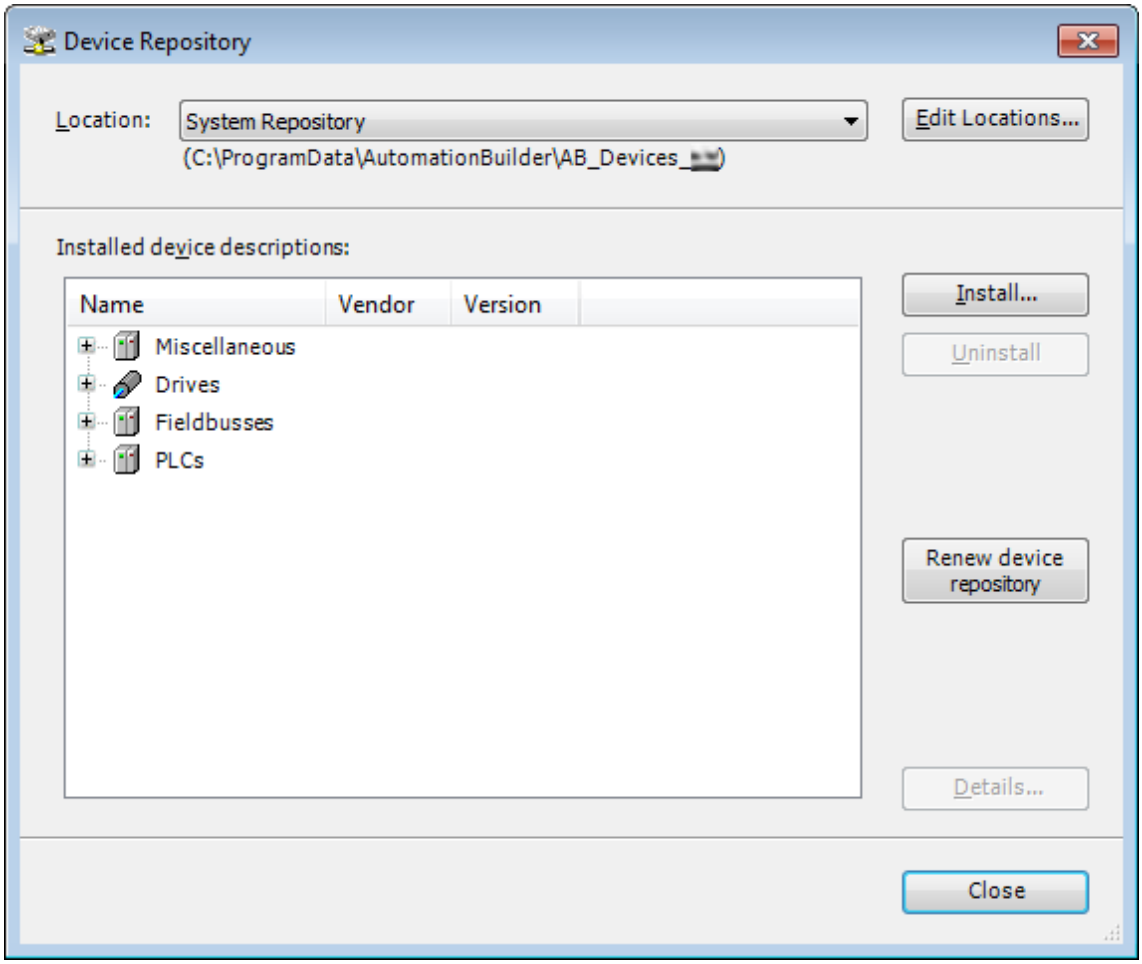
The Device Repository of Automation Builder manages the pool of devices that can be used in the PLC configuration.

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

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

Dialog device repository

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



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

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

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

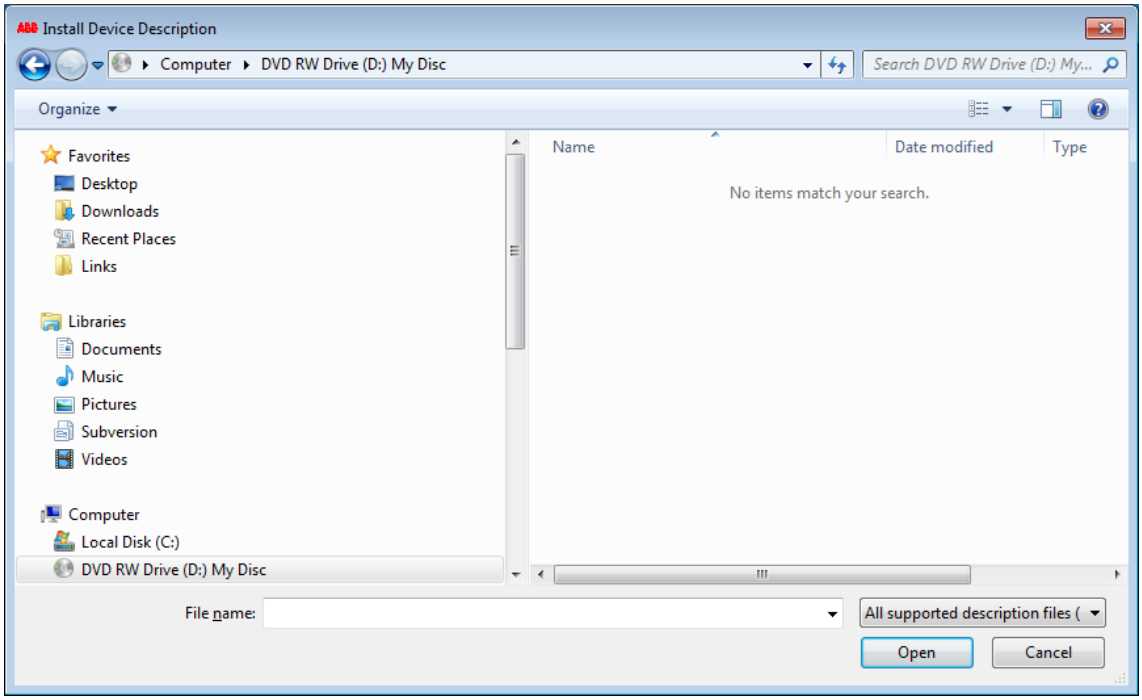
[Details]: Provides technical details on the selected device.

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

Installing devices

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

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



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

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

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

Uninstalling devices

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

The device is removed from the list.

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

1.2.11 Creating and configuring projects

What is a project?

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

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

See also

- *Chapter 1.4.1.20.2.1 "Object 'Application'" on page 819*
- *Chapter 1.4.1.20.2 "Objects" on page 818*
- *Chapter 1.4.1.20.4 "Dialogs" on page 1149*
- *Chapter 1.4.1.20.3.4.13 "Command 'Project information'" on page 1007*
- *Chapter 1.4.1.5 "Protecting and Saving Projects" on page 197*

Handling of AC500 projects such as project creation, export/import, comparison of projects etc. is described in the sections for AC500 V3 products.

Chapter 1.6.6.1.1 "Project handling" on page 3632

1.2.12 Handling of AC500 projects

Handling of AC500 projects such as project creation, export/import, comparison of projects etc. is described in the sections for AC500 V3 products.

Chapter 1.6.6.1.1 "Project handling" on page 3632

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

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

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

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

Project archive Automation Builder supports the creation and the import of project archive files. Archive files contain all relevant project data including the PLC configuration, the CODESYS project files and all device descriptions. This allows exchanging Automation Builder projects without taking care of the target environment General Settings. ↳ *Chapter 1.6.6.1 “General settings” on page 3631*

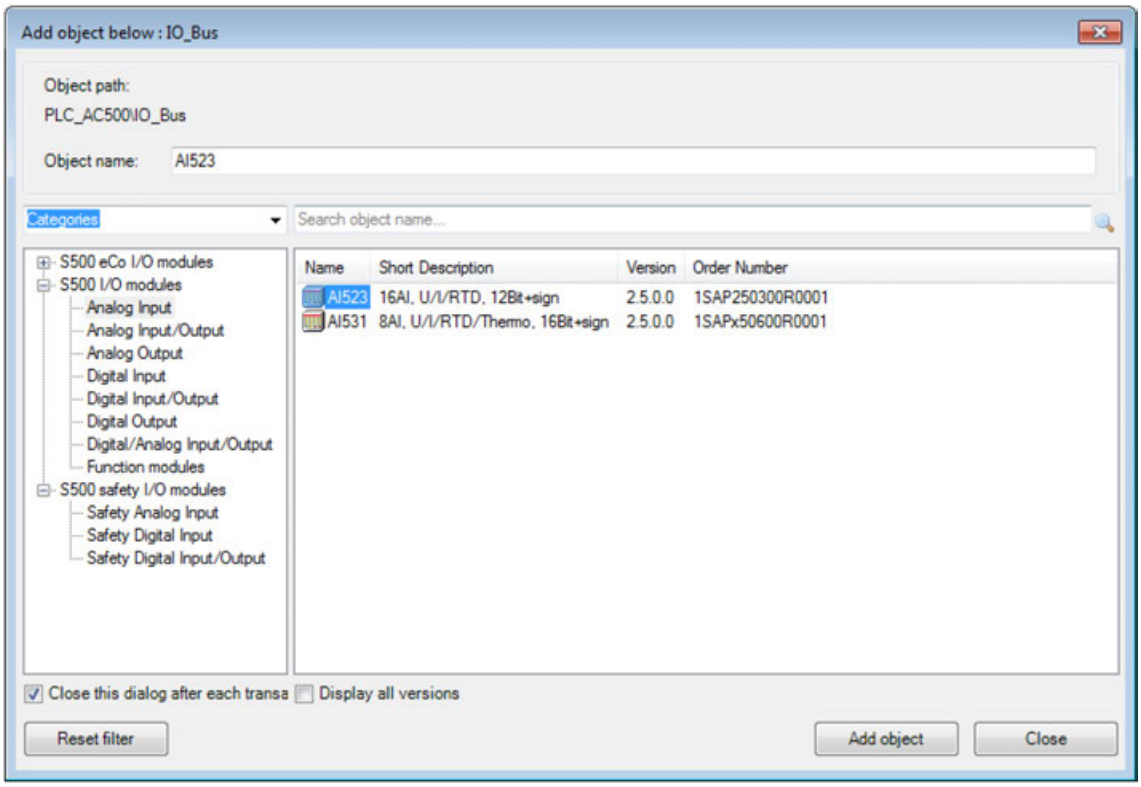
User and access rights of a project The 'User Management' provides functions for defining user accounts and configure the access rights within a project. The rights to access project objects via specified actions are assigned only to user groups, not to a single user account. So each user must be member of a group General Settings. ↳ *Chapter 1.6.6.1 “General settings” on page 3631*

1.2.13 Connection of devices

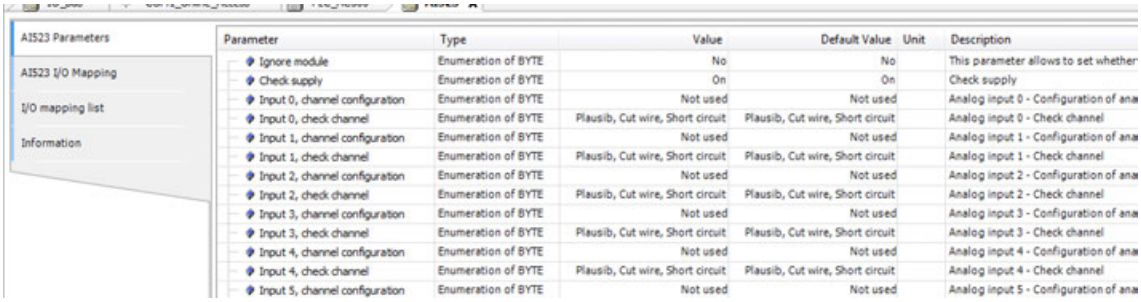
1.2.13.1 Configuring devices

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

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



2. Select the desired object and click *[Add object]*.
3. Double-click the new object in the device tree to configure the device settings. Depending on the selected item different configuration tabs are available.



1.2.13.2 Symbolic names for variables, inputs and outputs

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

Input and output mapping

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

The corresponding variable declarations are automatically available in the project.

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

Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
b_Input_IB0		Digital inputs I0 - I7	%IB0	BYTE			
x_Input_IB0_IX0		Digital input I0	%IX0.0	BOOL			
x_Input_IB0_IX1		Digital input I1	%IX0.1	BOOL			
x_Input_IB0_IX2		Digital input I2	%IX0.2	BOOL			
x_Input_IB0_IX3		Digital input I3	%IX0.3	BOOL			
x_Input_IB0_IX4		Digital input I4	%IX0.4	BOOL			
x_Input_IB0_IX5		Digital input I5	%IX0.5	BOOL			
x_Input_IB0_IX6		Digital input I6	%IX0.6	BOOL			
x_Input_IB0_IX7		Digital input I7	%IX0.7	BOOL			
		Digital inputs I8 - I15	%IB1	BYTE			
		Digital inputs I16 - I23	%IB2	BYTE			
		Digital inputs I24 - I31	%IB3	BYTE			
Fast counter							

AC500 uses Intel Byte Order (Little Endian).

Only entries with a data type set in column "Type" can be mapped. These entries can be expanded to show the available I/O channels.

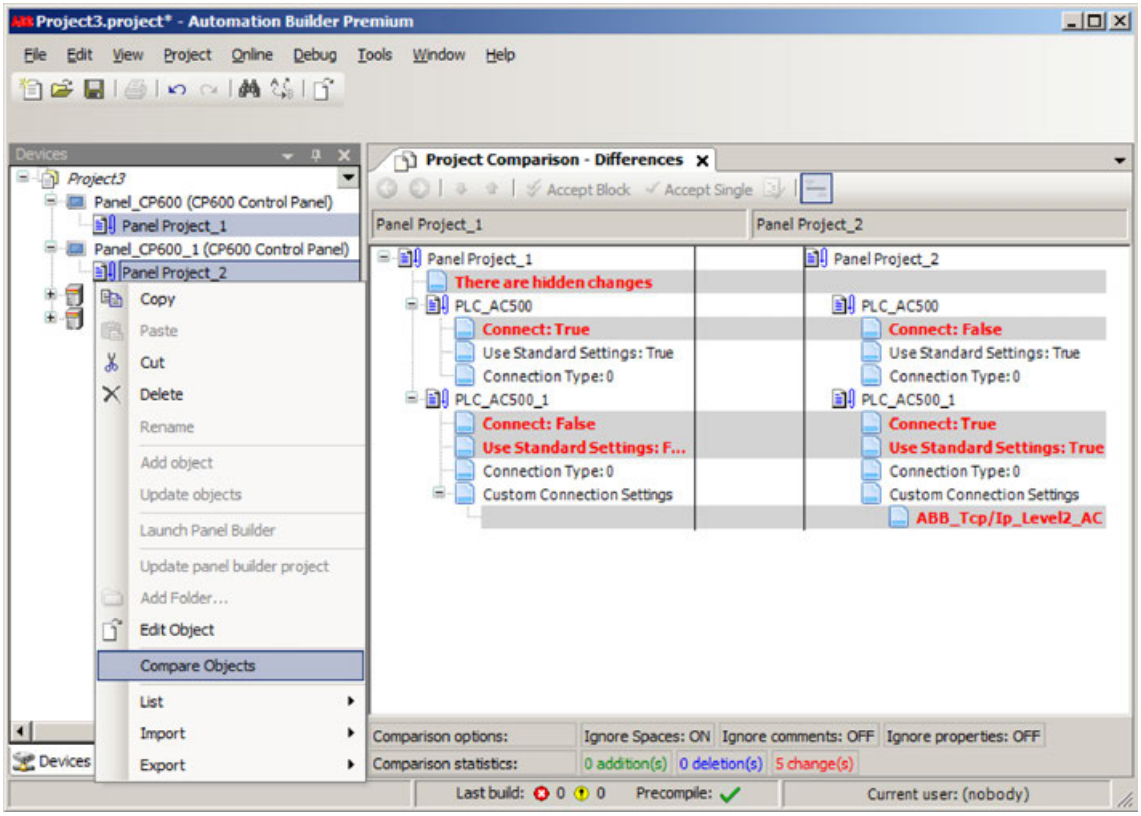
If the project has been imported from a previous Automation Builder version, all variables should be checked to avoid inconsistencies concerning the I/O mapping.

1.2.13.3 Update of AC500 devices

Perform a firmware update to update AC500 V3 devices. *Chapter 1.2.18.1.2.6 “ AC500 V3 firmware installation and update” on page 87*

1.2.13.4 Comparing objects

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



1.2.14 Connection of serial interfaces

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

AC500 V3 Products: *Chapter 1.6.6.2 “PLC devices and components” on page 3662*

1.2.14.1 Programming of applications

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

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

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

1.2.15 I/O mapping

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

Chapter 1.6.6.2.13.8 “I/O mapping list” on page 3777

1.2.16 AC500 PLC configuration

See Getting Started for AC500 V3 products. *Chapter 1.6.6.2.2 “PLC start-up” on page 3665*

1.2.17 Converting an AC500 V2 project to an AC500 V3 project

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

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

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

1.2.18 Example projects

1.2.18.1 Example projects for AC500 V3

1.2.18.1.1 Hardware AC500 V3

Configuration for example projects

The example projects require a small PLC configuration with I/O devices, e.g., as available in the training case TA5450-CASE. *https://to.abb/AfO9-ftT*

Table 2: Modules for example projects to get started with AC500 V3 PLC

Product name	Type	First project ↳ *Chapter 1.2.18.1.2 “Example project for central I/O expansion” on page 63*	Second project ↳ *Chapter 1.2.18.1.3 “Example project for remote I/O expansion with PROFINET” on page 109*
PM5630- 2ETH	AC500 V3 CPU	x	x
TB5620-2ETH	terminal base for CPU	x	x
DA501	analog/digital mixed input/output (I/O) module	x	x
TU516-H	terminal unit for I/O module	x	x
CM579-PNIO	PROFINET communication module	--	x
CI502-PNIO	PROFINET communication interface module	--	x
TU508-ETH	terminal unit for communication interface module	--	x
TA524	blind cap for terminal base	x	x

Connections

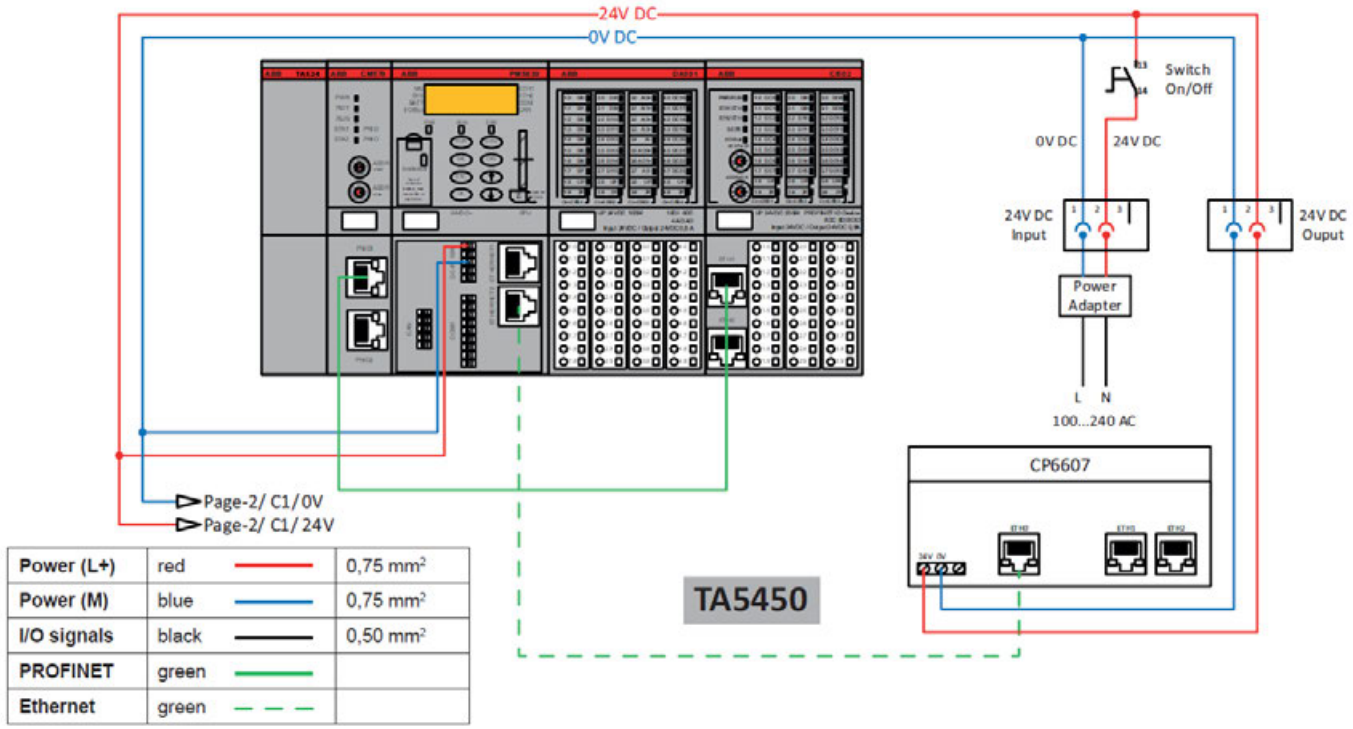


Fig. 5: Training case TA5450

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

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

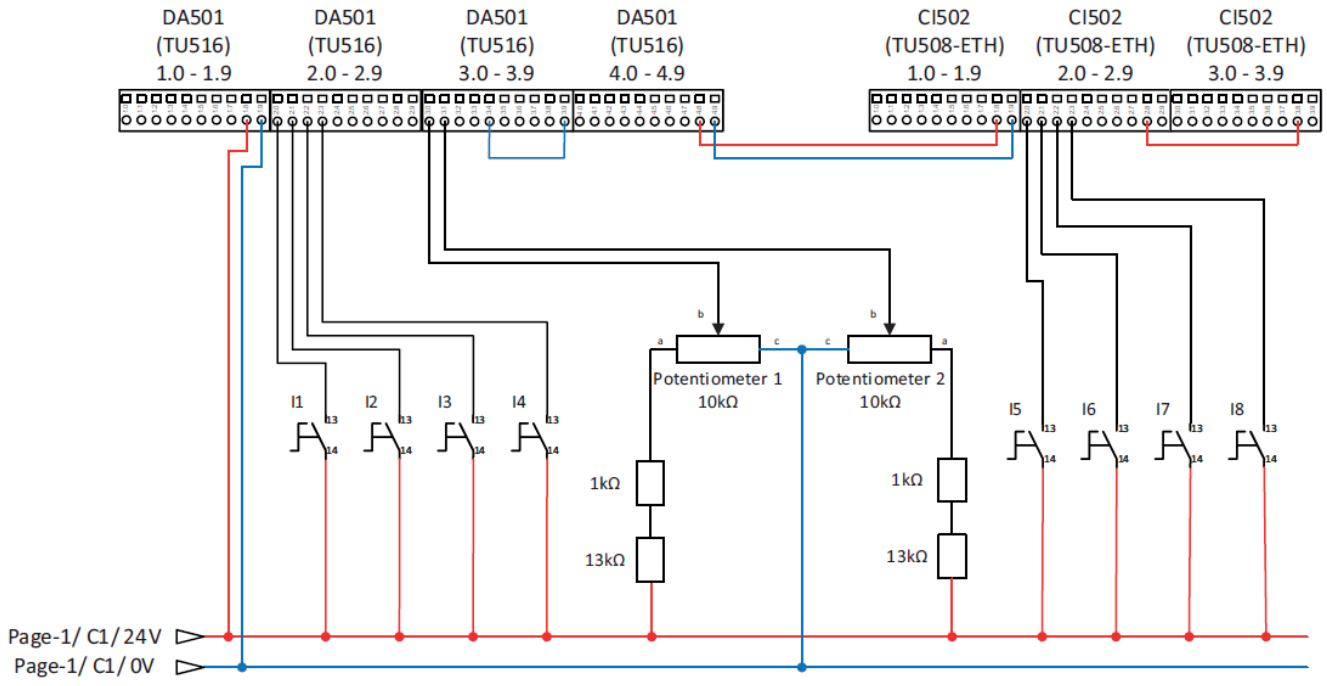


Fig. 6: Wiring of training case

For the example projects, not all input switches and none of the potentiometers included in training case are necessary.

You will need switch I1 for the example project for central I/O expansion.

You will need switch I5 for the example project for remote I/O expansion.

System assembly, construction and connection

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

You can mount AC500 PLC either to DIN rail or to a metal plate *Chapter 1.6.4.6.3 “Mounting and demounting” on page 3408*. Here, we recommend to mount on DIN rail.

1. Snap the terminal base onto DIN rail.
2. Snap the additional terminal units for I/O modules onto DIN rail.
3. Make the sensor/actuator wire connections according to the dedicated electronic module you want to use. Provide external process power supply as required.
4. If required, make the fieldbus connections according to the dedicated master communication module you want to use.
5. Plug the appropriate electronic and I/O modules in the correct locations (processor module, communication modules on terminal base, and eventually also communication interface modules and I/O modules onto dedicated terminal units).
6. Connect a programming cable (Ethernet cable between ETH port of CPU and PC with engineering software).

1.2.18.1.2 Example project for central I/O expansion

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

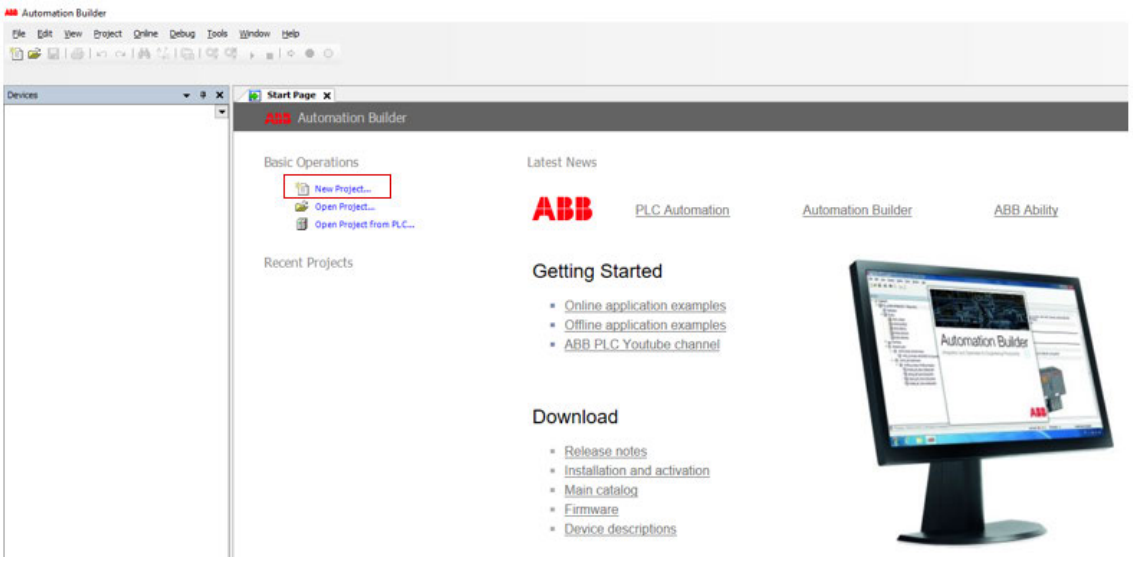
Preconditions

- Automation Builder is installed and licensed as, at least, basic edition *Chapter 1.2.4 “Managing your licenses” on page 20*.
- AC500 V3 CPU is assembled and connected to the PC *Chapter 1.2.18.1.1 “Hardware AC500 V3” on page 61*.

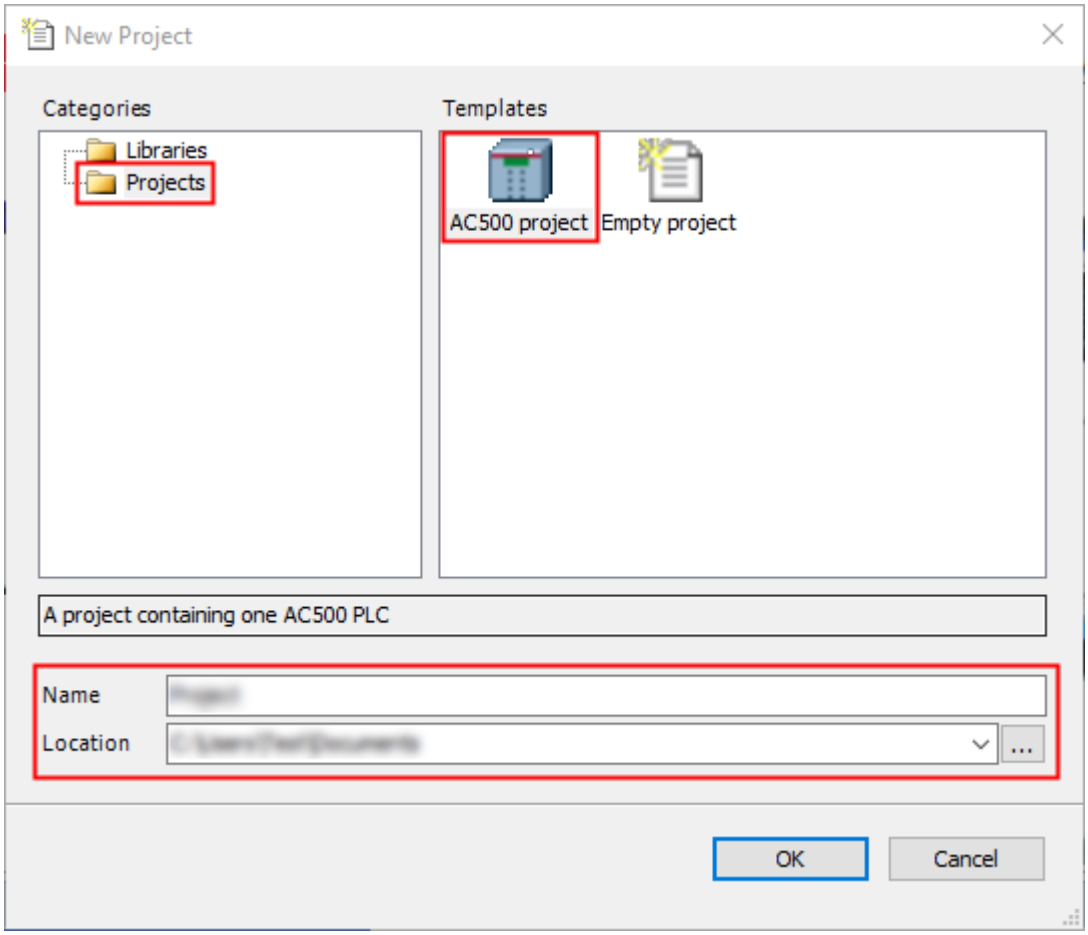
Create, set-up and save your AC500 V3 project

Create a project

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

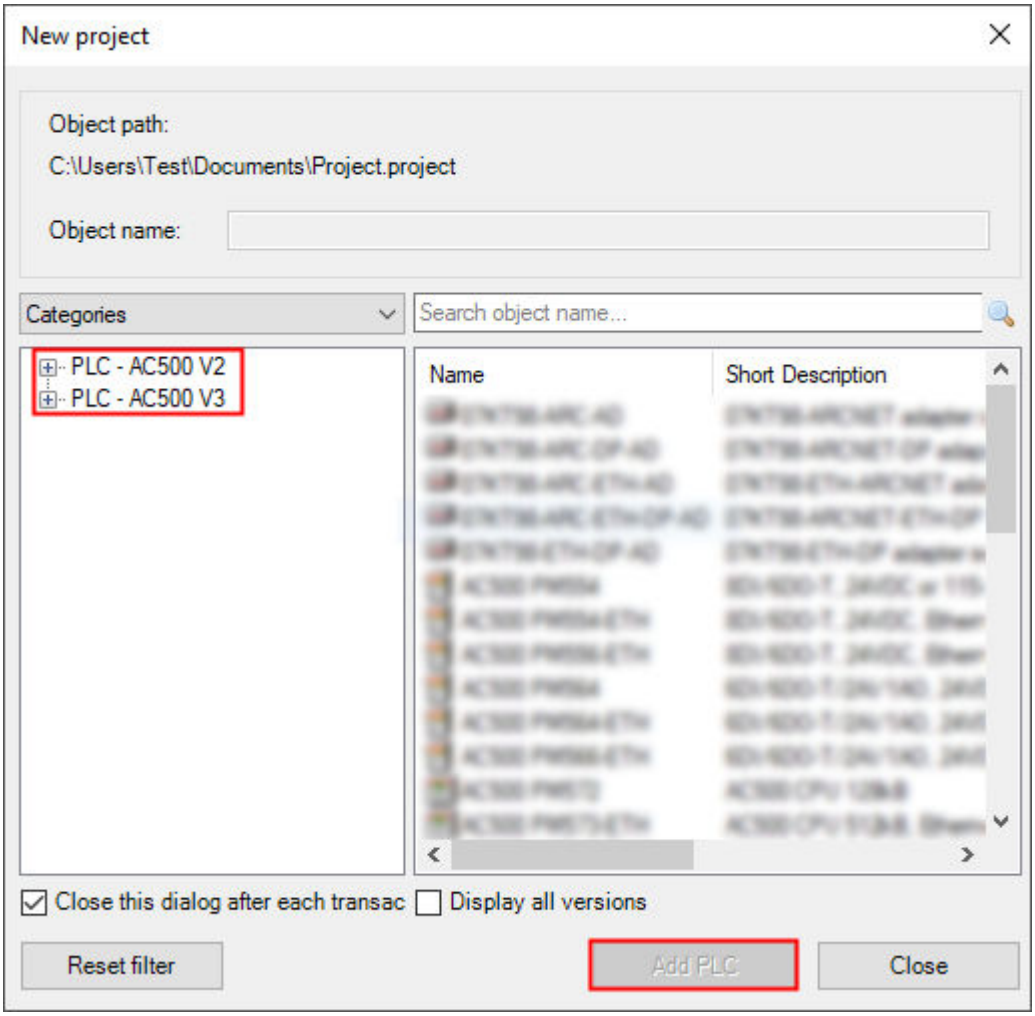


2. Select *"New Project"* or go to menu *"File ➔ New Project"*.



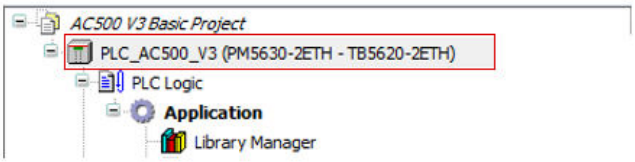
3. Select *"Projects"*.
4. Select *"AC500 project"*.
5. Fill in project name.
6. Choose a location to save the project to.

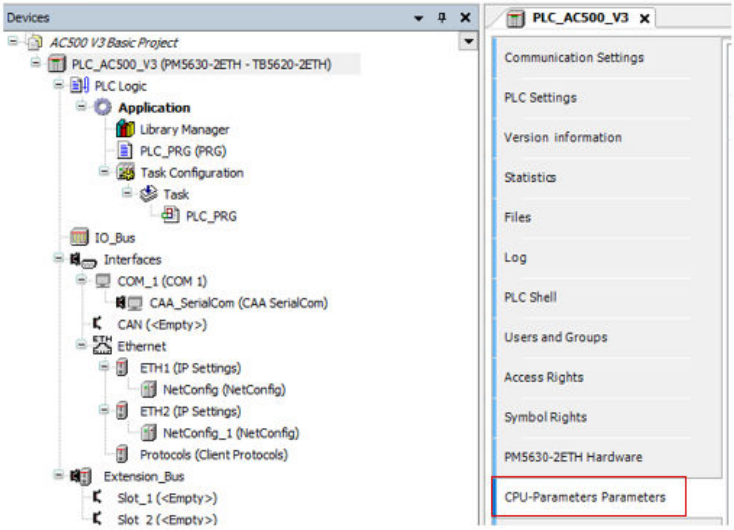
7. Select *“OK”*.
8. Select *“PLC - AC500 V3”*.
9. Select the CPU according to your hardware set-up.



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

Configure your CPU



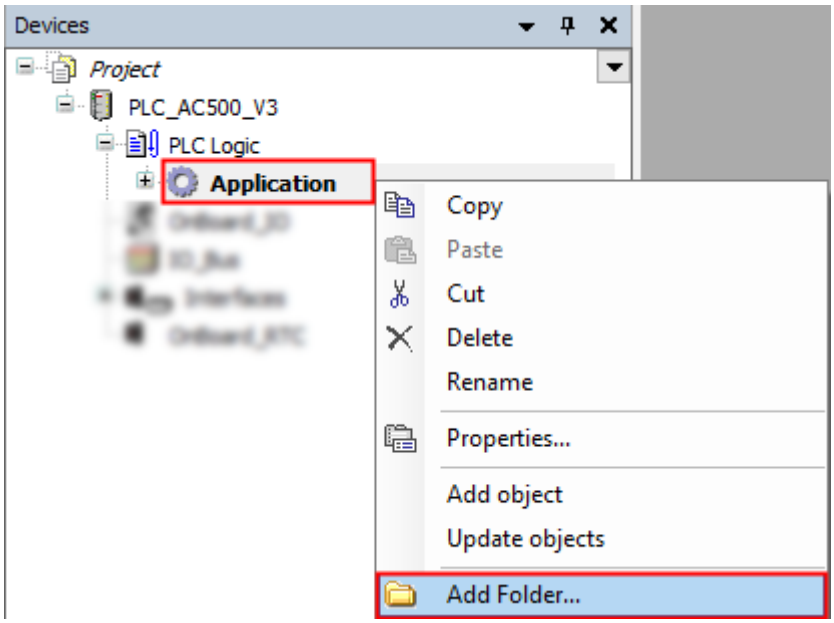


Parameter	Type	Value	Default Value
Error LED	Enumeration of BYTE	On	On
Check battery	Enumeration of BYTE	On	On
Stop on error class	Enumeration of BYTE	Diagnosis of at least error class 2	Diagnosis of at least error class 2

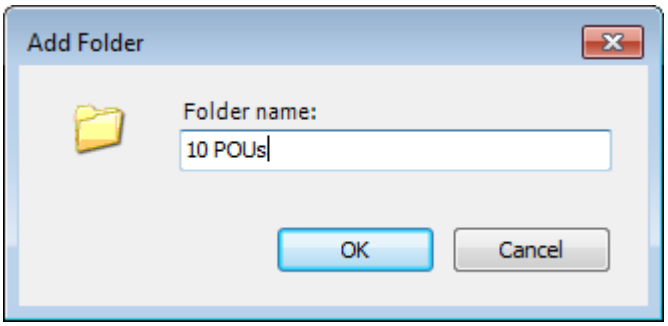
1. Double-click *“PLC_AC500_V3”*.
   ⇨ A tab opens in the editor view.
2. Select *“CPU-Parameters Parameters”*.
3. Under parameter *“Check battery”*, choose the value *“Off”* since there is no battery present inside the CPU module.
4. Keep the default values for all other parameters.

Create folders in the device tree

To optimize the project readability, you will create different folders to group similar objects. The folder names are exemplary. Because the device tree view follows an alphabetical order, we use number prefixes to determine the order.



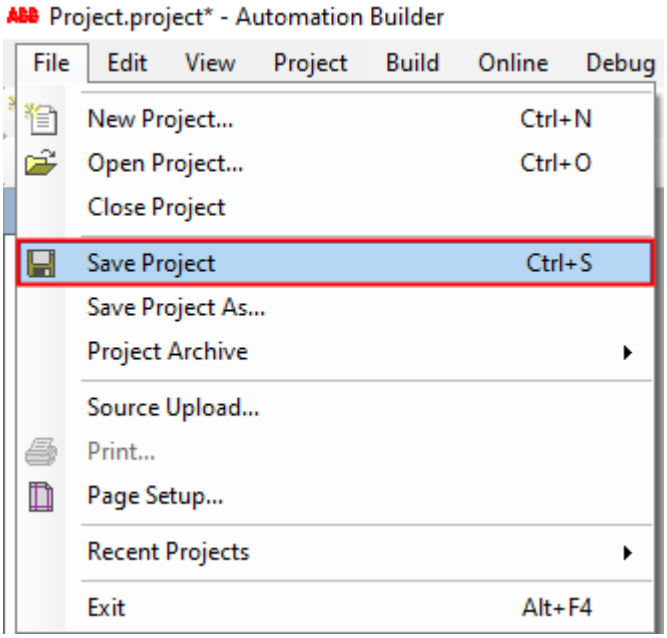
1. Right-click *“Application”*.
2. Select *“Add Folder”*.



3. Type in "10 POUs". This is a name example. Here, the intention is to see this folder as a last one.

   The folder "10 POUs" is for program organization units (POU). POUs are objects of type program, function or function block that are used to create a user program.

Save the project



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

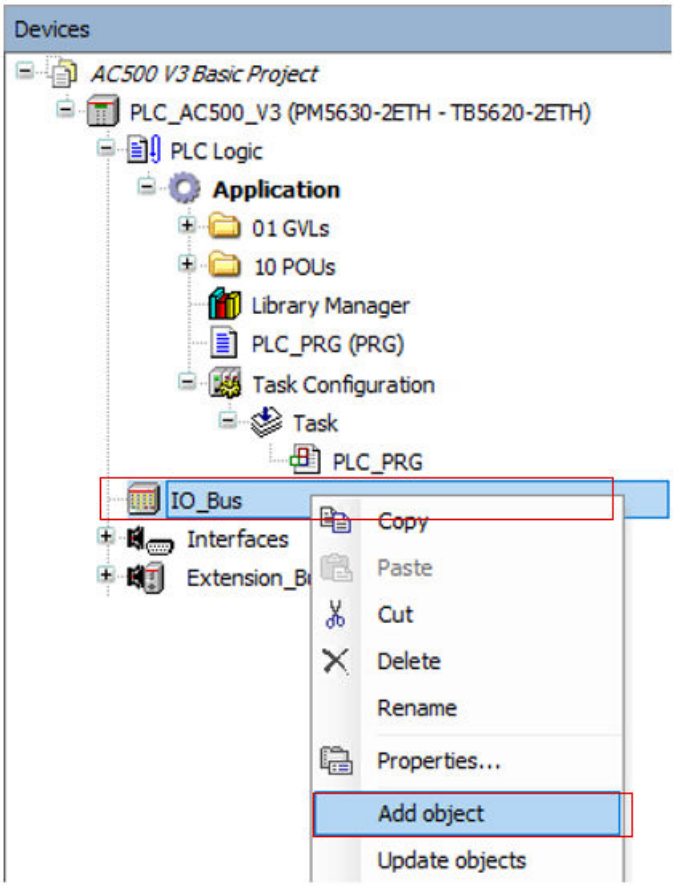
Alternatively, select the save icon in the tool bar.

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

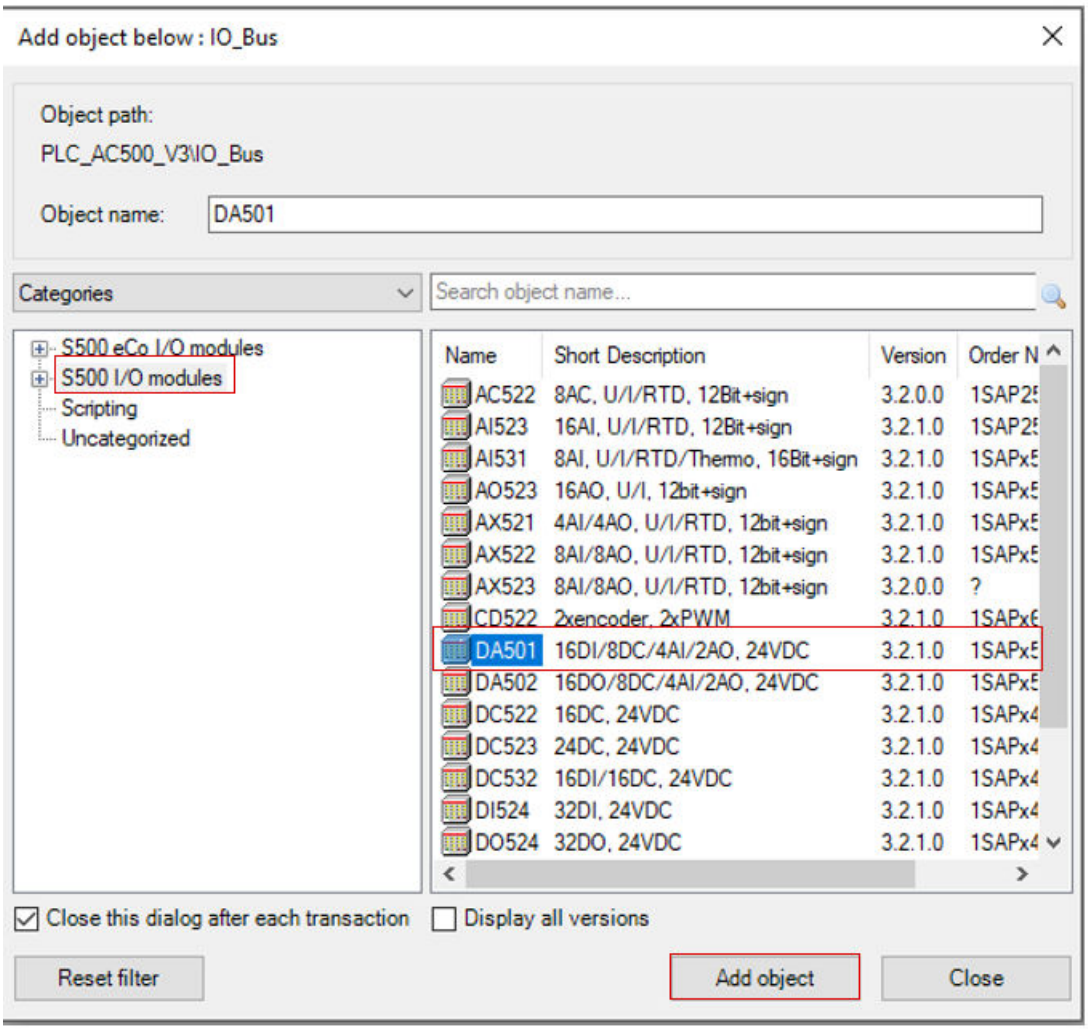
Configure the I/O module

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

Add an I/O bus module

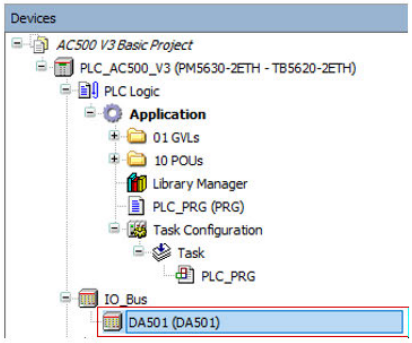


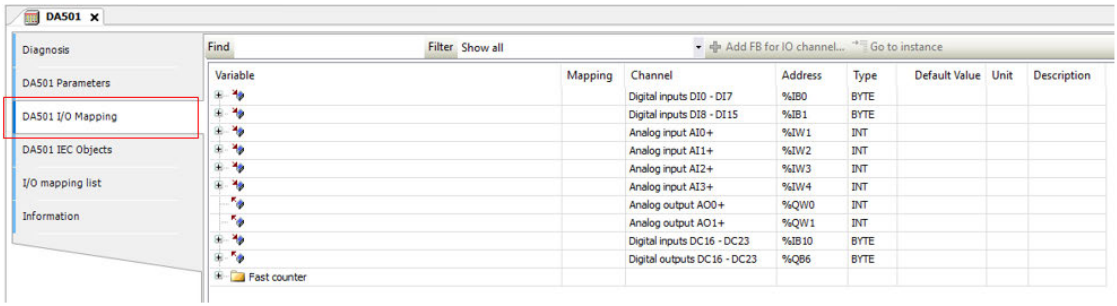
1. Right-click *"IO_Bus"* in the device tree.
2. Select *"Add object"*.



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

DA501 variable mapping

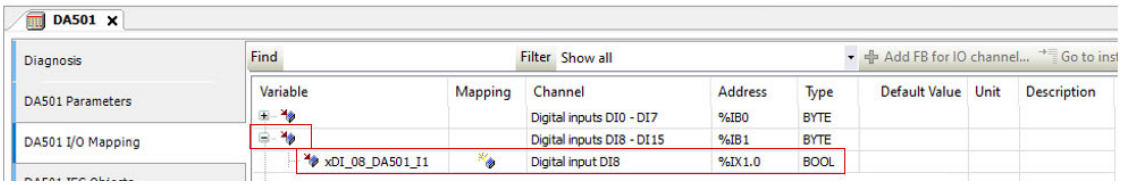




1. Double-click *“DA501”* in the device tree.
   ⇨ A tab opens in the editor view.
2. Select *“DA501 I/O Mapping”*
   ⇨ Here, you will map variable names (symbols) for the channels you will need in the program.

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

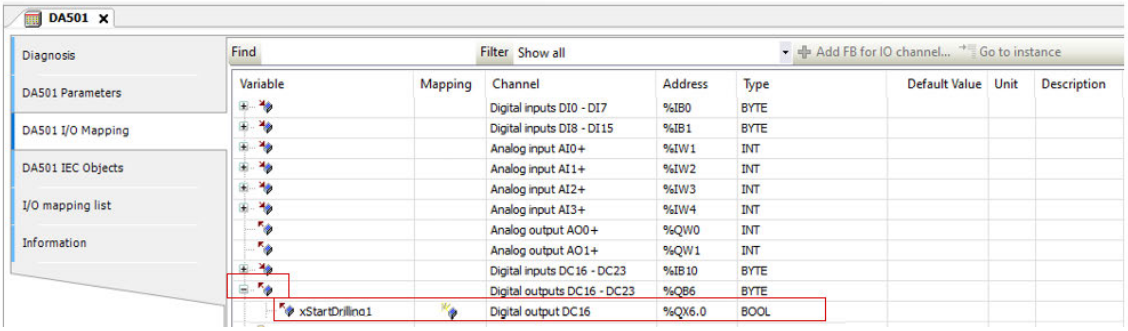
Handle the digital input variables



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

Channel	Type	Variable
Digital input DI8	BOOL	xDI_08_DA501_I1

Handle the digital output variables



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

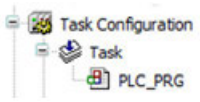
Channel	Type	Variable
Digital output DC16	BOOL	xStartDrilling1

Programming and compiling

Task configuration

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

For this project you will use only one cycling task.



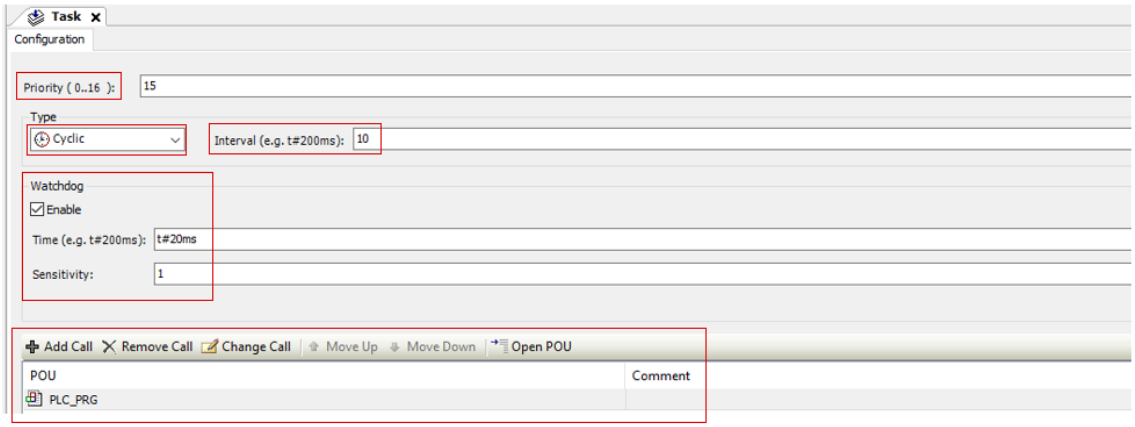
In the device tree, you see the objects *“Task configuration”* and *“Task”*. Both created automatically with the project.

For this project you will use only one cycling task.

▷ Double-click *“Task”* in the device tree.

⇨ A tab opens in the editor view.

For this project you will use only one cyclic task. Keep the default settings for the task.

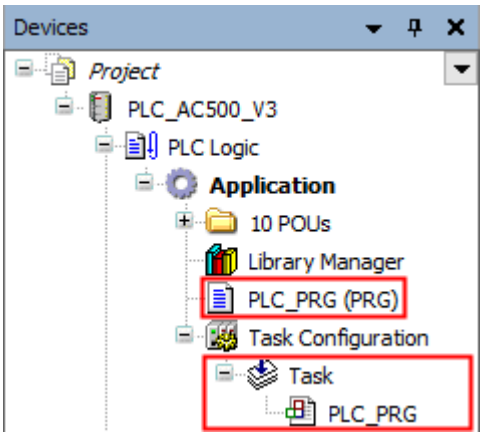


Priority	This is how the CPU prioritizes the task, when more than one task is defined. Priority 0...15 = real time tasks, priority 16 = non-real time task.
Type	In the CPU you can run tasks dependent on the demands of the process
Interval	For cyclic tasks you can set the cyclical execution time. It is usually set in milliseconds with IEC time syntax
Watchdog	To keep track of the time it takes to complete the task
Calls	You can call in one or more program POUs in one single task

Main program PLC_PRG

In the default task configuration ↳ *(shown in chapter 1.2.18.1.2.4.1 Task configuration on page 72)*, there is one call of a POU (program organization unit) i.e. "PLC_PRG".

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



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

To optimize the project readability, you will work with the previously created folder "10 POUs" and add the created subroutines (POUs) to this folder. The subroutines will be created in FBD (Function Block Diagram) editor.

Boolean logic "NOT"

Application example "driller"

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

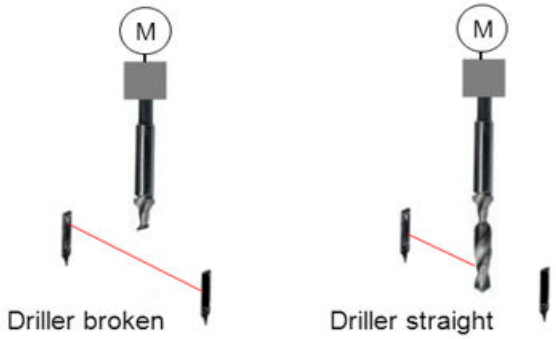


Table 3: Required behavior

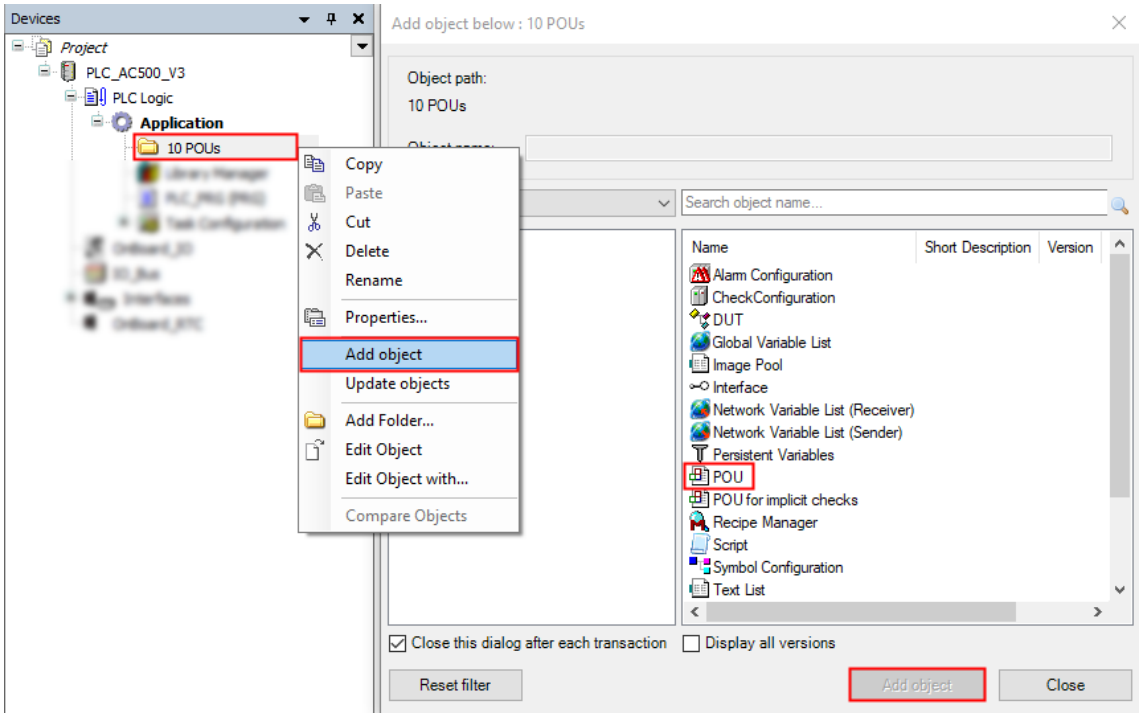
Signal from photo sensor	Required signal of motor ON
FALSE	TRUE
TRUE	FALSE

Table 4: Hardware set-up

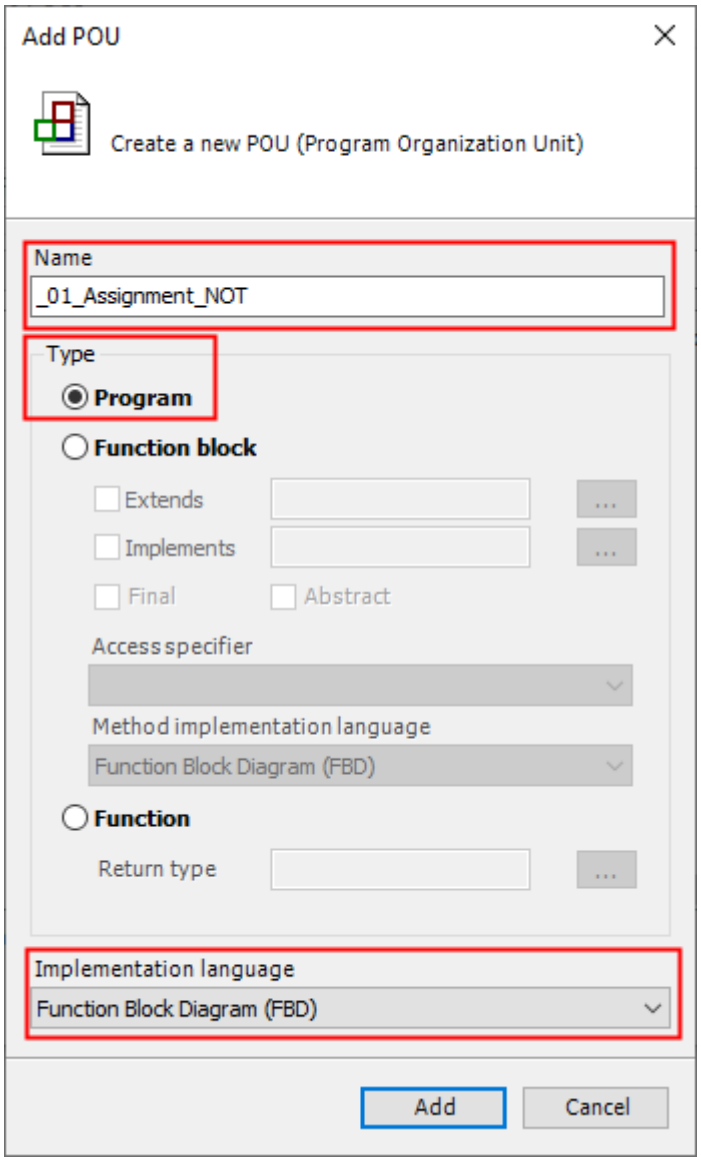
Element	HW channel	Symbol	Description
Switch I1	DA501 DI8	xDI_08_DA501_I1	Photo sensor
LED output DC16	DA501 DC16	xStartDrilling1	Motor on

Implementation

Create a new program POU in the project



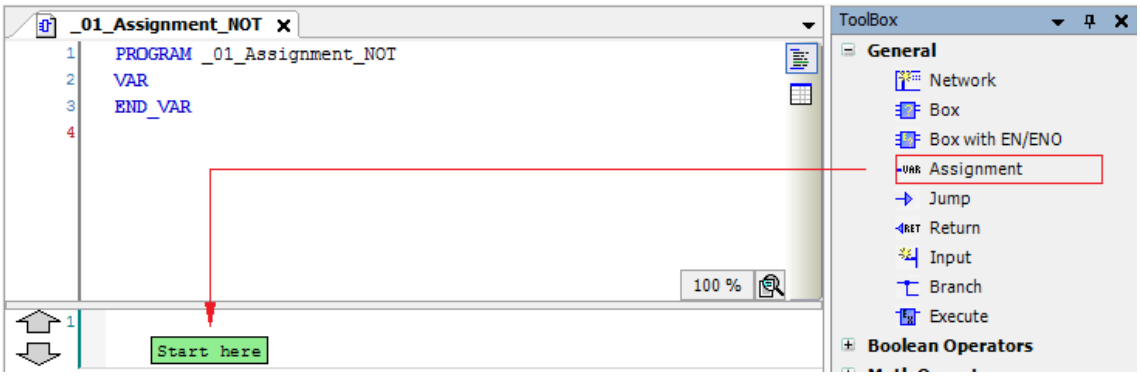
1. Right-click *“10 POUs”*.
2. Select *“Add object”*.
3. Select *“POU”*.
4. Select *“Add object”*.



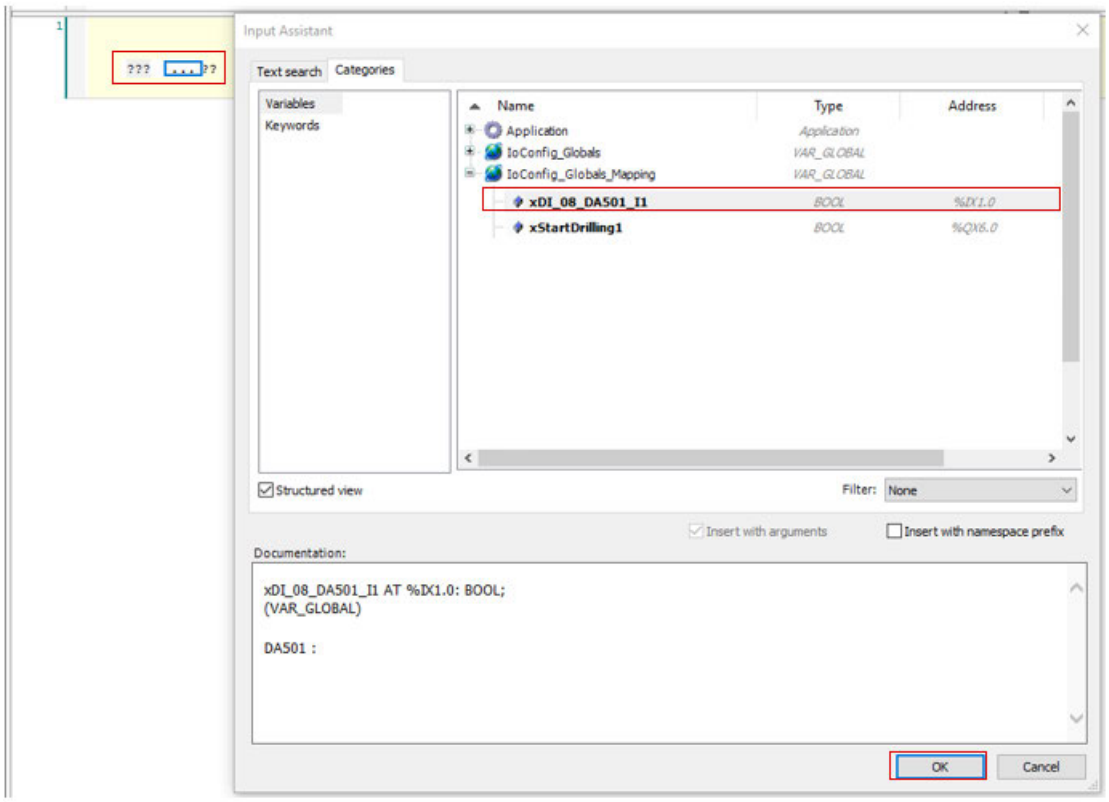
5. Enter *"_01_Assignment_NOT"*.
6. Select *"Program"*.
7. Select *"Function Block Diagram (FBD)"*.
8. Select *"Add"*.
   ⇨ POU has been added.

Assign the hardware DI signals to local variables

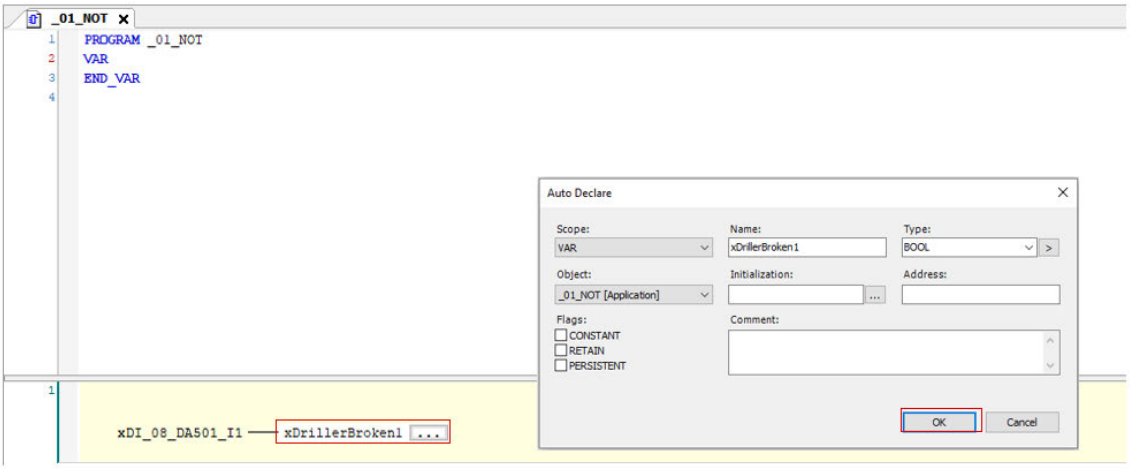
1. Double-click POU*"_01_Assignement_NOT"* in the device tree.



2. Select *"Assignment"* from the ToolBox.
3. Drag and drop *"Assignment"* into the "Start here" field in network *"1"*.

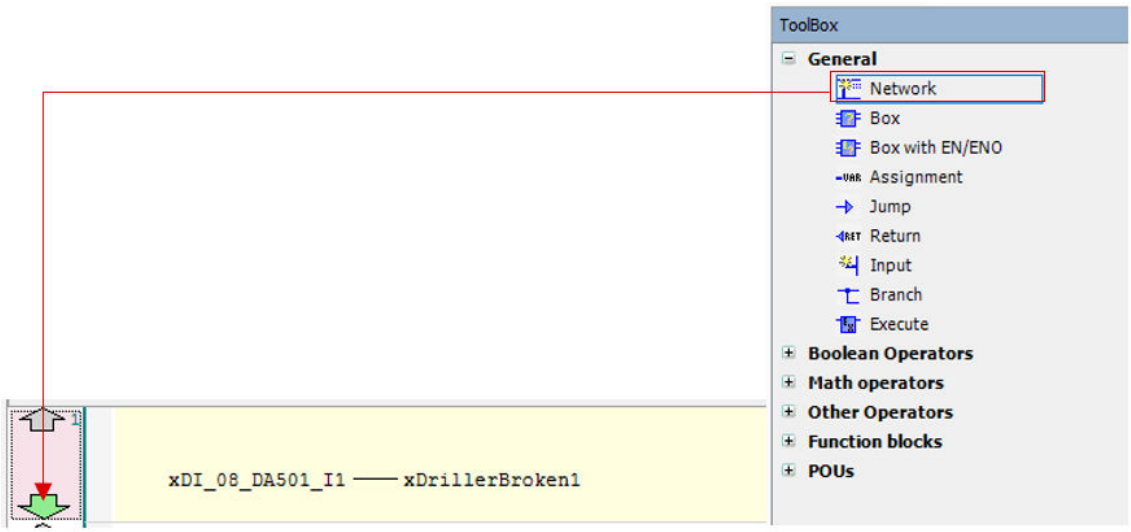


4. Select *"???"* on the left side of the assignment, then select *"..."*.
5. Open the *"Io Config_Globals_Mapping"* mapping list and select *"xDI_08_DA501_I1"*.
6. Select *"OK"* to add this variable to the left side of the assignment connector.



7. Select *"???"* on the right side of the assignment connector and mark the "???".
8. Create a new local variable by typing in "xDrillerBroken1" which will replace the "???".
9. Press *[Enter]*.
   ⇨ *"Auto Declare"* opens.

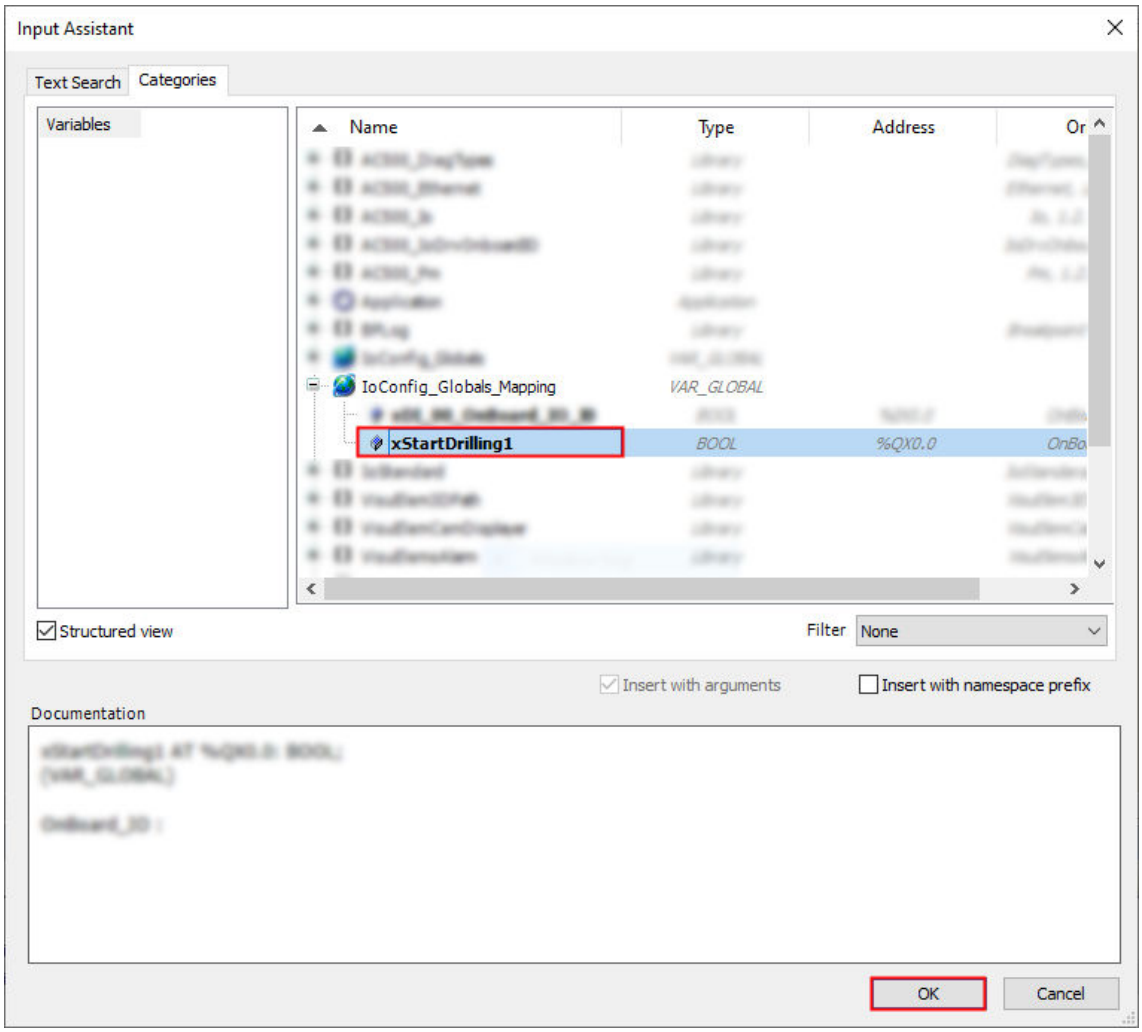
   You see the written variable name and the data type BOOL. The scope is "VAR". It means it is a local variable within this POU.
10. Select *"OK"* to accept the entries.



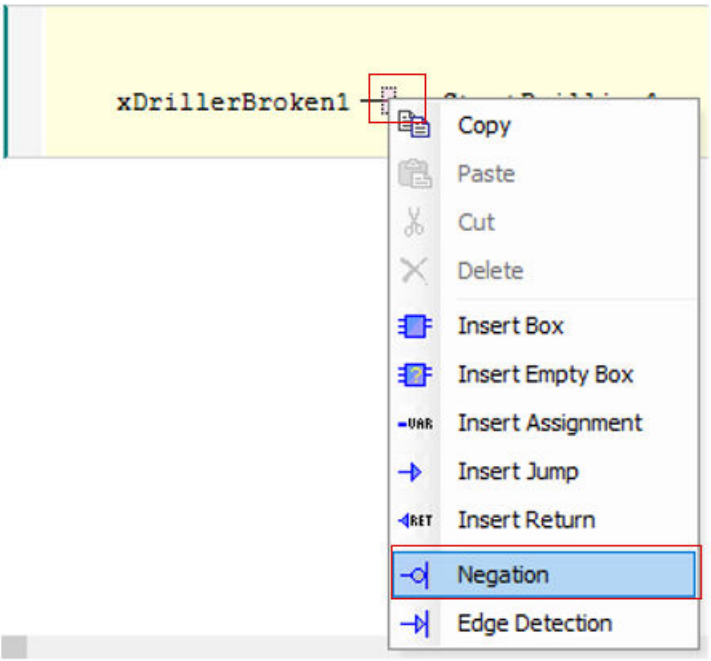
11. Drag and drop *"Network"* from the ToolBox to the down-arrow of network 1.
    ⇨ You added a network *"2"* below network 1.

Add assignments and a Boolean NOT to the DO signals

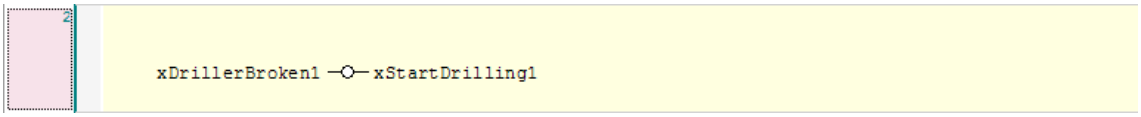
1. Add an assingment from the ToolBox.
2. Type in or copy & paste "xDrillerBroken1" to the left side of the instruction line.
3. Select *"???"* on the right side of the instruction line, then select *"..."*.
   ⇨ *"Input Assistant"* opens.



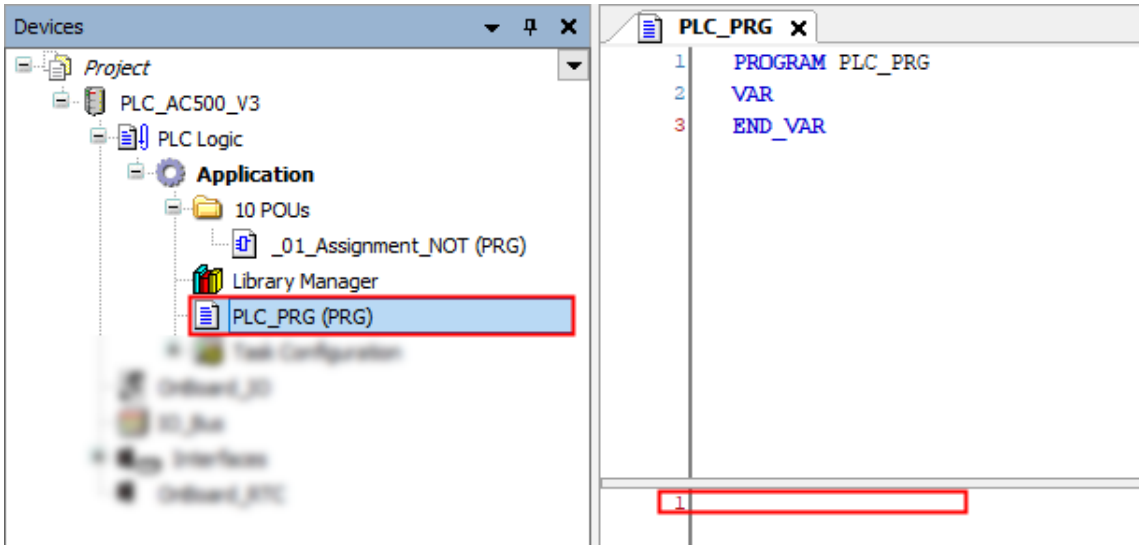
4. In the *"IoConfig_Globals_ Mapping"* variable list, select *"xStartDrilling1"*.
5. Select *"OK"* to close the dialog.



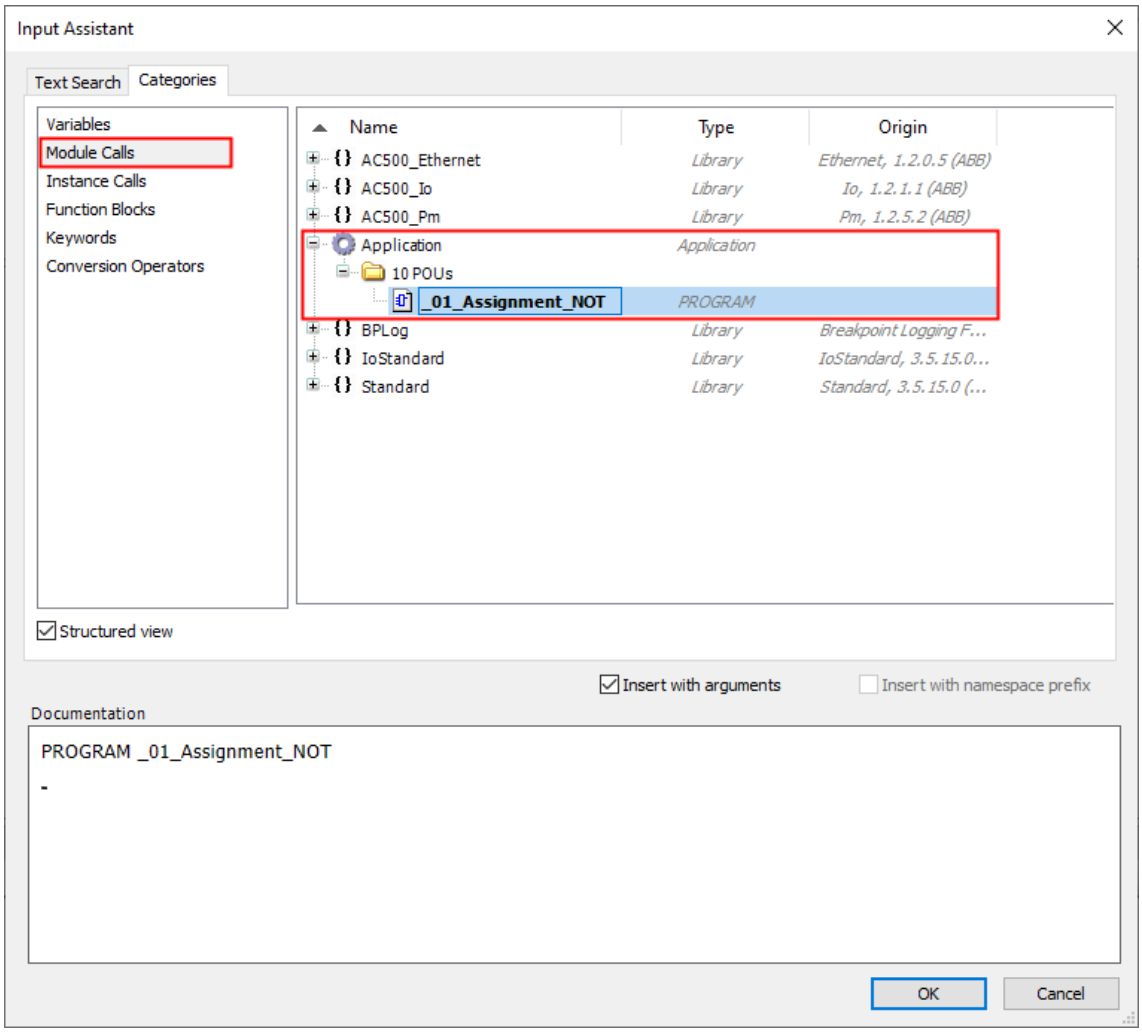
6. Right-click the center of assignment PIN.
7. Select *"Negation"* to add a negation to the assignment.



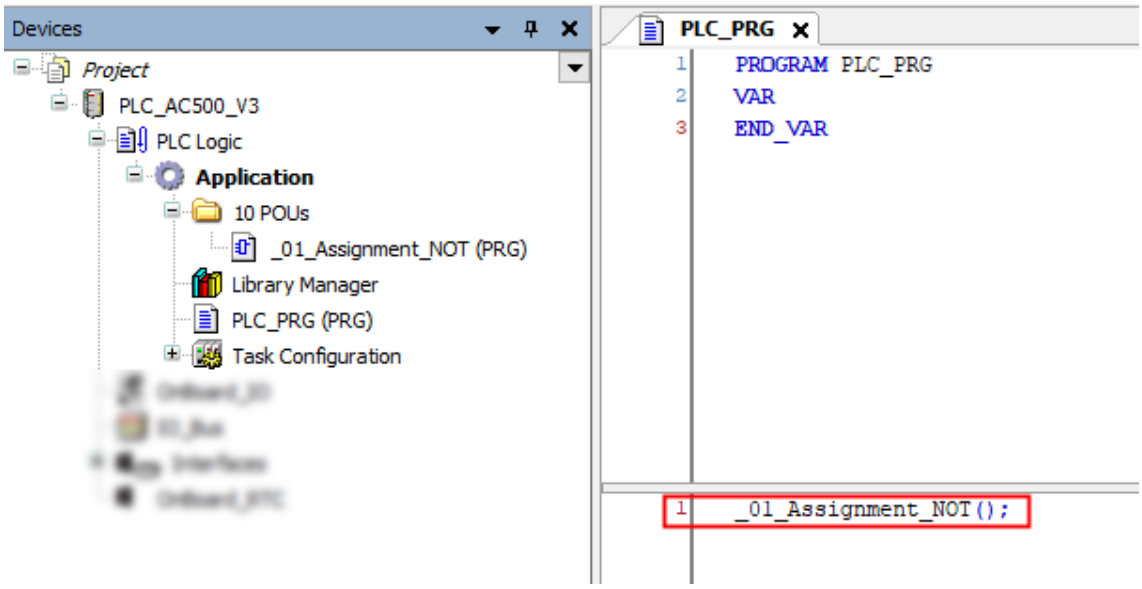
Call the POU in the PLC_PRG



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

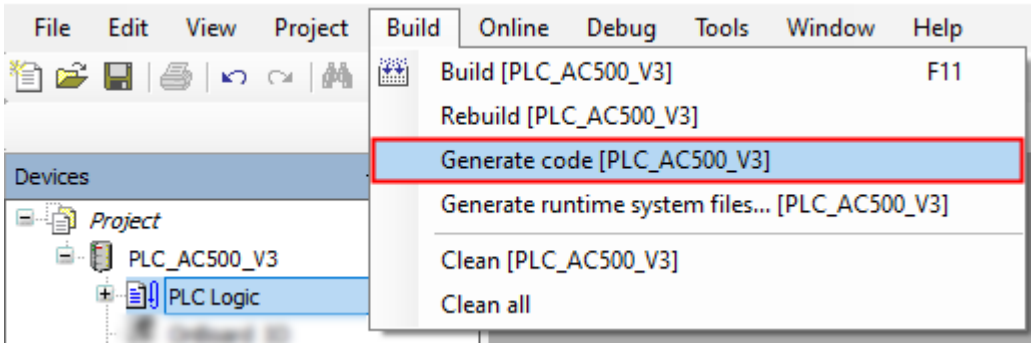


3. Select *“Module Calls”*.
4. Open *“Application”*.
5. Open *“10 POUs”* and select *“_01_Assignment_NOT”*.
6. Select *“OK”* to close the dialog.



Compile the project

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

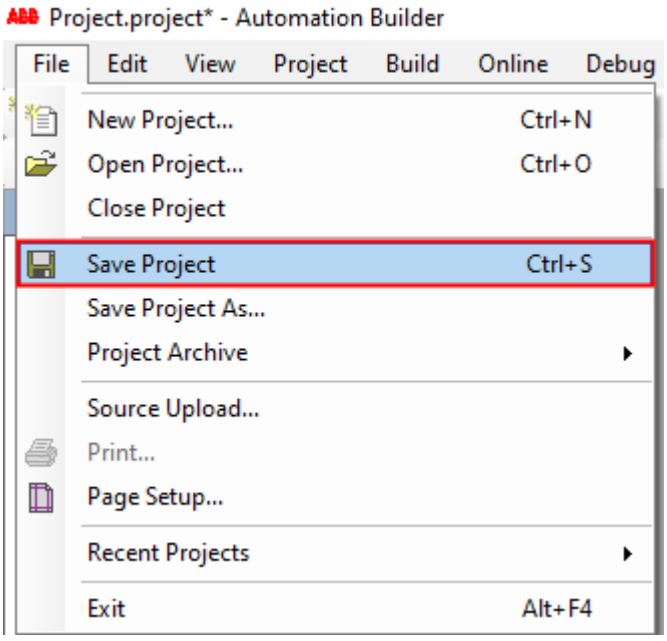


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



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

Alternatively, select the save icon in the tool bar.

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

Set-up the communication gateway

Set-up communication parameters

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

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

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

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

Subnet mask should be 255.255.255.0.

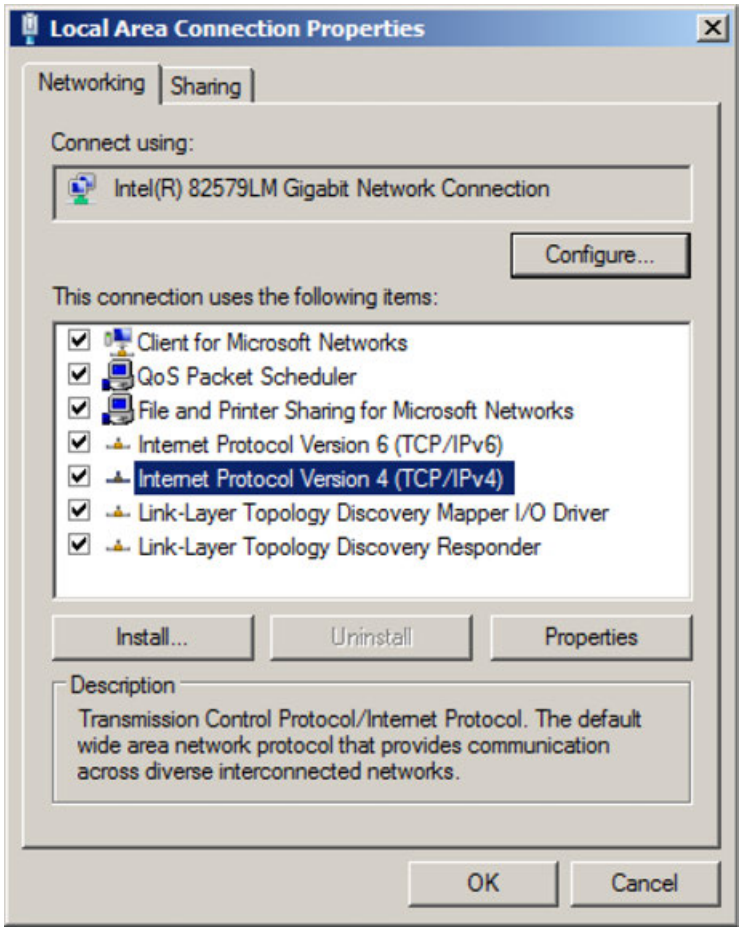
Change the IP address

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

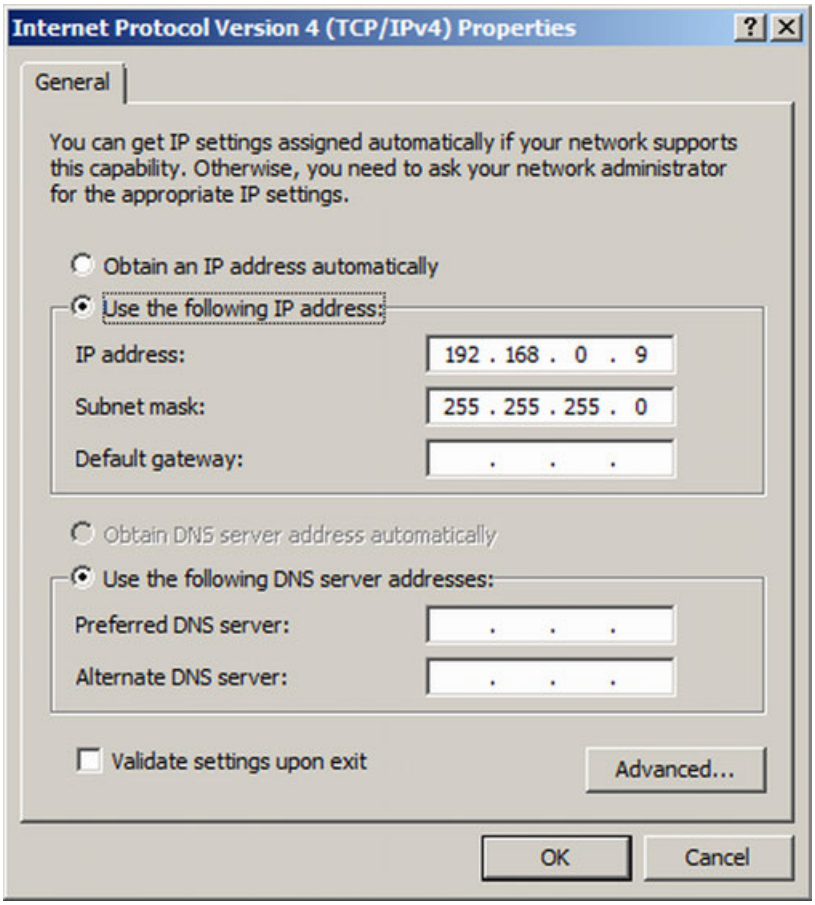
   ⇨

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

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



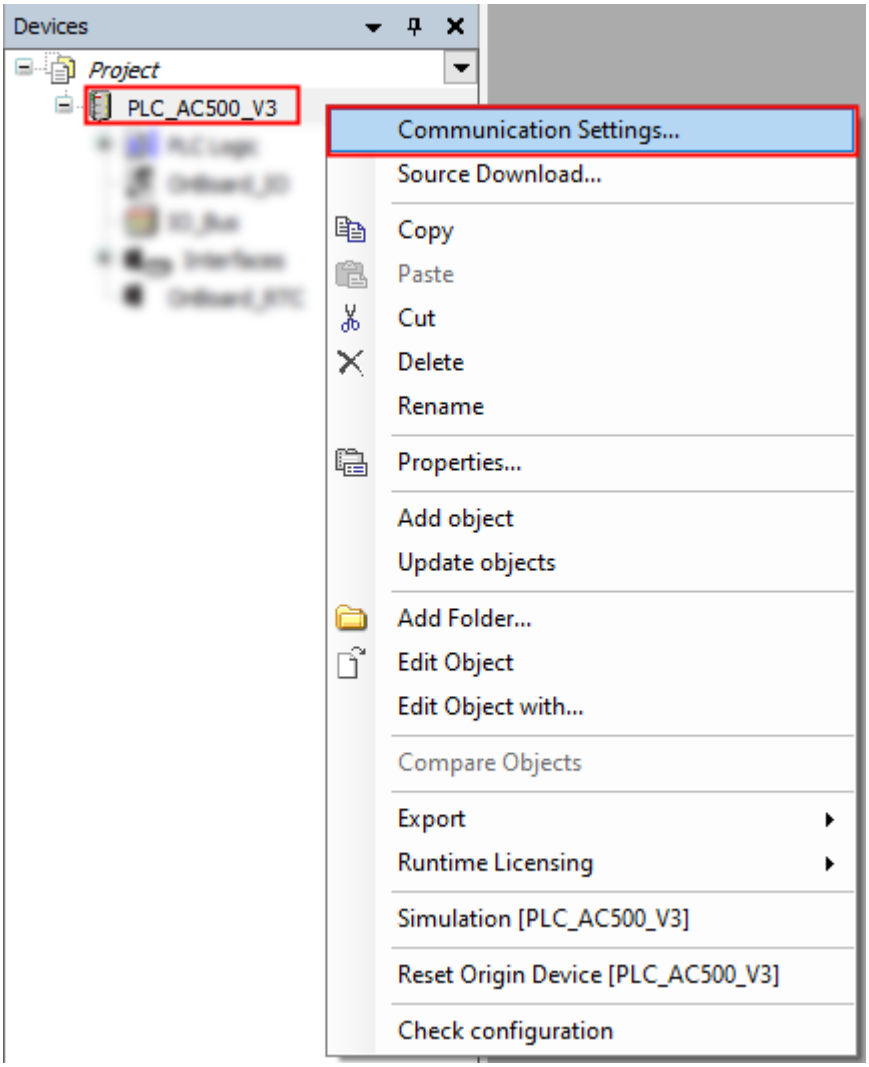
4. Double-click **Internet Protocol Version 4 (TCP/IPv4)**.



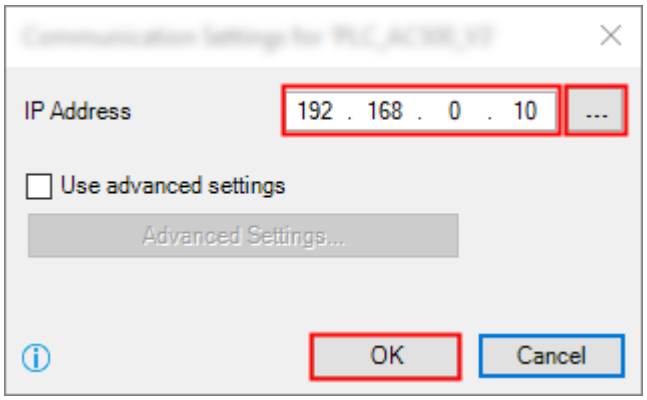
5. Enter your desired IP address and subnet mask.

Set-up the communication gateway

☑ CPU and PC are connected with an Ethernet cable.



1. In the Automation Builder device tree right-click *“PLC_AC500_V3”*.
2. Select *“Communication Settings”*.



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

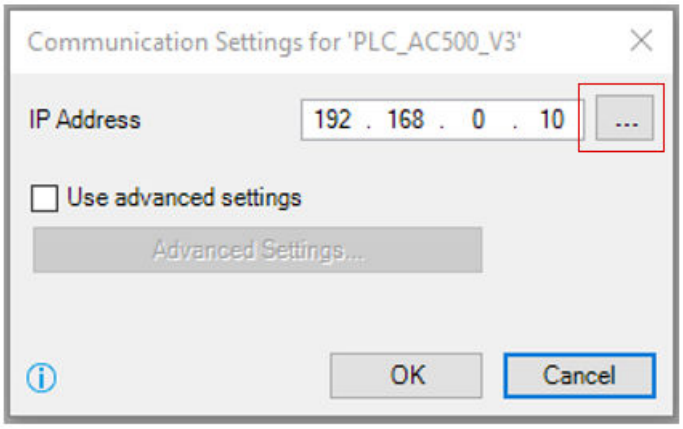
> *The standard (default) IP address of the port ETH1 is: 192.168.0.10*

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

Network scan

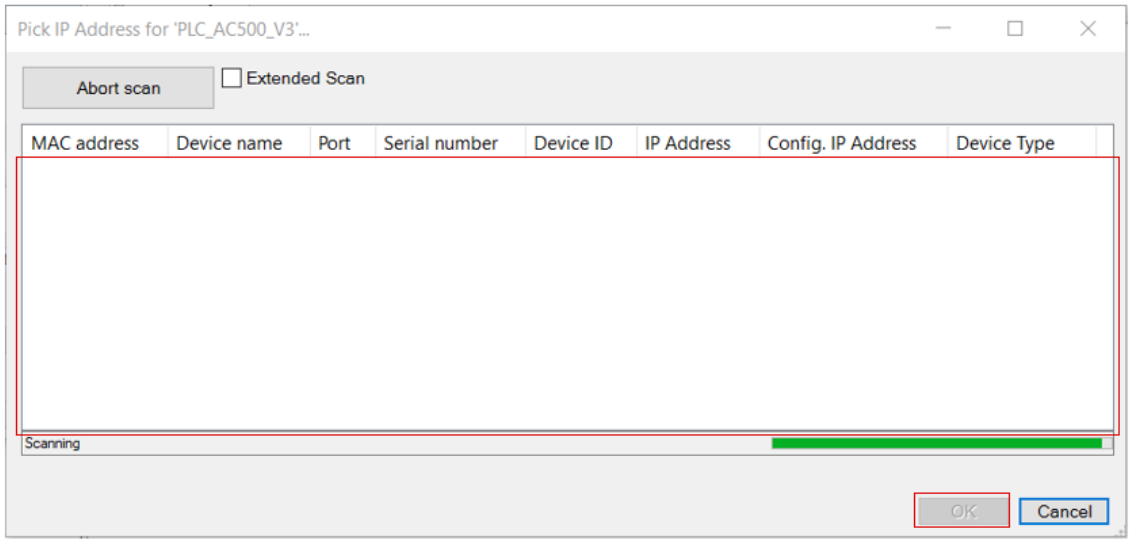
If you need to scan the network for the CPU or if you have multiple CPUs on the same network.

1. Right-click *"PLC_AC500_V3"* in the device tree.
2. Select *"Communication Settings"*.



3. Select *"..."*.

   ⇨ *"Pick IP Address for "PLC_AC500_V3""* opens.



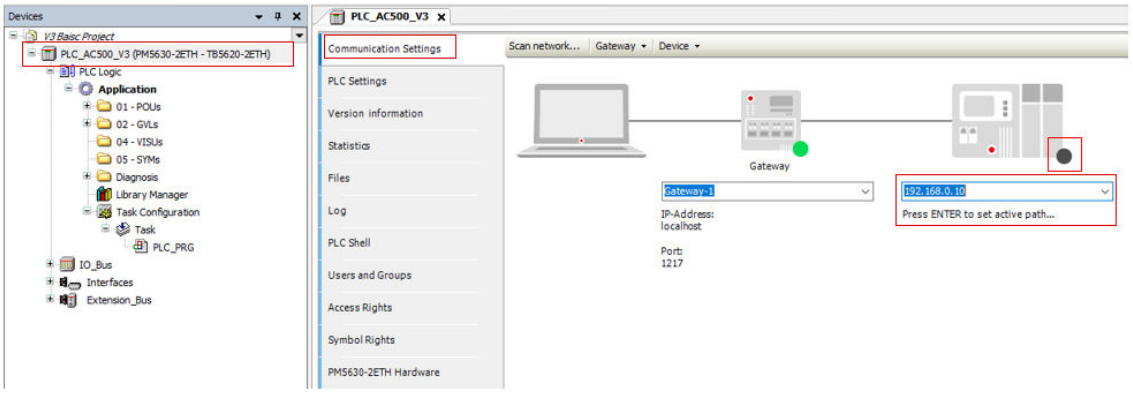
   The automatic scan runs.

   The results will appear in this field.

4. Select the CPU in the field and select *"OK"* to implement the needed communications gateway.

Check communication settings

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



1. Double-click *"PLC_AC500_V3"* in the device tree.

2. Select *“Communication Settings”*.
   ⇨ The selected IP address is shown.
3. If the IP address is not visible, enter the IP address manually.
4. To test the connection and/or to see the CPU information press *[Enter]* or click on the black dot next to the PLC figure.

AC500 V3 firmware installation and update

The PLC firmware can be updated via Automation Builder.

> *This is also necessary for commissioning V3 CPUs.*

A very new CPU has no pre-installed firmware. To guarantee the authenticity of delivered AC500 firmware, V3 CPUs are delivered with a boot loader only. You need to download a valid firmware to the CPU. After download, the functionality of the CPU is given.

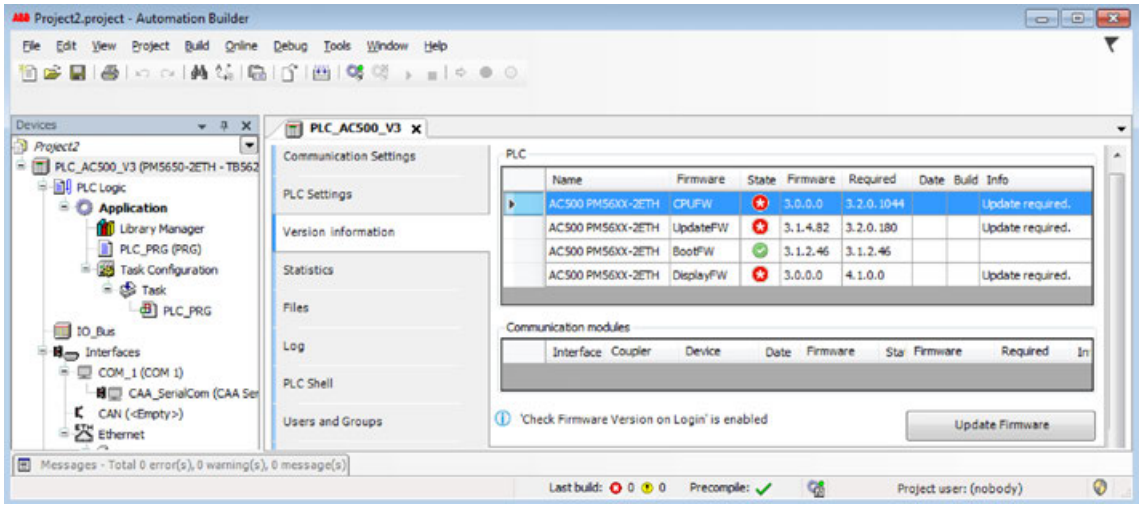
☑ An Automation Builder project with an AC500 V3 CPU is open.

☑ CPU is in "stop" mode or shows **uPdAtE** (update) on the display.

☑ After update the CPU shows either **donE** or **StoP** on the display

☑ For new modules: IP address is set. (The default IP address is 192.168.0.10)

1. Double-click CPU *“PLC_AC500_V3”*.
2. Select *“Version information”*.



3. Select *“Update Firmware”*.
   ⇨ While the update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

4. Wait for the PLC to finish the update.

   A completed update is indicated by a message on the display. Either **donE**, or **StoP**.

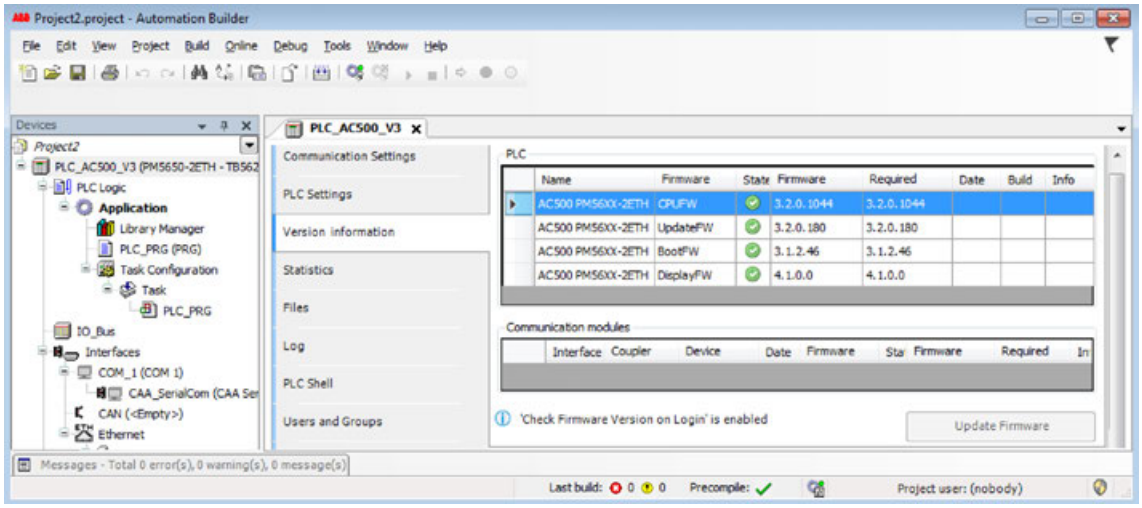
> **NOTICE!**
> Do not disconnect the power supply during the update process! The PLC could be damaged.

   ⇨ **StoP** indicates a restart has been performed by the CPU. When **donE** is displayed sometimes it is necessary to re-boot the CPU manually, e.g., by powering-off. Manual re-boot might be, e.g., for some older CPU versions or if downgrading to an older firmware version according to application settings.

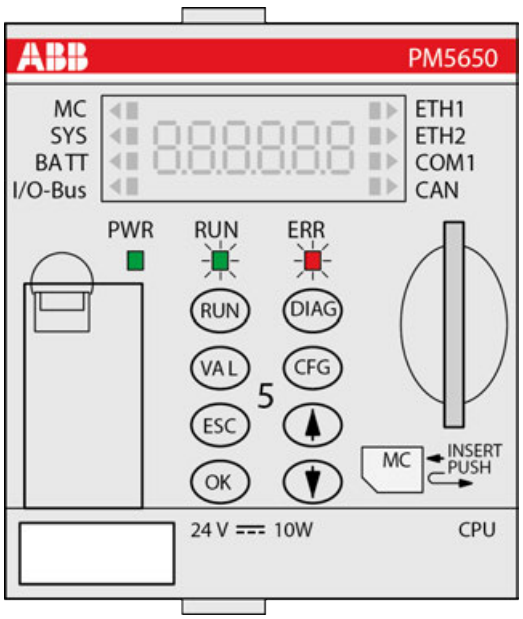
   The CPU display shows "stop" after re-boot. The update process is finished.

5. If necessary, refresh the version information by switching to another tab and back.

   ⇨ Successful firmware update:



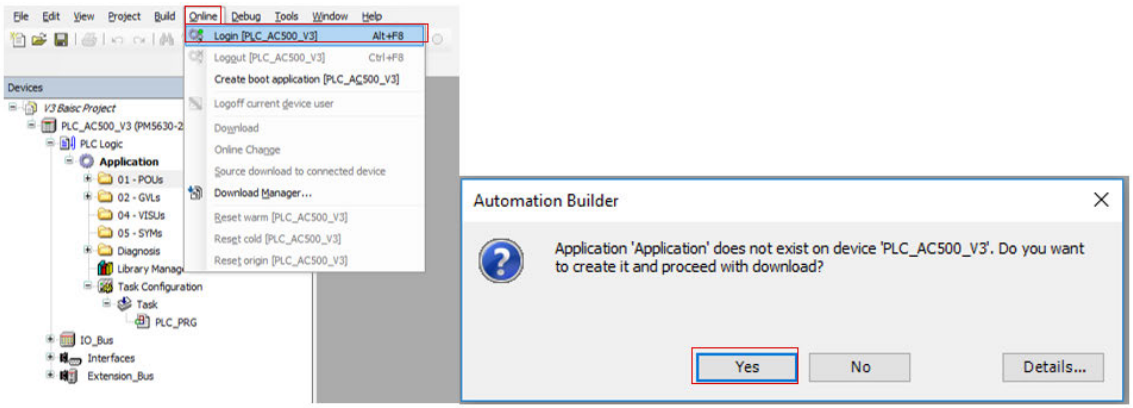
Behavior of LEDs during firmware update



LED	LED flashes	Status
RUN and ERR	Toggling	Update pending
RUN	Flashing slow	Done successful
ERR	Flashing slow	Done failed

Log-in to CPU and download the program

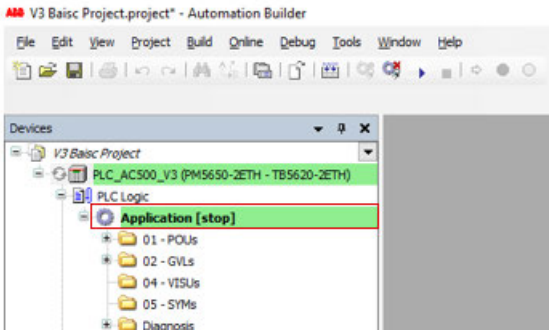
Logging-in to the CPU will load the project into the AC500 V3 CPU. The first log-in will also load the hardware set-up.



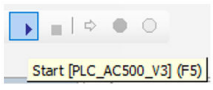
1. In the Automation Builder menu select *“Online ➔ Login [PLC_AC500_V3]”*.

   ⇨ A pop-up will appear.

2. Select *“Yes”* to download the application to the AC500V3 CPU.



   ⇨ PLC is in "stop" mode.



3. Start the PLC ↳ *Chapter 1.2.18.1.2.8.1 “Start the program execution” on page 90*.

Generally, if the CPU is in RUN mode, i.e. in program execution mode, a download will always cause the mode change to "stop". In stop mode the CPU is not controlling the system!

Always, after selecting the "Login" command, read carefully the dialog box text to ensure that you are aware of the CPU’s behavior after the command confirmation.

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

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

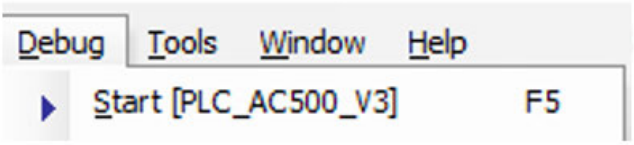
Test the program

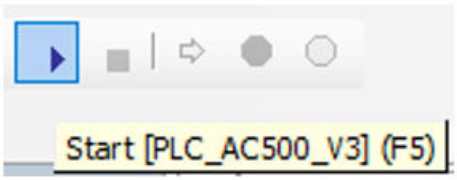
Start the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in "stop" mode.





▷ Select menu *"Debug ➔ Start [PLC_AC500_V3]"*.

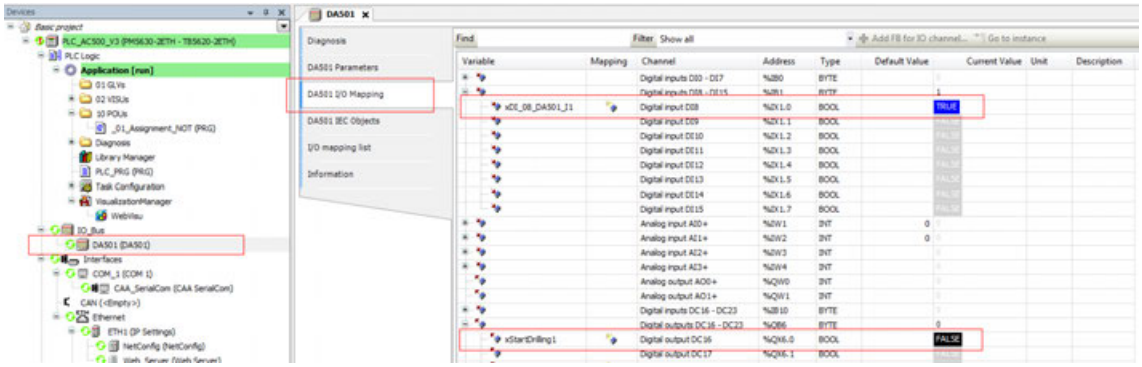
Alternatively, select the "start" icon in the tool bar.

Alternatively, press *[F5]*.

Test the function

▷ Operate the switch I1 and observe:

- The LEDs of the relevant DA501 inputs and outputs.
- The online status of inputs and outputs within the POU.

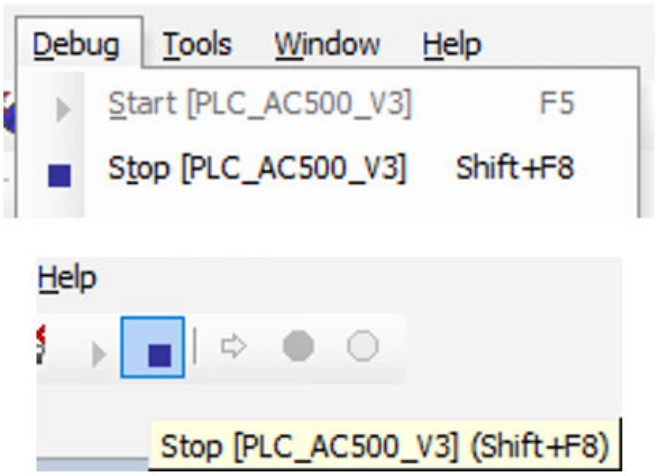


Stop the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

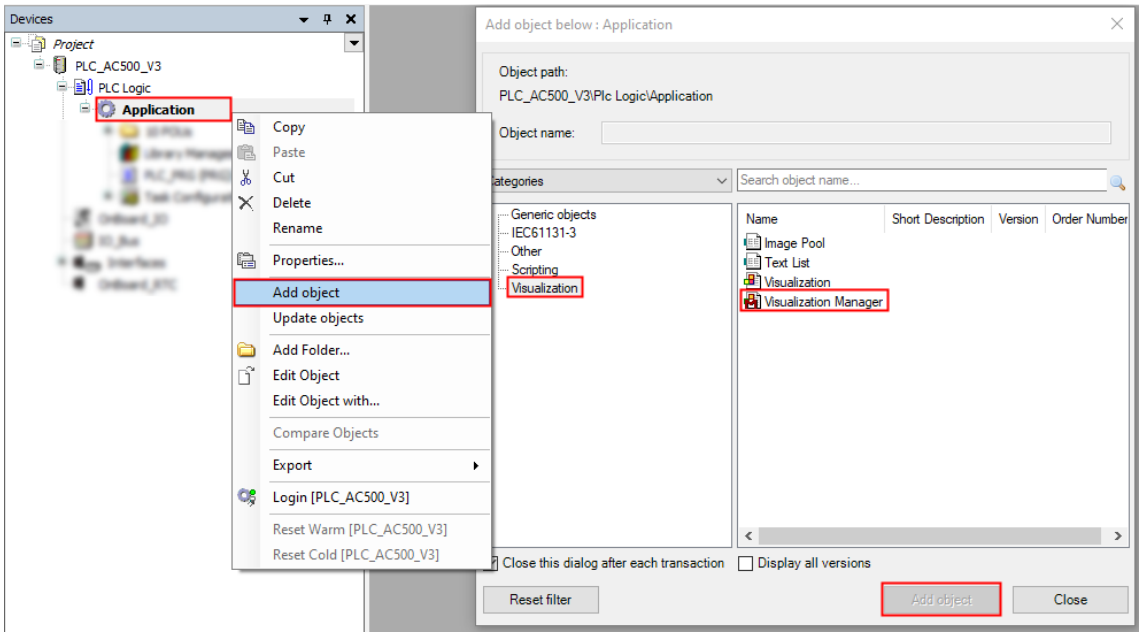
☑ The CPU is in RUN mode.



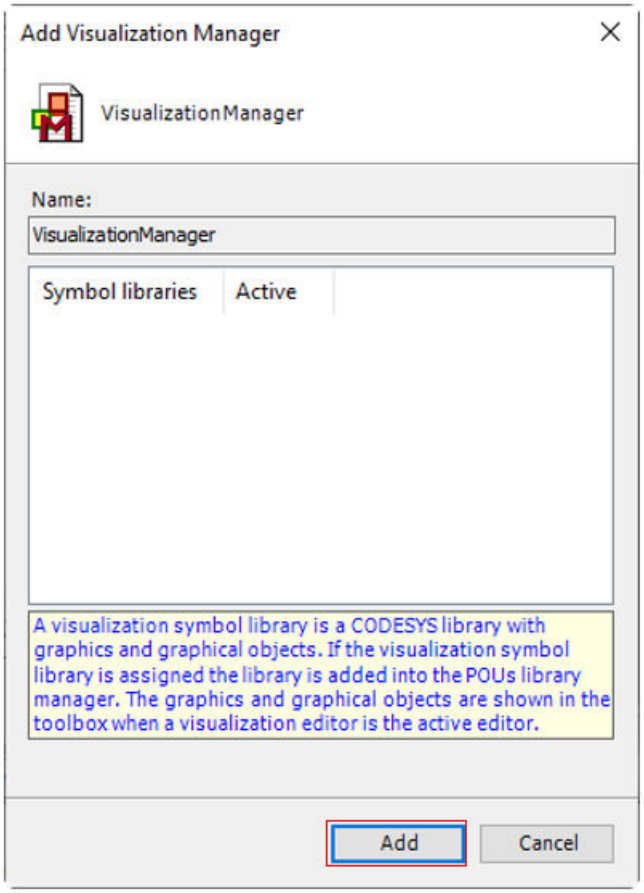
▷ Select menu *“Debug ➔ Stop [PLC_AC500_V3]”*
Alternatively, select the "stop” icon in the tool bar.
Alternatively, press *[Shift] + [F8]*.

Set-up visualization

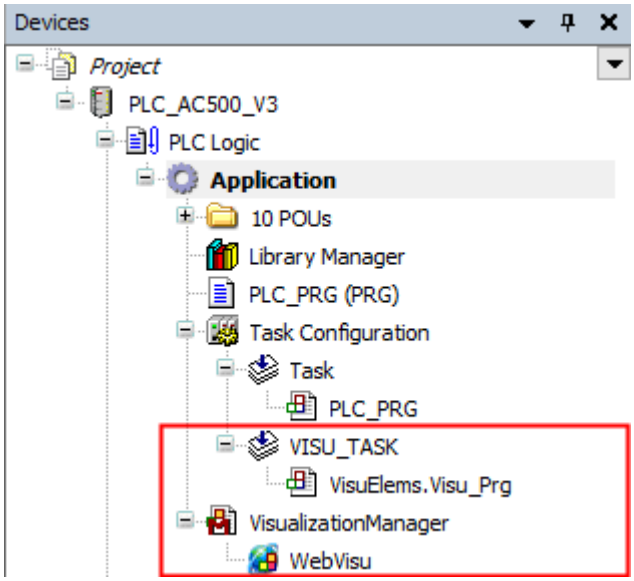
Add the VisualizationManager



1. Right-click *“Application”* in the device tree.
2. Select *“Add object”*.
3. Select *“VisualizationManager”*.
4. Select *“Add object”* to add the VisualizationManager to the project.
   ⇨ Dialog *“Add Visualization Manager”* opens.

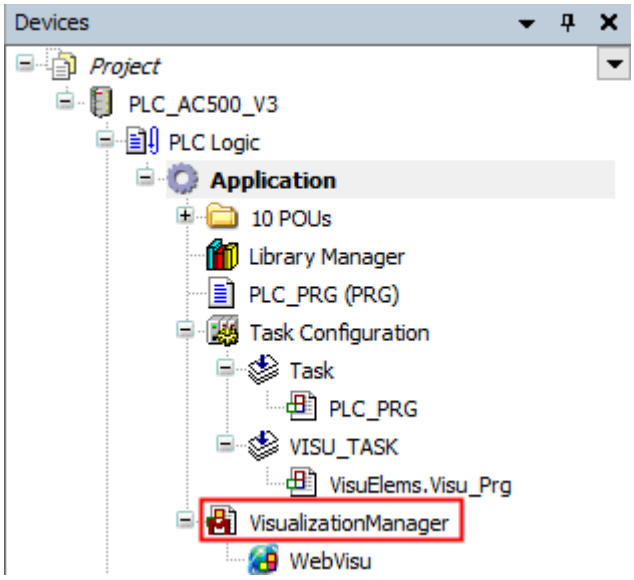


5. Select *“Add”*.

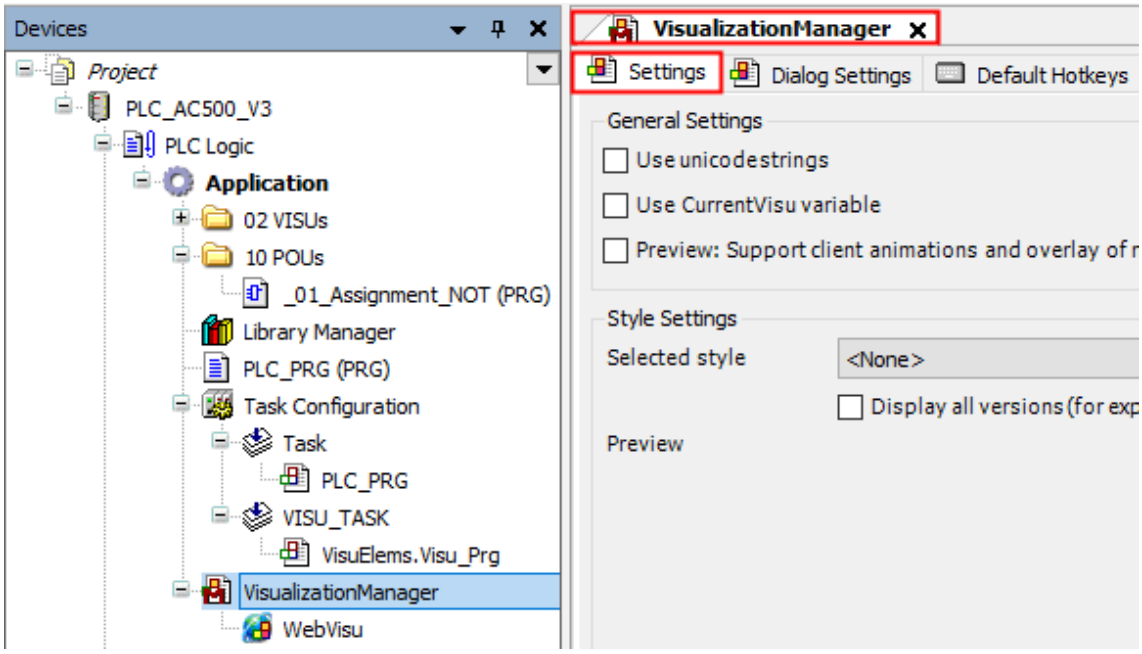


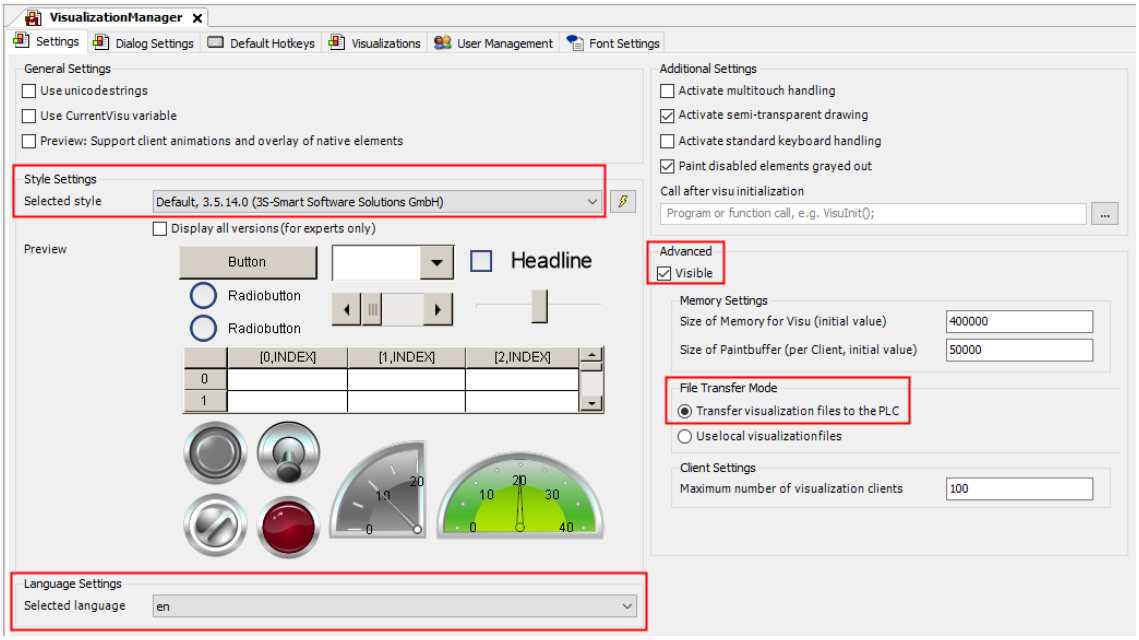
⇨ You added the objects *“VisualizationManager”* and *“VISU-TASK”* to the device tree.

Set-up the VisualizationManager



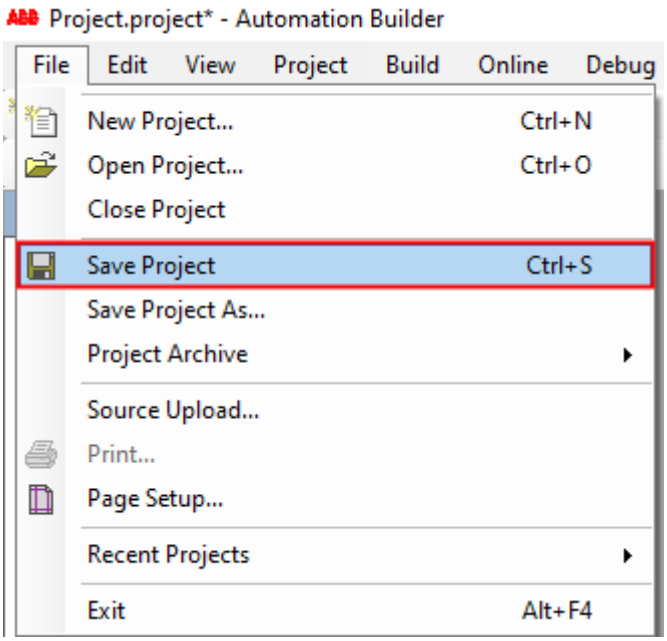
1. Double-click VisualizationManager in the device tree.
   ⇨ A tab opens in the editor view.





2. Select *“Settings”*.
3. Open the drop-down menu *“Selected style”*.
4. Select *“Default, x.x.x”* (exemplary).
5. Open the drop-down menu *“Selected language”*.
6. Select *“en”* for English language in the visualization.
7. Enable *“Visible”* for advanced settings.
8. Keep the file transfer to enable the visualization on the PLC (mandatory for web server function ↳ *Chapter 1.2.18.1.2.11 “Enable web visualization” on page 103*).

Save the project



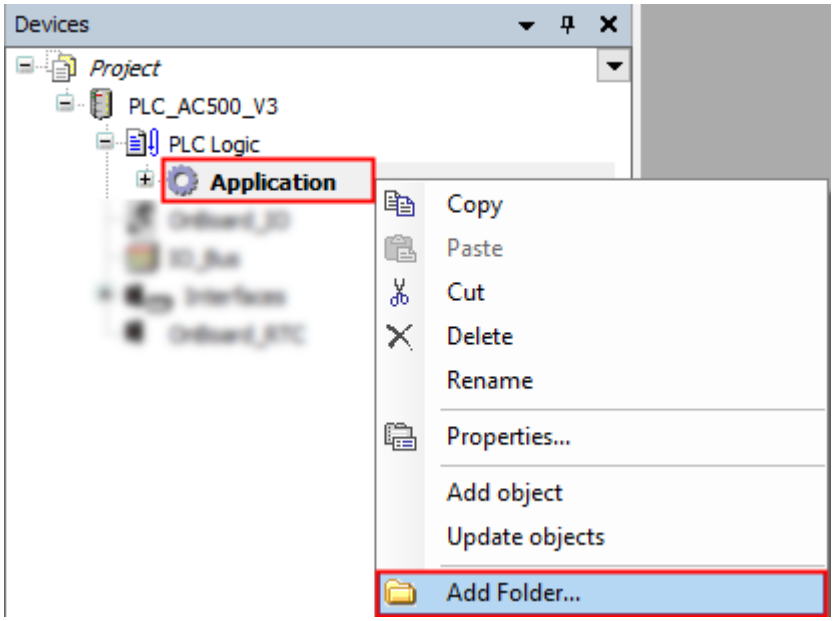
▷ Select menu *“File ➔ Save Project”*.

Alternatively, select the save icon in the tool bar.

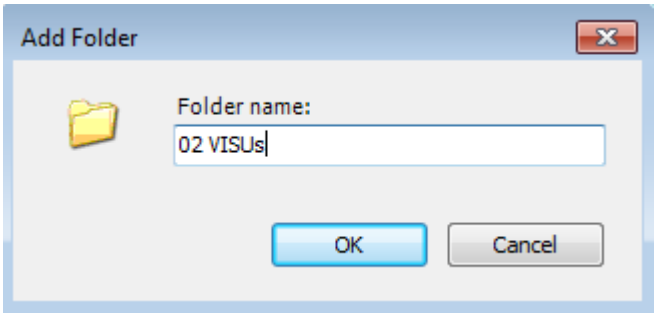
Alternatively, press *[Ctrl] + [S]*.

Create visualization

Add a folder for visualization screens

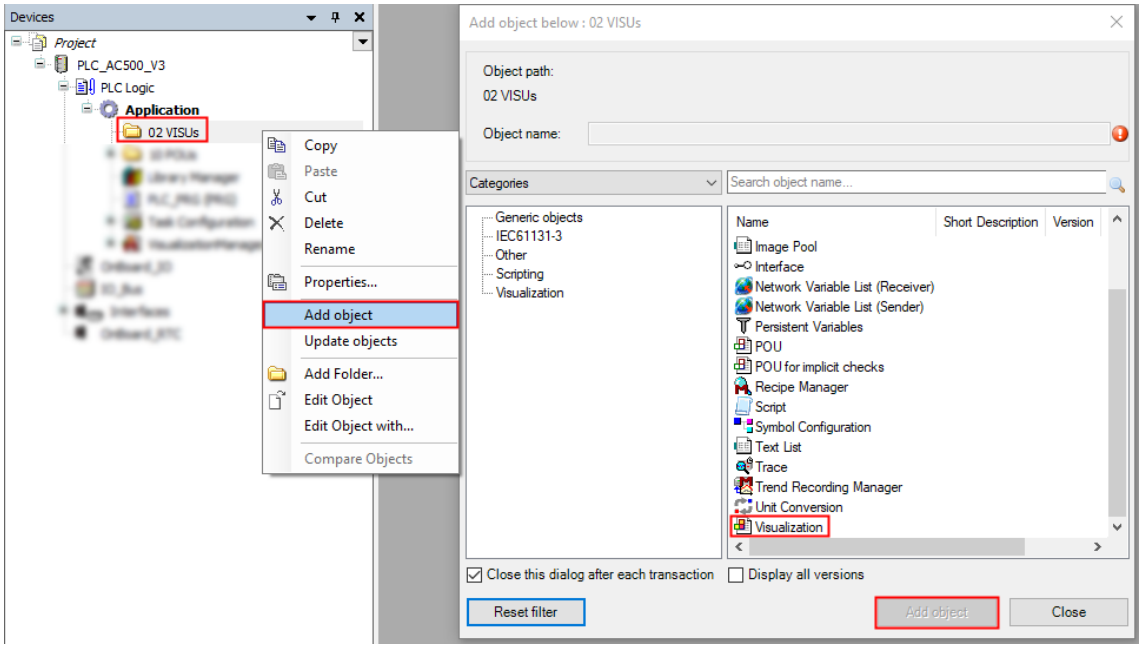


1. Right-click *“Application”* in the device tree.
2. Select *“Add Folder”*.

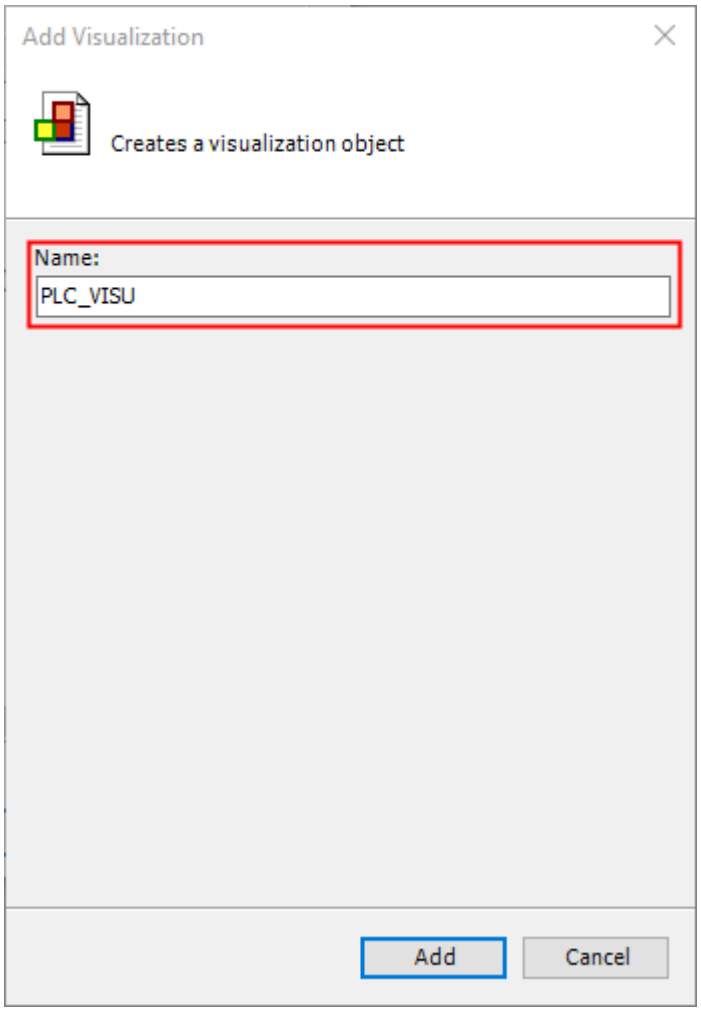


3. Type in "02 VISUs".
4. Select *“OK”* to add the folder.

Add a screen for "_01_Assignment_NOT" POU



1. Right-click *“02 VISUs”*.
2. Select *“Add object”*.
3. Select object *“Visualization”*.
4. Select *[OK]*.



5. Type in "PLC_VISU".
6. Select *“Add”*.
   ⇨ A tab opens in the editor view.

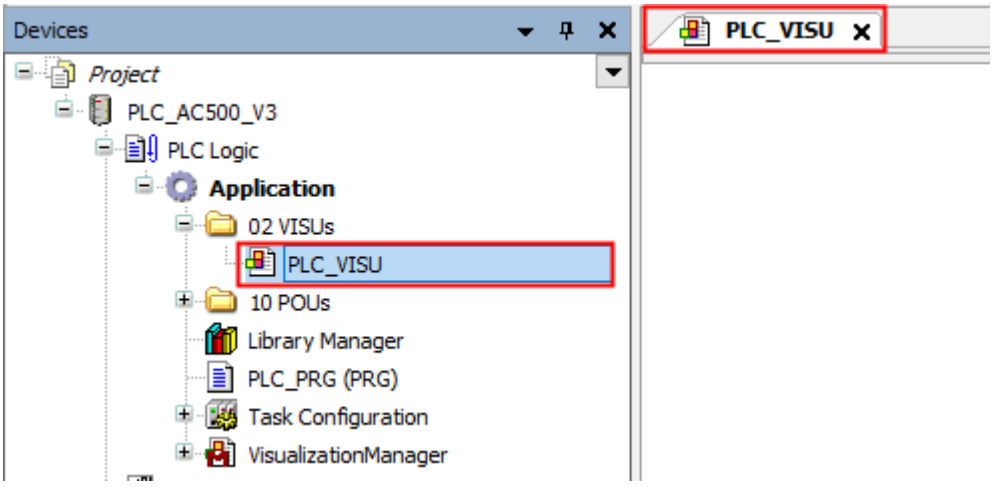


Fig. 7: PLC_VISU_tab

The name "PLC_VISU" has been chosen, because it is the default name for a home screen in a web visualization.

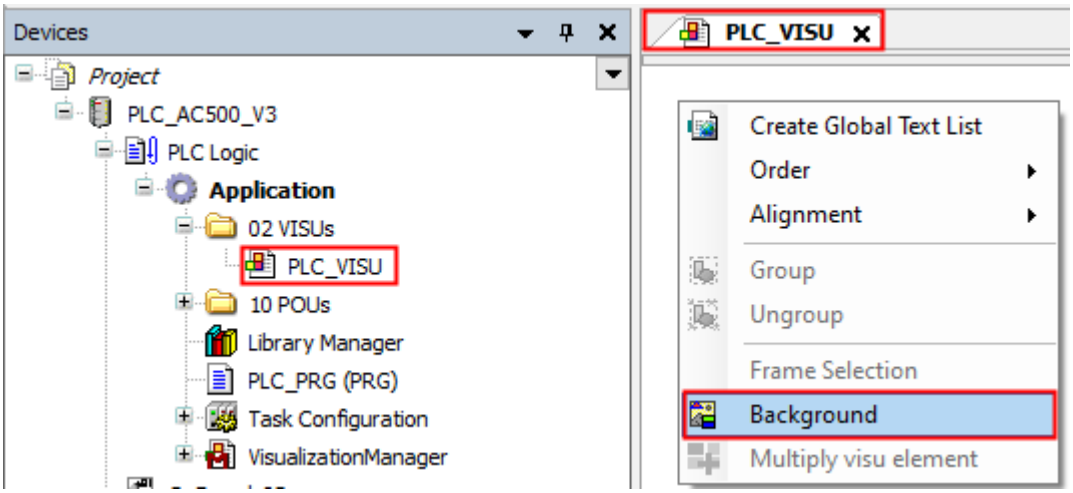
If you have more than one visualization object in your project, it will be useful to choose another name, e.g. "_01_Assignment_NOT_v". And to choose "PLC_VISU" as a home screen to access all available visualization screens.

The name of a visualization object can be modified afterwards.

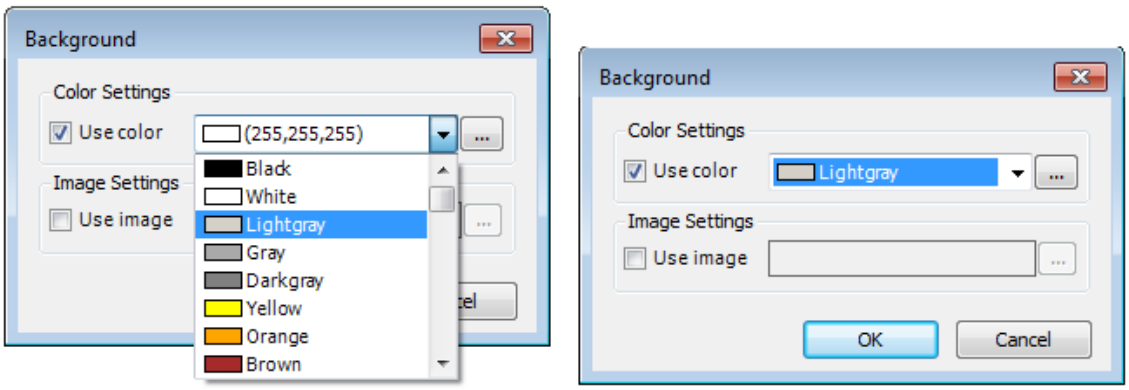
Creating and configuring of visualization

Change background color

1. Double-click *“PLC_VISU”* in the device tree.
   ⇨ A tab opens in the editor view.



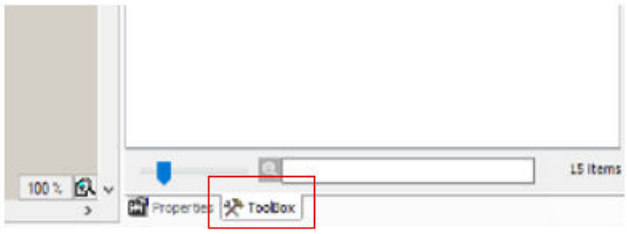
2. Right-click anywhere on the "PLC_VISU" editor page.
3. Select *“Background”*.



4. Enable the check box *“Use Color”*.
   ⇨ This enables the drop-down menu.
5. Select a color, e.g., *“Lightgray”*.
6. Select *[OK]* to add the color to "PLC_VISU".

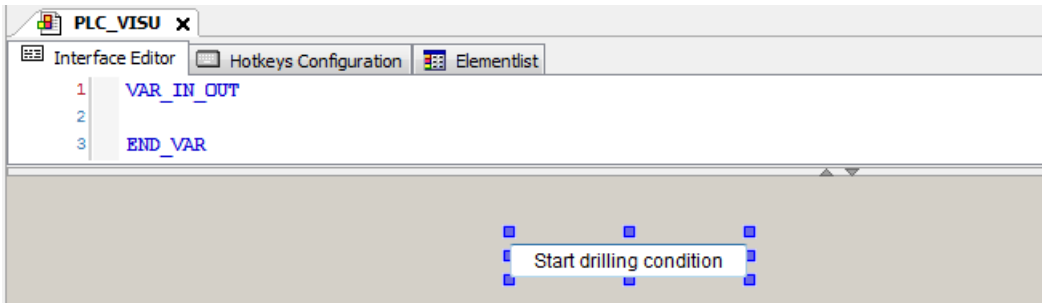
Add a screen title

1. Double-click on *"PLC_VISU"* in the device tree.



2. Select *"ToolBox"*.

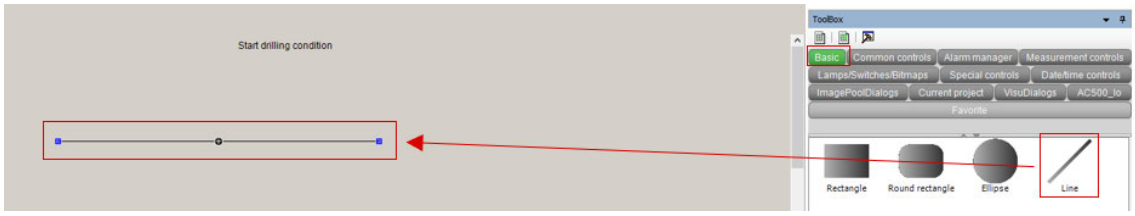
3. Select *"Common controls"*.
4. Drag and drop *"Label"* to the page.



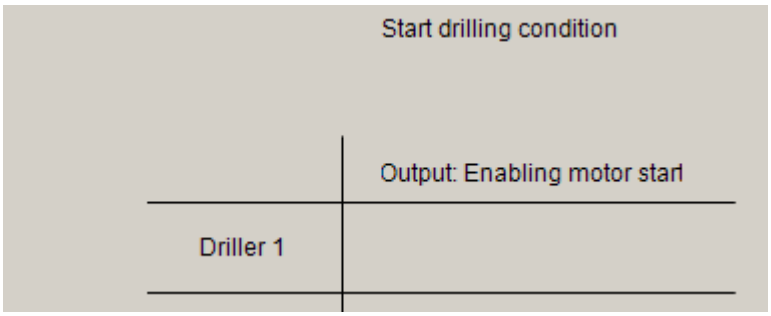
5. Type in "Start drilling condition".

Further lines and labels

1. Double-click on *"PLC_VISU"* in the device tree.



2. Select *"ToolBox"*.
3. Select *"Basic"*.
4. Drag and drop the line. Then drag the line to the needed length.

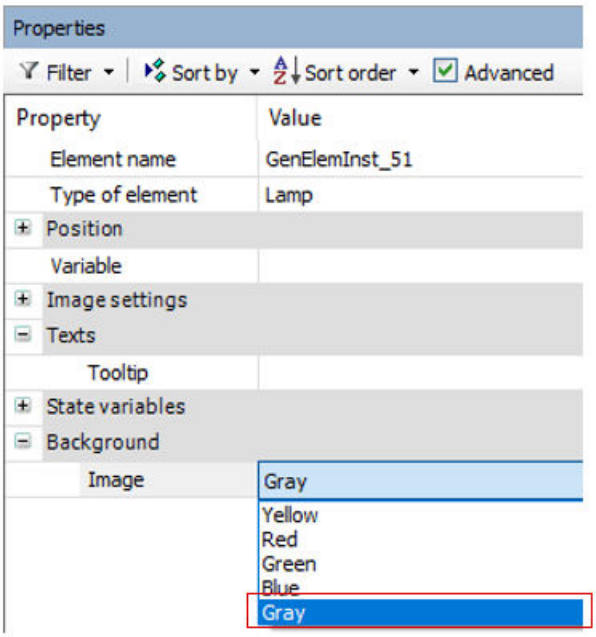


5. Follow the same procedure to create the other shapes and labels.

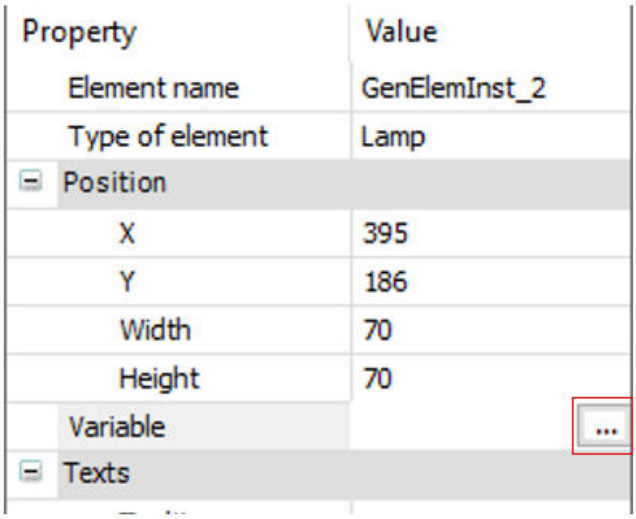
Lamp element for signal indication

1. Double-click on *“PLC_VISU”* in the device tree.

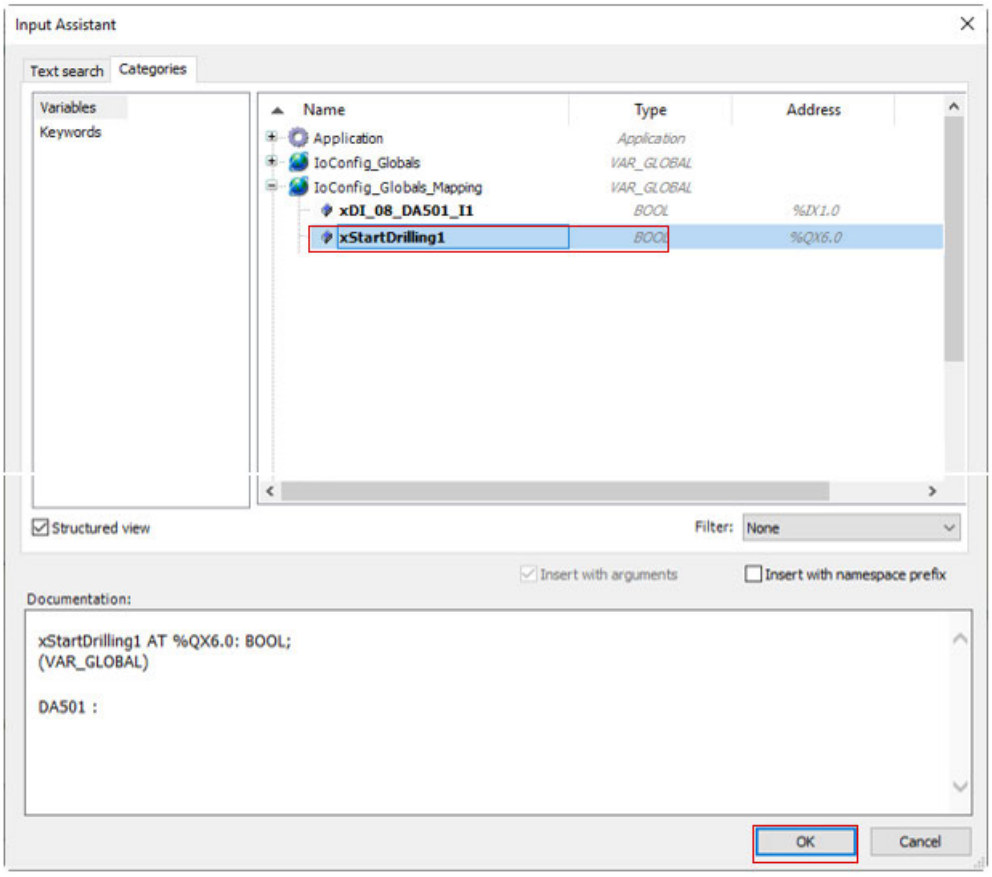
2. Select *“ToolBox”*.
3. Select *“Lamps/Switches/Bitmaps”*.
4. Drag and drop *“Lamp”* to the screen.
5. Adapt the size, if required.



6. Under *“Image”*, select *“Gray”*.



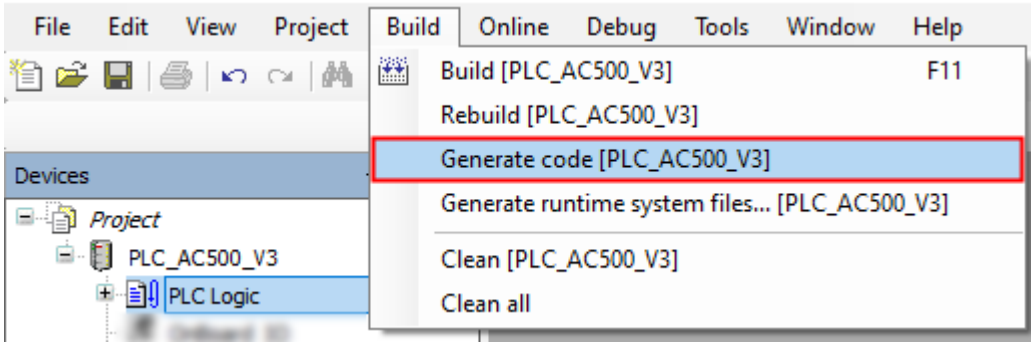
7. Double-click on *"Variable"* and select *"..."* to select a variable from the list.



8. Under *"IoConfig_Globals_Mapping"*, select *"xStartDrilling1"*.
9. Select *[OK]*.

Compile the project

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

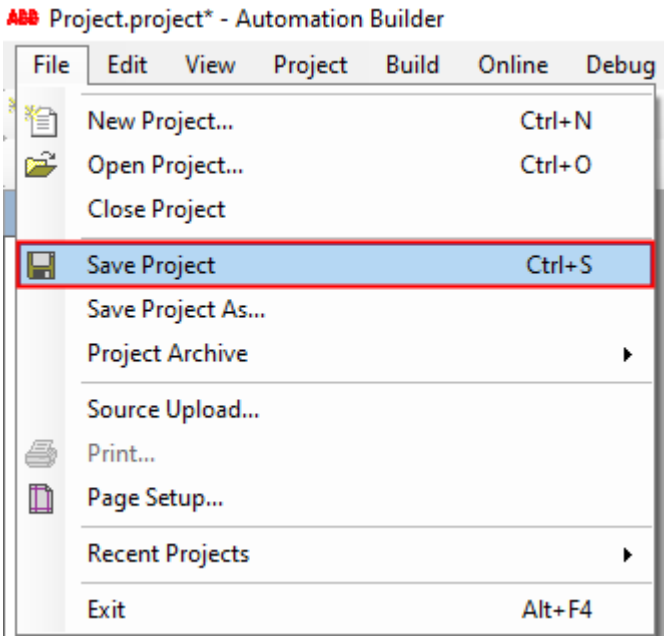


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



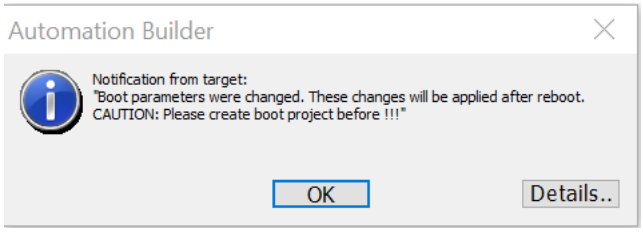
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

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

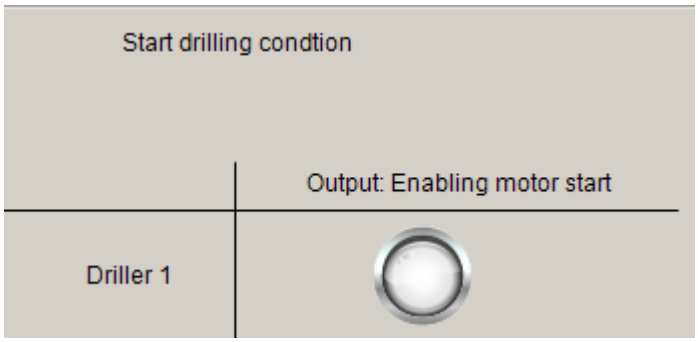
Loading the project to the CPU

1. Download the project to the CPU *as described in Chapter 1.2.18.1.2.7 , on page 89*.
2. Check the notification window at the end of the download. In case of message "Boot parameters were changed. These changes will be applied after reboot", a reboot of the CPU is required after creation of the boot project.



Test the program

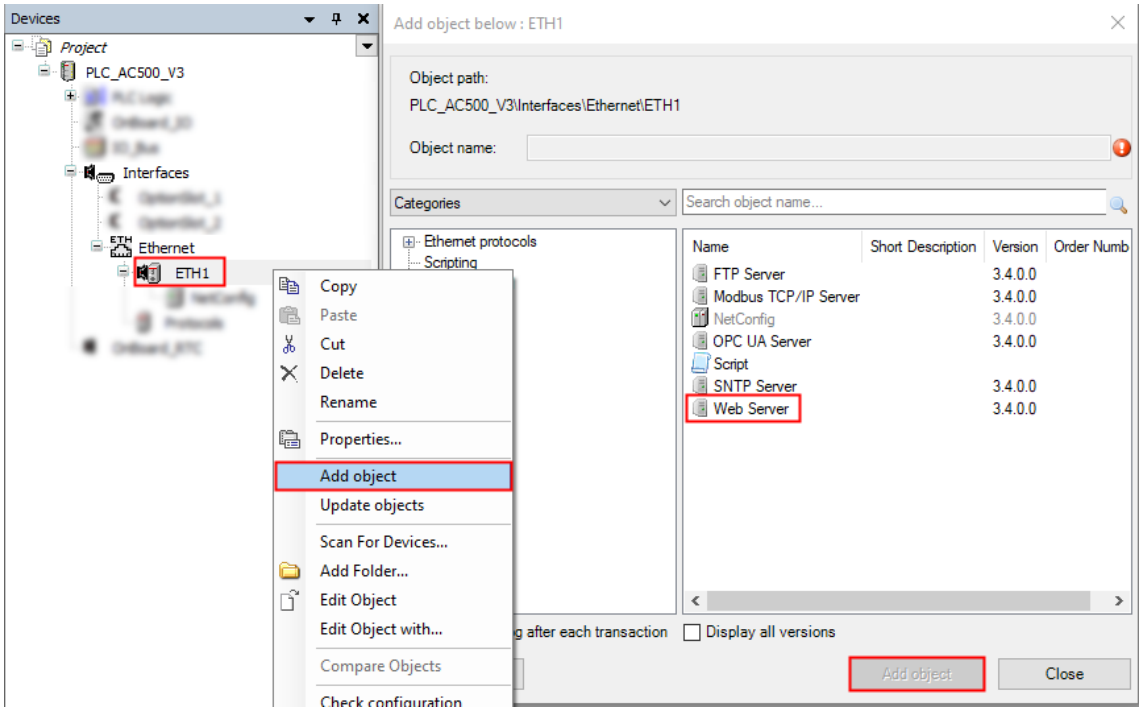
▷ Operate the switches and observe the visualization screen.



Enable web visualization

Add a web server object to the device tree

Ethernet ports can be configured for web server protocol. This description deals with ETH1 configuration for the web server

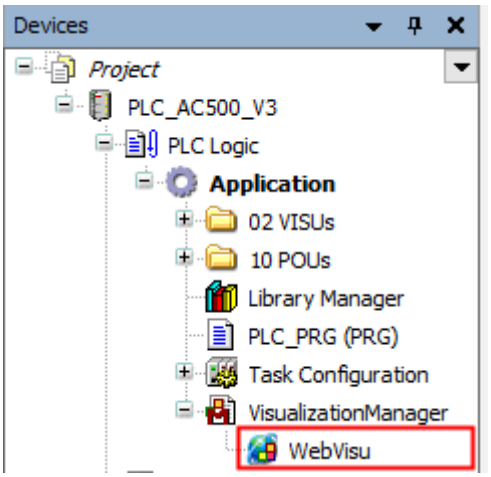


1. Right-click *“ETH1”* in the device tree.
2. Select *“Add object”*.
3. Select *“Web Server”*.

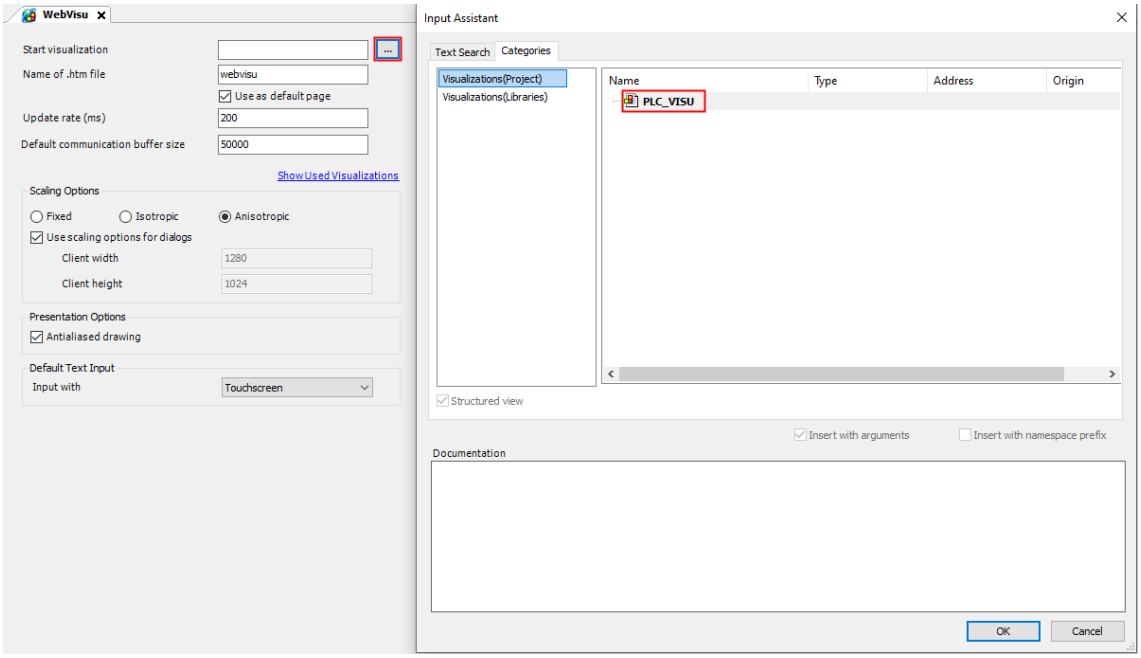
4. Select *“Add object”*.

   ⇨ You added and activated a web server on Ethernet port 1 on the AC500 V3 CPU.

Set-up the web server



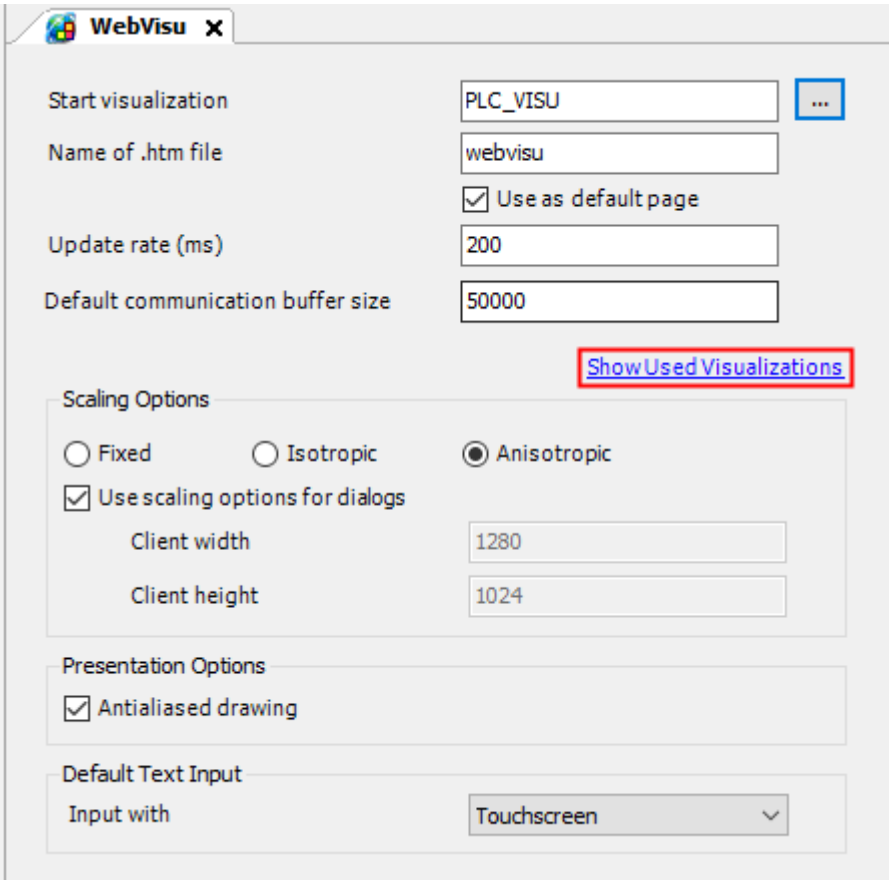
1. Double-click *“WebVisu”* in the device tree.



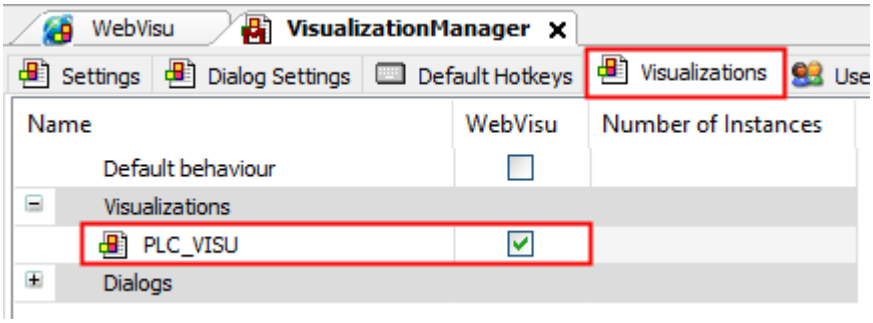
2. Under *“Start Visualization”*, select *“...”*.

   ⇨ A list opens.

3. Select the *“PLC_VISU”* screen from the list.
4. Keep all further settings with default values.



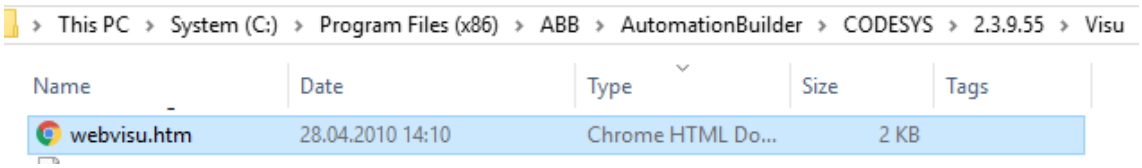
5. Select the link *"Show used visualizations"*.



⇨ The VisualizationManager editor and there the tab *"Visualizations"* opens. All screens and dialog elements created in the project are visible.

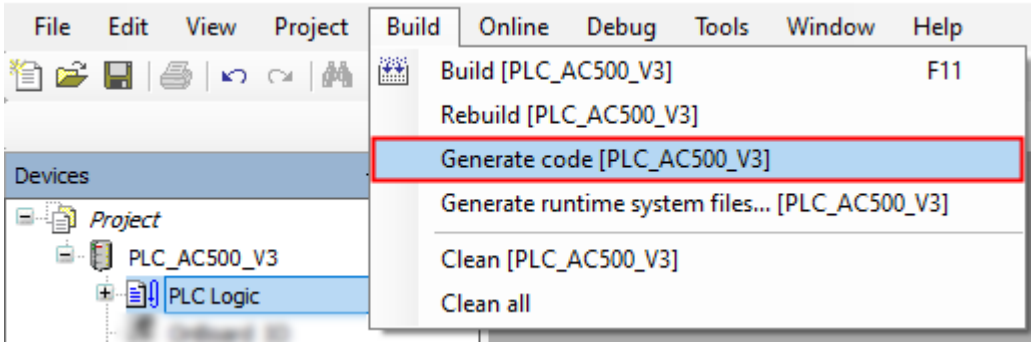
Here, you can select which screens are enabled or disabled for web visualization.

If you want to select another screen as a start visualization, you must modify the adequate parameter in the webvisu.htm file: `<param name="STARTVISU" value="PLC_VISU">`



Compile the project

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

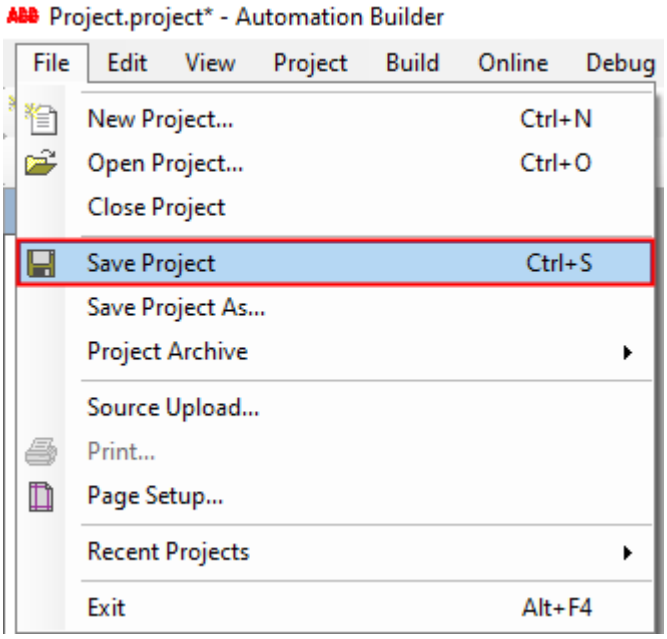


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



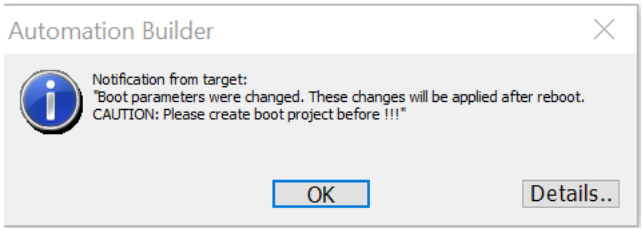
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

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

Loading the project to the CPU

1. Download the project to the CPU *as described in Chapter 1.2.18.1.2.7 , on page 89*.
2. Check the notification window at the end of the download. In case of message "Boot parameters were changed. These changes will be applied after reboot", a reboot of the CPU is required after creation of the boot project.



Create a boot project

By default, after project download, the boot project is created automatically.

Rebooting the CPU

▷ Reboot the CPU by switching OFF and ON the power supply. (The parameter for web server activation is a boot pamater which is loaded during boot of the CPU)

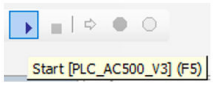
Test the web visualization

☑ You have downloaded the project and created the boot project.

☑ The CPU has been rebooted.

☑ You are logged in.

☑ CPU is in "stop" mode.



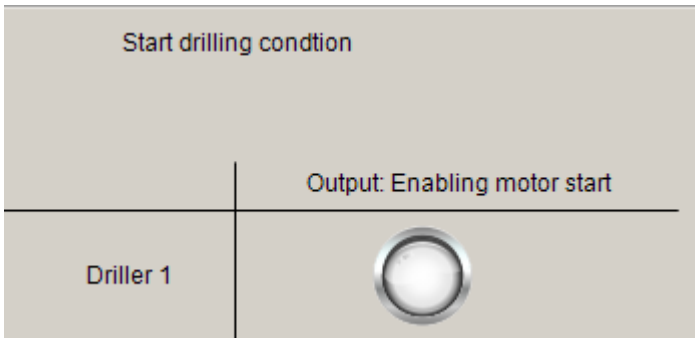
1. Start the project execution, e.g., from the tool bar.
2. Launch an internet browser.
3. Type in the URL field: *http://192.168.0.10/webvisu.htm*.

   192.168.0.10 is the IP address of CPU's ETH1 port.

   /webvisu.htm is the default htm file.

   ⇨ Web visualization will be loaded.

   The start screen "PLC_VISU" is displayed in a responsive view.



4. Test the function by operating switch I1.
5. Test the results for responsive view by changing the web browser window size.

Reset the CPU

Reset values and parameters

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

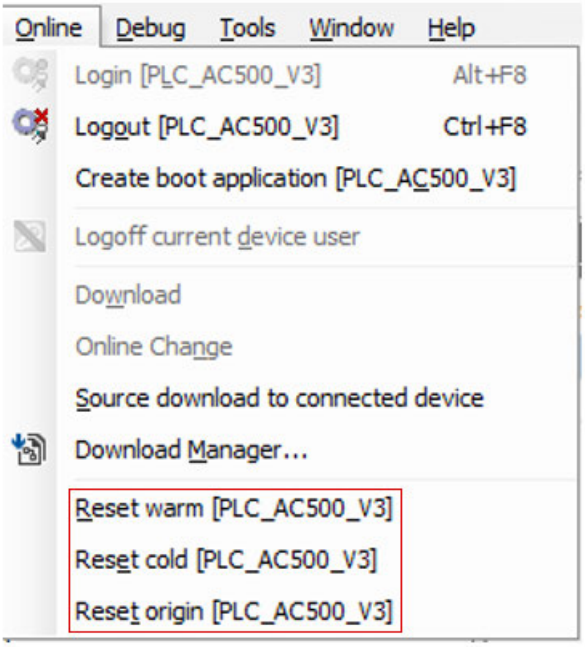


Fig. 8: Reset commands in "Online" menu

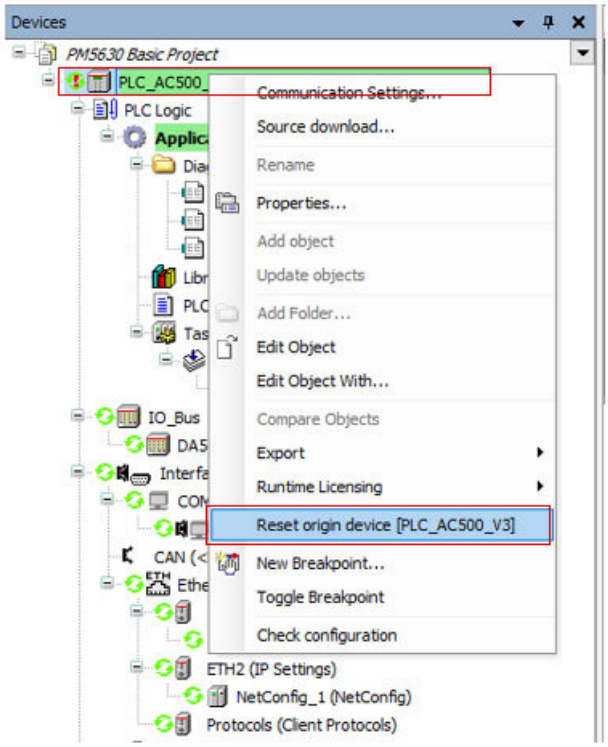
Reset warm	All variables are reset, except RETAIN PERSISTENT variables.
Reset cold	Causes initialization of all variables, except PERSISTENT variables. By recommended creation of remanent variables always with both properties: PERSISITENT and RETAIN, this command resets all variables, except PERSISTENT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

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

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

Complete reset of the CPU

To do a complete reset of the CPU thereby erasing the application from the RAM and flash EEPROM do the following.



1. Right-click the station object *"PLC_AC500_V3"* in the device tree.
2. Select *"Reset origin device [station name]"*.
   ⇨ The application is completely erased from the CPU (complete project from all memory areas).

1.2.18.1.3 Example project for remote I/O expansion with PROFINET

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

Preconditions

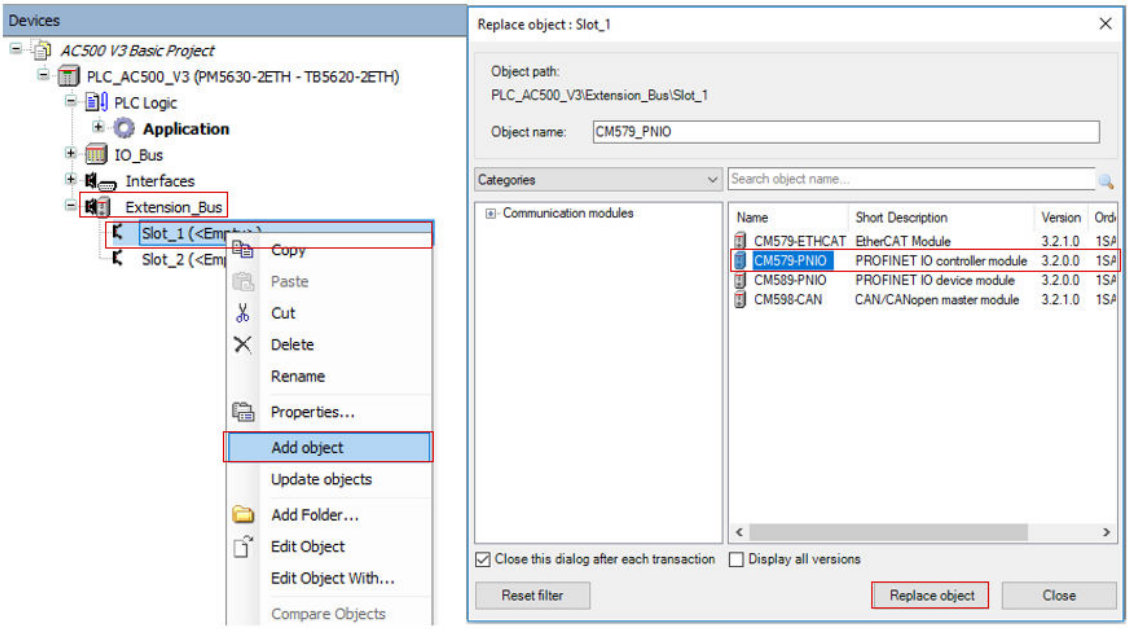
- Automation Builder is installed and licensed as, at least, standard edition ↳ *Chapter 1.2.4 "Managing your licenses" on page 20*.
- AC500 V3 CPU is assembled and connected to the PC ↳ *Chapter 1.2.18.1.1 "Hardware AC500 V3" on page 61*.
- Configuration and programming of this example project will be made in the existing example project for central I/O expansion ↳ *Chapter 1.2.18.1.2 "Example project for central I/O expansion" on page 63*.
- CM579-PNIO communication module is inserted in terminal base and connected to the PLC ↳ *Chapter 1.2.18.1.1 "Hardware AC500 V3" on page 61*.
- CI502-PNIO communication interface module is inserted in terminal unit and connected to the PLC ↳ *Chapter 1.2.18.1.1 "Hardware AC500 V3" on page 61*.

Set-up PROFINET controller

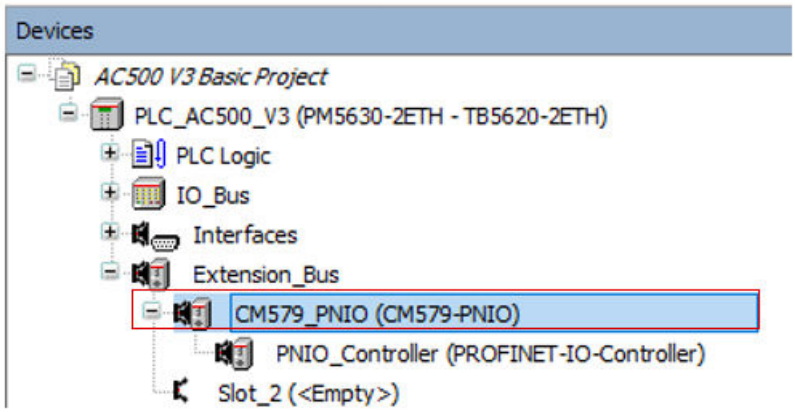
Add the CM579-PNIO to the device tree

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

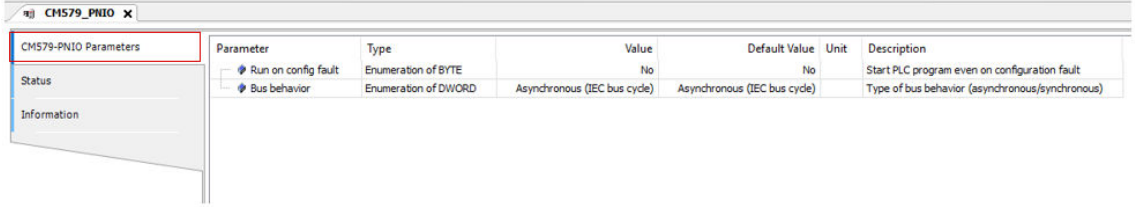
3. Select *“CM579-PNIO”*.
4. Select *“Replace object”* to add the CM579-PNIO.



Set-up the general behavior



1. Under *“Extension_Bus”*, double-click *“CM579_PNIO”* in the device tree.
   ⇨ A tab opens in the editor view.
2. Select *“CM579-PNIO Parameters”*.

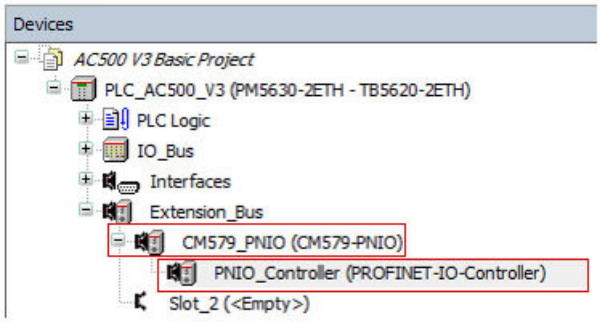


Run on configuration fault	This parameter will prohibit the PLC from running if the CM579-PNIO has a configuration error.
Bus behavior	This parameter sets how the data from the bus flows in/out of the CM579-PNIO.

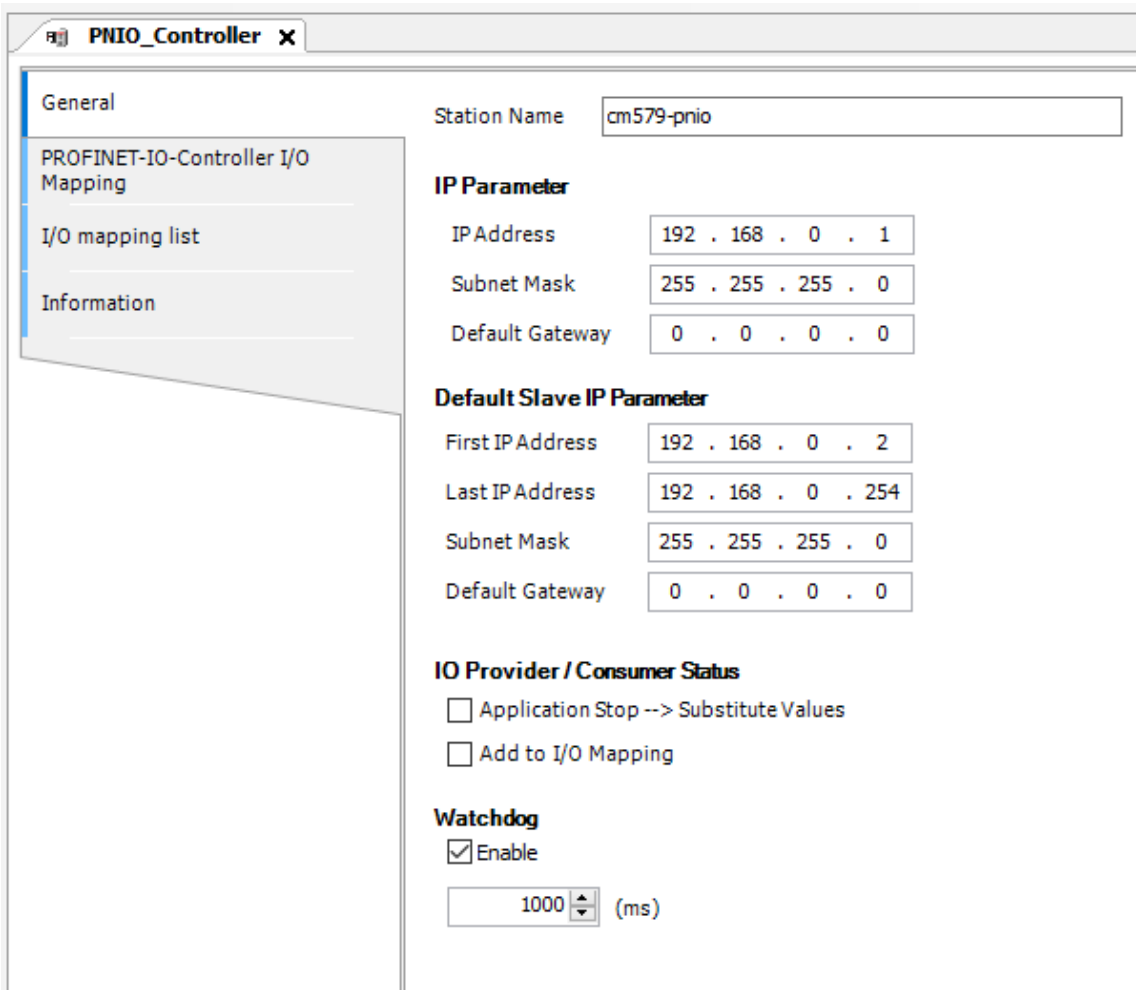
3. Select *“Status”*.
   ⇨ This opens the bus controller status and gives a basic status overview.
4. Select *“Information”*.
   ⇨ This page contains general information about the CM579-PNIO.
5. For the example project, you can keep the default settings.

Set-up the PROFINET IO controller

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



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



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

   For the example project, keep the default settings.

Set-up PROFINET device

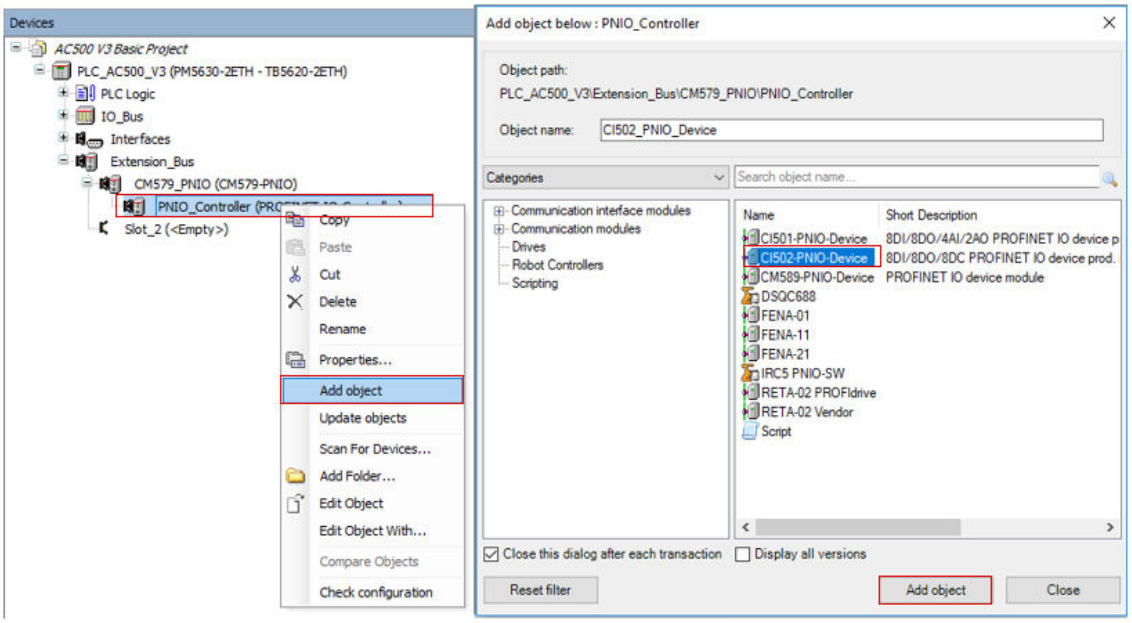
Hardware preparation

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

Add the CI502-PNIO to the device tree

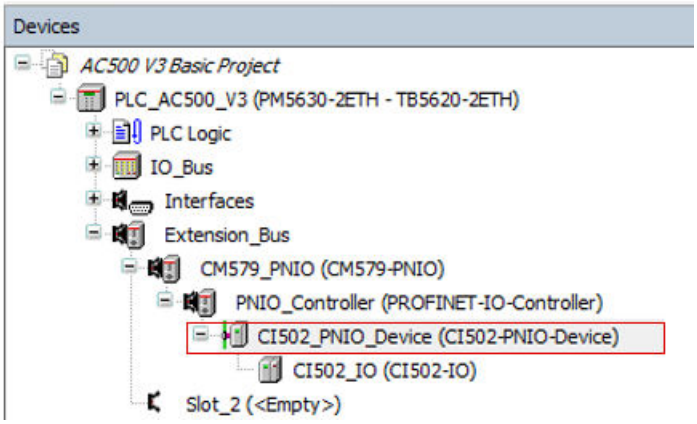
1. Right-click *“PNIO_Controller”* in the device tree.
2. Select *“Add object”*.
3. Select *“CI502-PNIO-Device”*.

4. Select *“Add object”* to add the device.



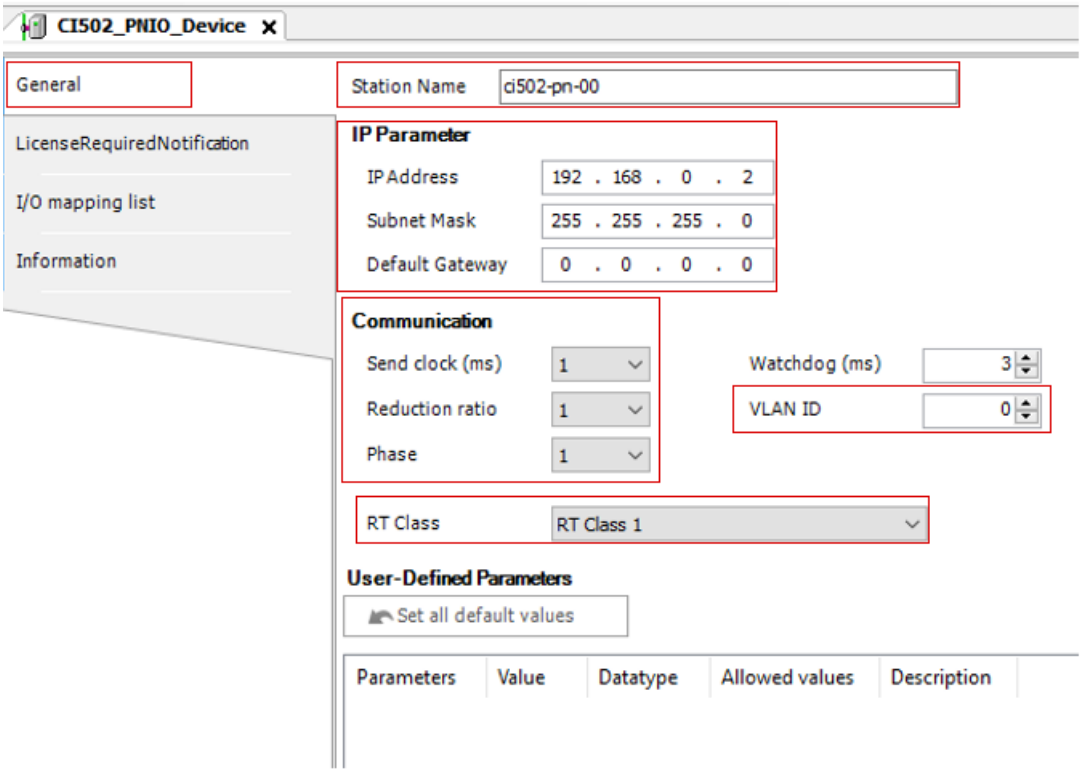
Configure the CI502-PNIO device

Configure the CI502-PNIO PROFINET IO device



1. Double-click *“CI502_PNIO_Device”*.
   ⇨ A tab opens in the editor view.

2. Select *“General”*.



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

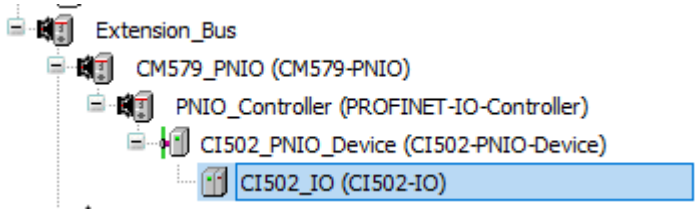
3. Set station name to "ci502-pn-02" according to hardware settings.

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

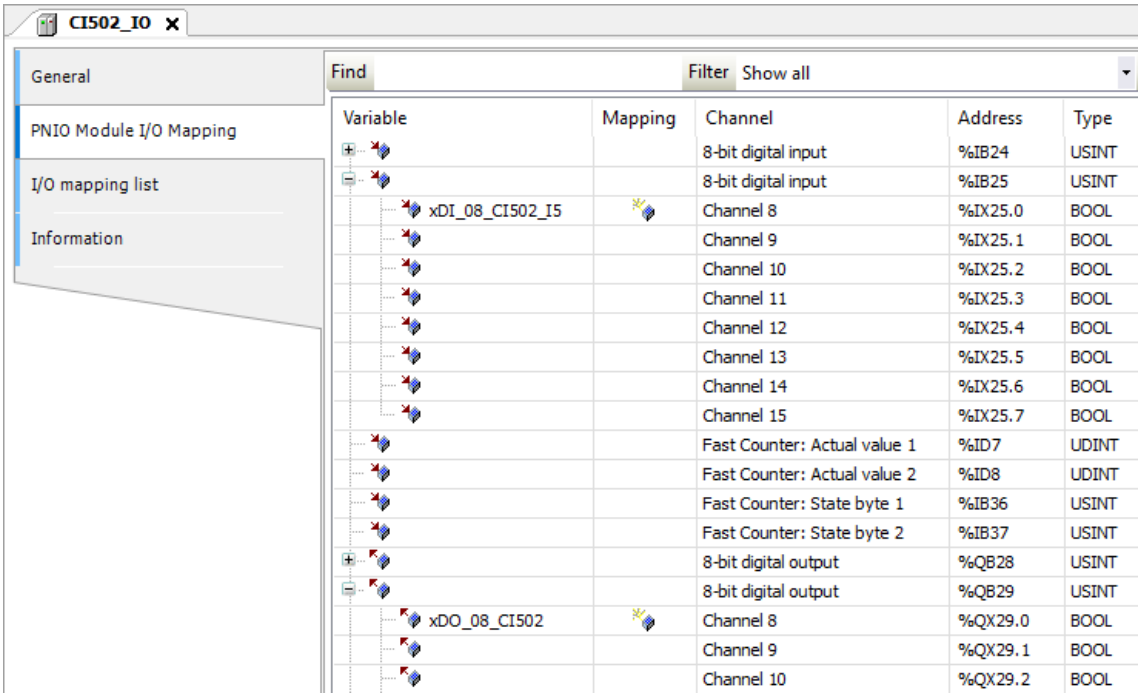
4. Leave the default settings for *“IP Parameter”*.
5. Adjust the communication time settings to get a Watchdog (ms) 24:
   - *“Send clock (ms)”*: 4
   - *“Reduction ratio”*: 2
   - *“Phase”*: 1
6. Leave the default settings for *“VLAN ID”*.
7. Leave the default settings for *“RT Class”*.

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

Create CI502-PNIO I/O mapping to symbols



1. Double-click *"CI502_IO"*.

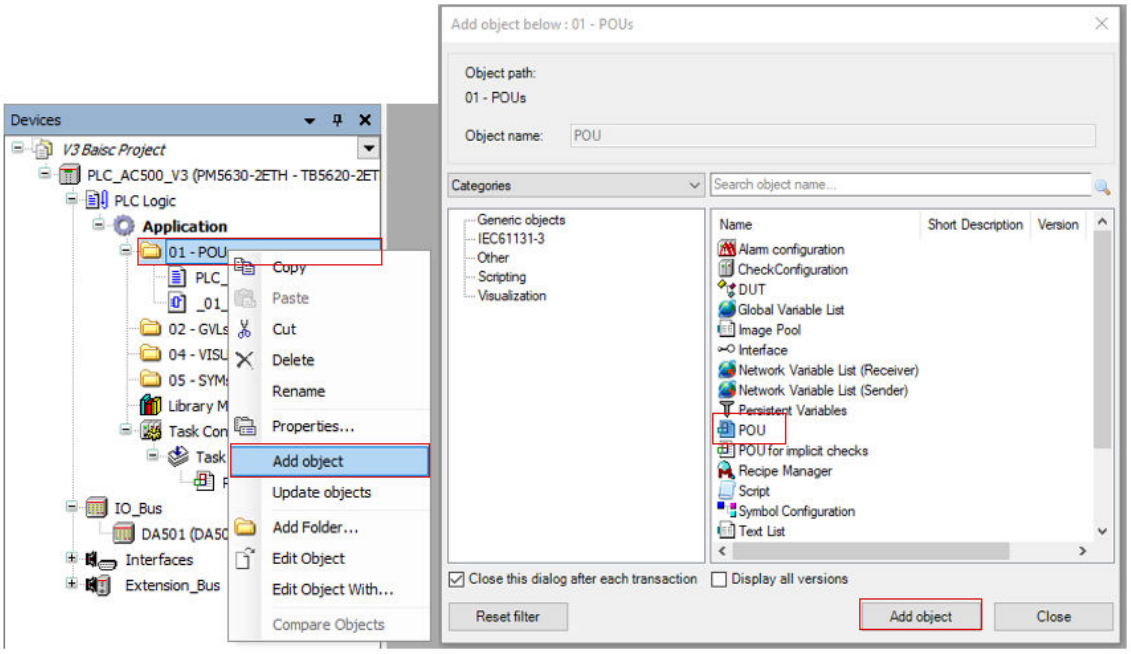


2. Select *"PNIO Module I/O Mapping"*.
3. Fill in the variable names:

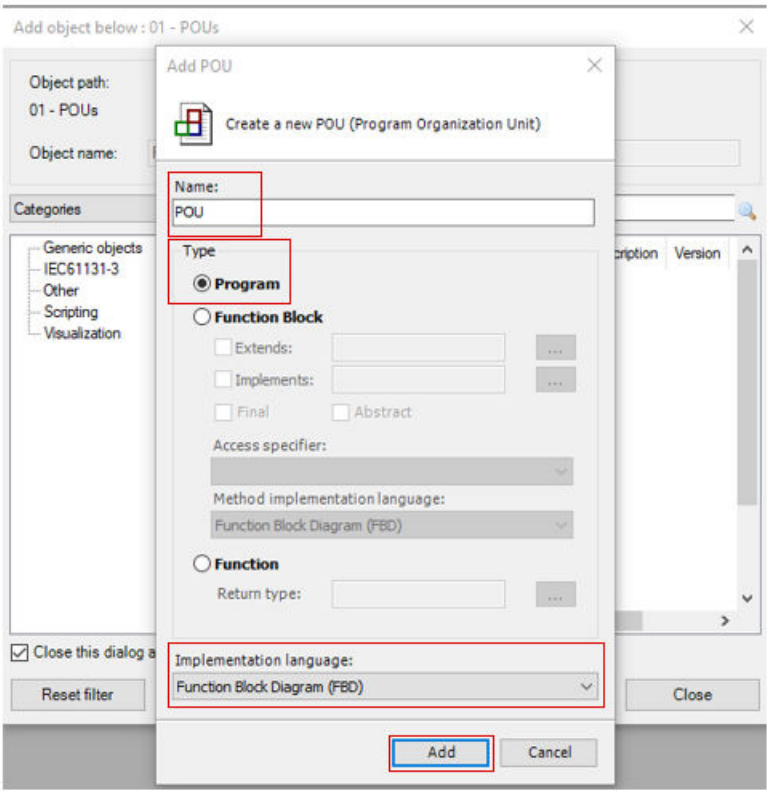
Element	Hardware channel	Symbol
Switch I5	CI502 DI8	xDI_08_CI502_I5
LED output DO8	CI502 DO 8	xDO_08_CI502

Add remote I/O expansion to project

Add a program POU to the project



1. Right-click *“01 - POUs”* in the device tree.
2. Select *“Add object”*.
3. Select *“POU”*.
4. Select *“Add object”*.

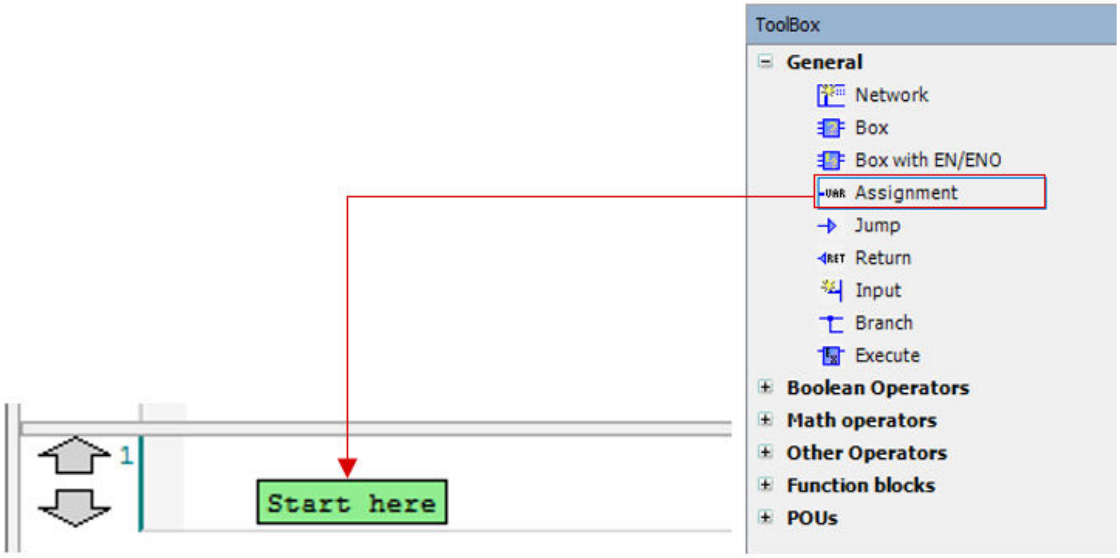


5. Fill in "_30_PNIO_test".
6. Select *“Program”*.
7. Select *“Function Block Diagram”*.

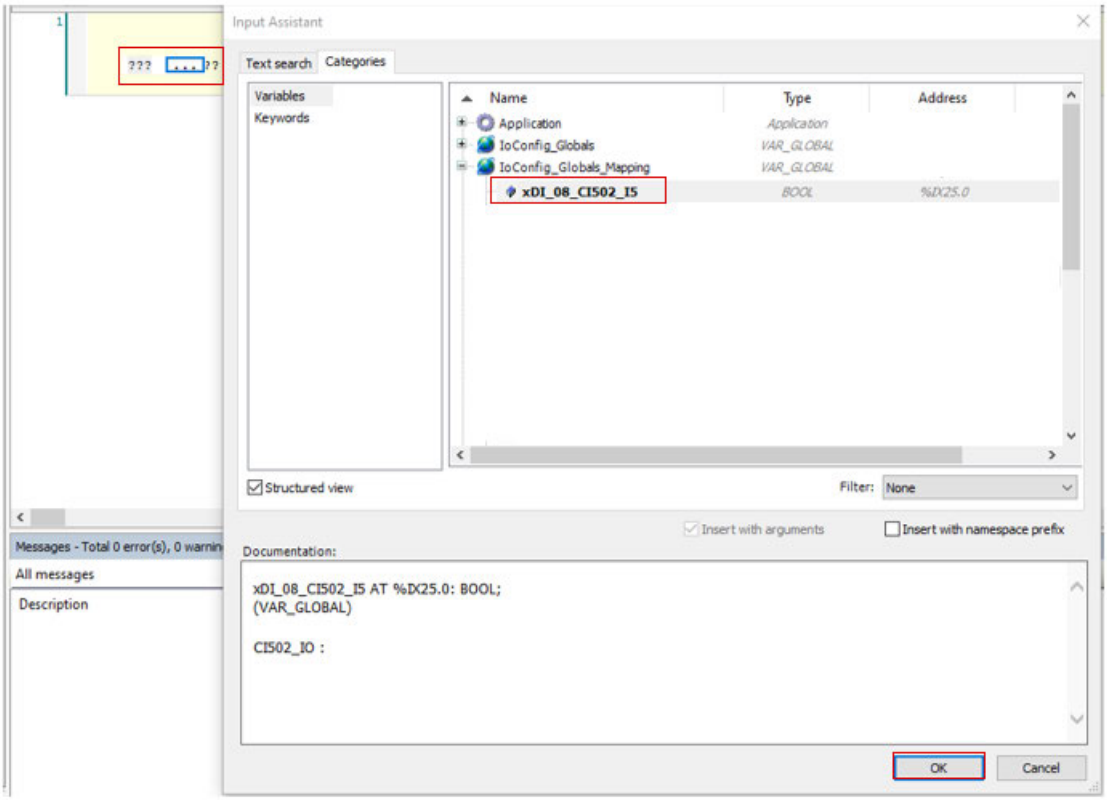
8. Select *[Add]* to add the POU.

Create a POU logic

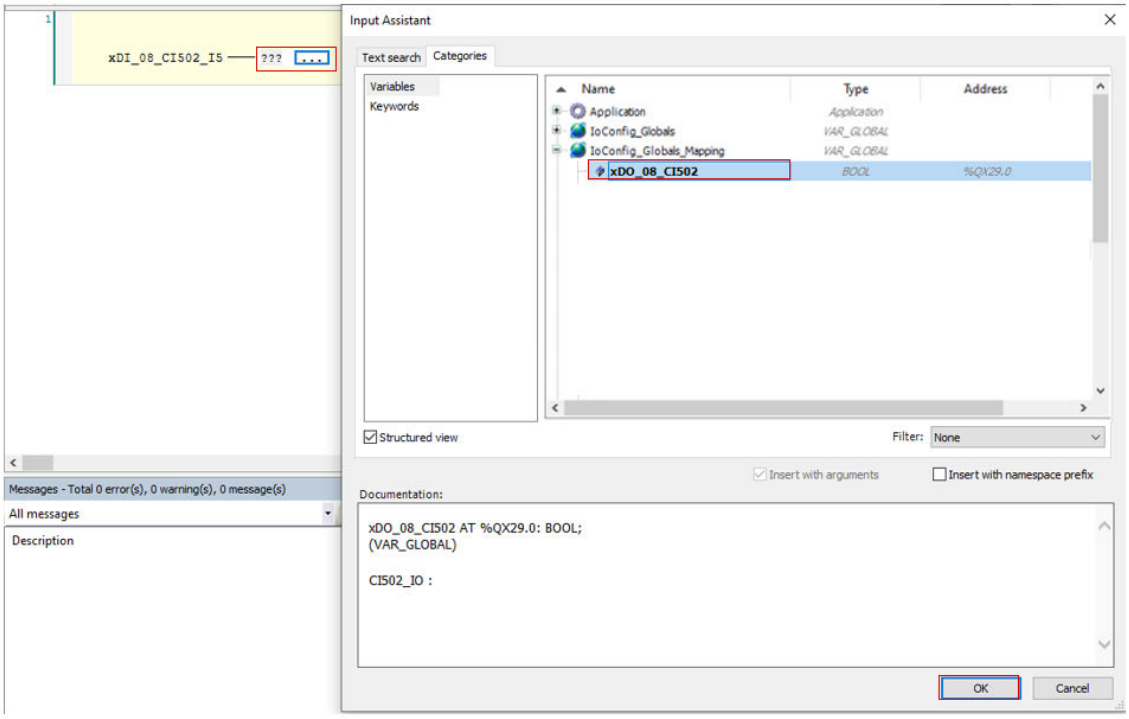
1. Double-click *“30_PNIO_test”* in the device tree.



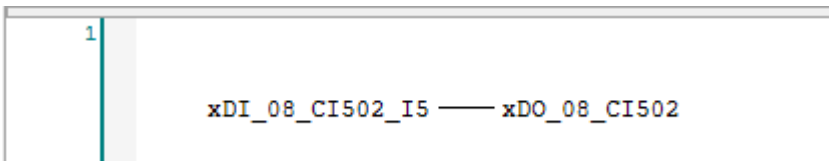
2. In the ToolBox, select *“Assignment”*.
3. Drag and drop *“Assignment”* into the "Start here" field in network "1".



4. Select *“???”* on the left side of the assignment, then select *“...”*.
5. In *“IoConfig_Globals_Mapping”* list, select *“xDI_08_CI502_I5”*.
6. Select *[OK]* to add this variable to the left side of the assignment connector.

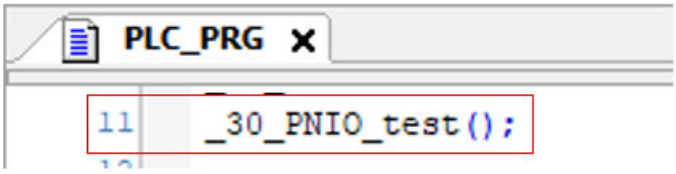


7. Select *"???"* on the right side of the assignment, then select *"..."*.
8. In *"IoConfig_Globals_Mapping"* list, select *"xDO_08_CI502"*.
9. Select *[OK]*.



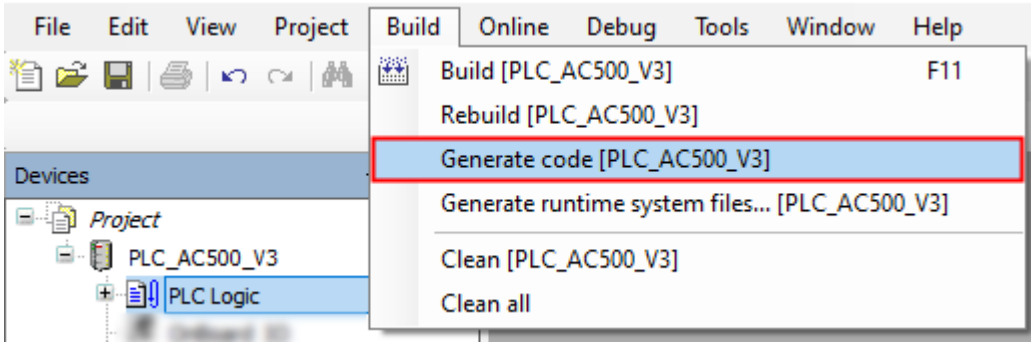
Call the POU in PLC_PRG

1. Double-click *"PLC_PRG"*.
2. Select the next free line in *"PLC_PRG"* and press *[F2]*.
   ⇨ *"Input Assistent"* opens.
3. Select *"Module Calls"*.
4. Open *"Application"*.
5. Open *"10 POUs"* and select *"_30_PNIO test"*.
6. Select *[OK]* to close the dialog.



Compile the project

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

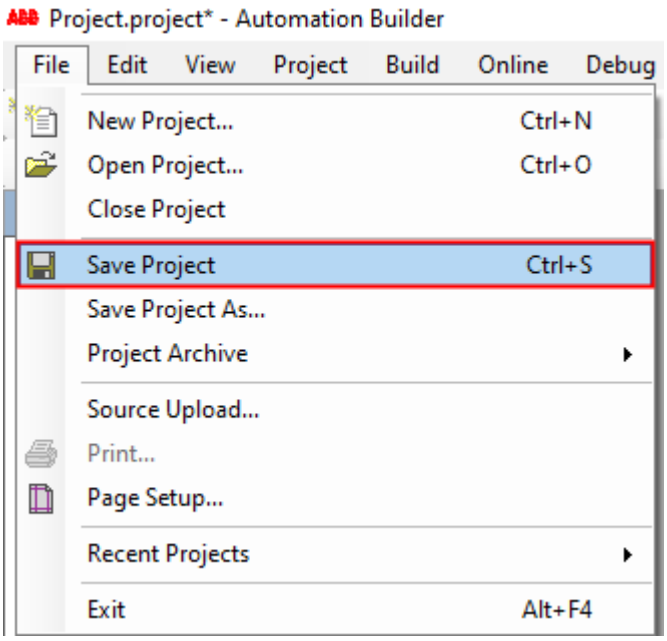


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



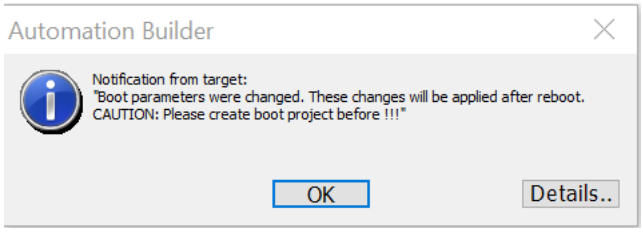
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

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

Loading the project to the CPU

1. Download the project to the CPU ↬ *as described in Chapter 1.2.18.2.2.7 , on page 148*.
2. Check the notification window at the end of the download. In case of message "Boot parameters were changed. These changes will be applied after reboot", a reboot of the CPU is required after creation of the boot project.



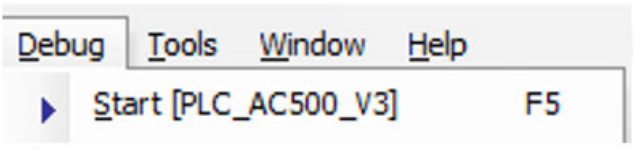
Test the program

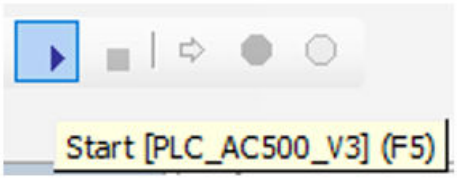
Start the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in "stop" mode.





▷ Select menu *"Debug ➔ Start [PLC_AC500_V3]"*.

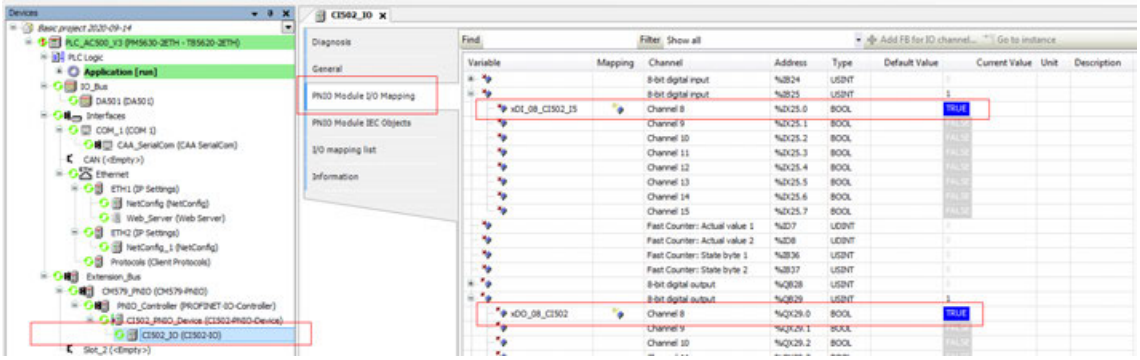
Alternatively, select the "start" icon in the tool bar.

Alternatively, press *[F5]*.

Test the function

▷ Operate the switch I5 and observe:

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



Reset the CPU

Reset values and parameters

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

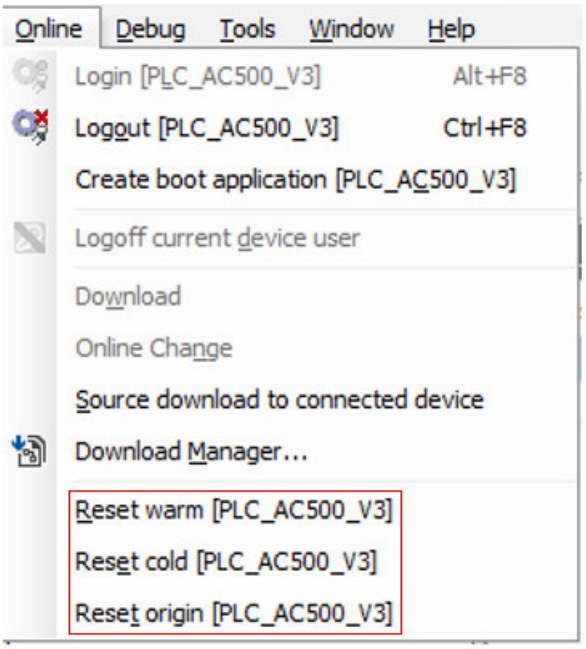


Fig. 9: Reset commands in “Online” menu

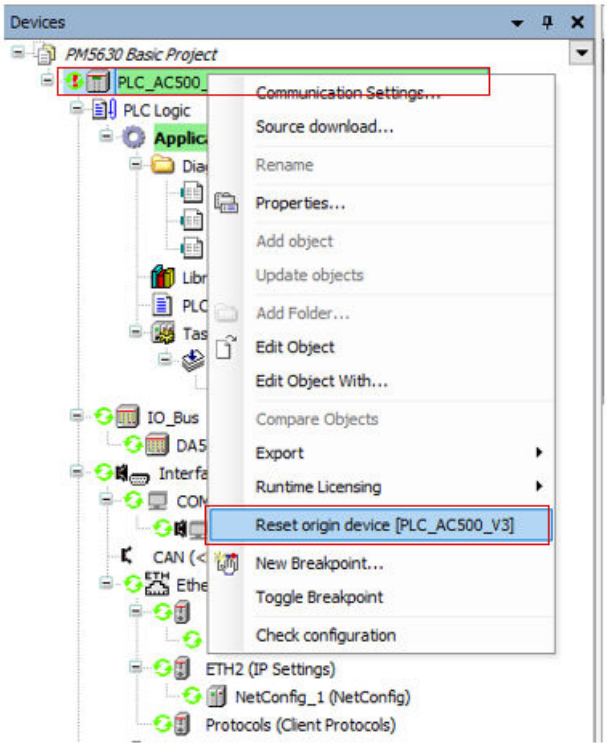
Reset warm	All variables are reset, except RETAIN PERSISTENT variables.
Reset cold	Causes initialization of all variables, except PERSISTENT variables. By recommended creation of remanent variables always with both properties: PERSISITENT and RETAIN, this command resets all variables, except PERSISTENT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

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

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

Complete reset of the CPU

To do a complete reset of the CPU thereby erasing the application from the RAM and flash EEPROM do the following.



1. Right-click the station object *"PLC_AC500_V3"* in the device tree.
2. Select *"Reset origin device [station name]"*.
   ⇨ The application is completely erased from the CPU (complete project from all memory areas).

1.2.18.2 Example projects for AC500-eCo V3

1.2.18.2.1 Hardware AC500-eCo V3

Configuration for example projects

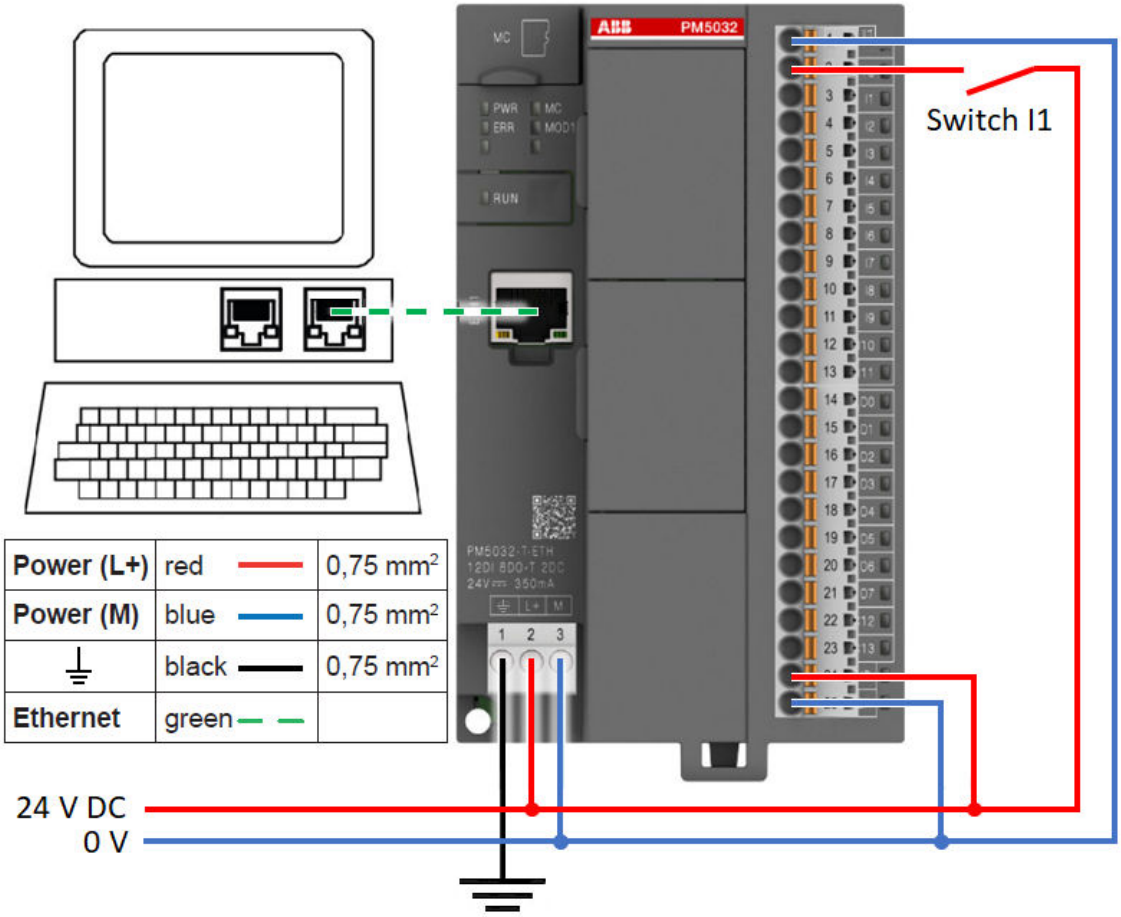
The example projects require a AC500-eCo V3 CPU. The onboard I/O channels are used.

The visualization example is running on CPUs as of PM5032-T-ETH.

Table 7: Modules for example projects to get started with AC500 V3 PLC

Product name	Type	First project
PM5032-T-ETH	CPU	x

Electrical connection



System assembly, construction and connection

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

You can mount AC500 PLC either to DIN rail or to a metal plate ↬ *Chapter 1.6.4.6.3 “Mounting and demounting” on page 3408*. Here, we recommend to mount on DIN rail.

1. Snap the terminal base onto DIN rail.
2. If needed, remove option board slot covers from the CPU and insert option boards.
3. If needed, snap the additional I/O modules onto DIN rail and slide them on the rail to establish the I/O bus connection.

4.

> *The terminal blocks are not included in the scope of delivery.*
>
> *The terminal blocks have to be ordered separetellly according to the CPU type and the type of terminal blocks needed (screw or spring technology).*

Insert terminal blocks for power and I/O connection to CPU, options and I/O modules.

5. Make the sensor/actuator wire connections according to the dedicated electronic module you want to use. Provide external process power supply as required.
6. Connect a programming cable (Ethernet cable between ETH port of CPU and PC with engineering software).

1.2.18.2.2 Example project

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

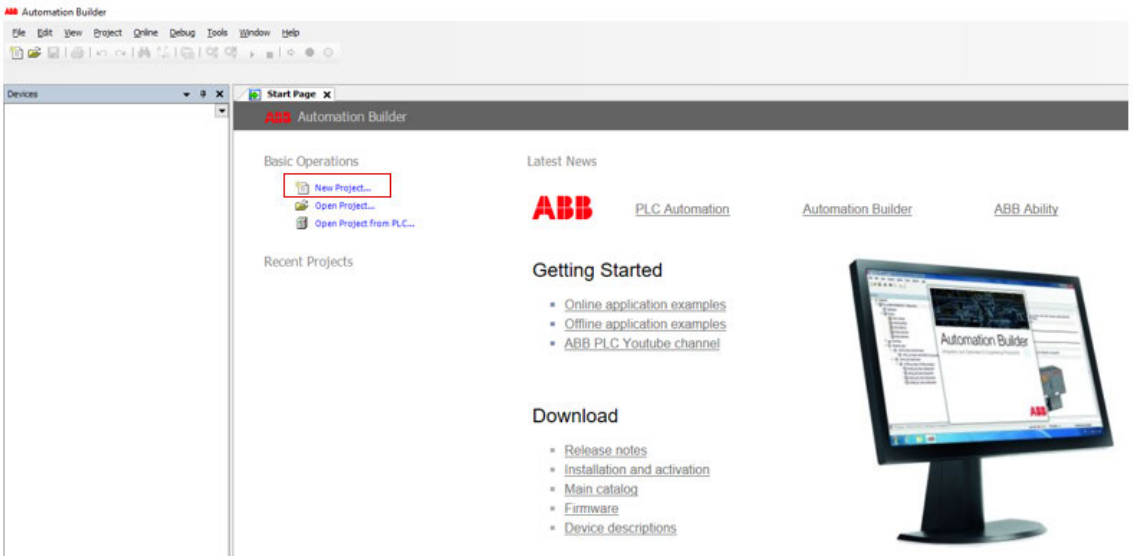
Preconditions

- Automation Builder is installed and licensed as, at least, basic edition ↬ *Chapter 1.2.4 “Managing your licenses” on page 20*.
- AC500 V3 CPU is assembled and connected to the PC ↬ *Chapter 1.2.18.2.1 “Hardware AC500-eCo V3” on page 122*.

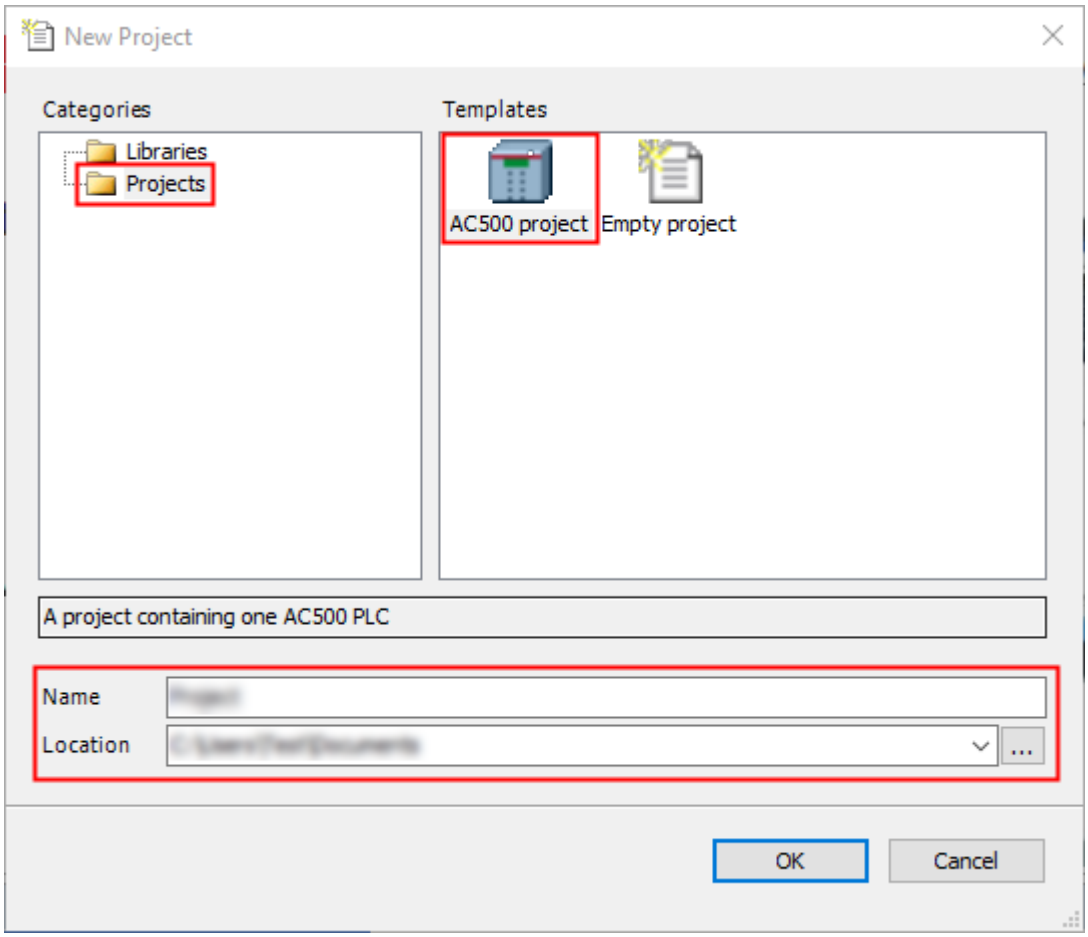
Create, set-up and save your AC500 V3 project

Create a project

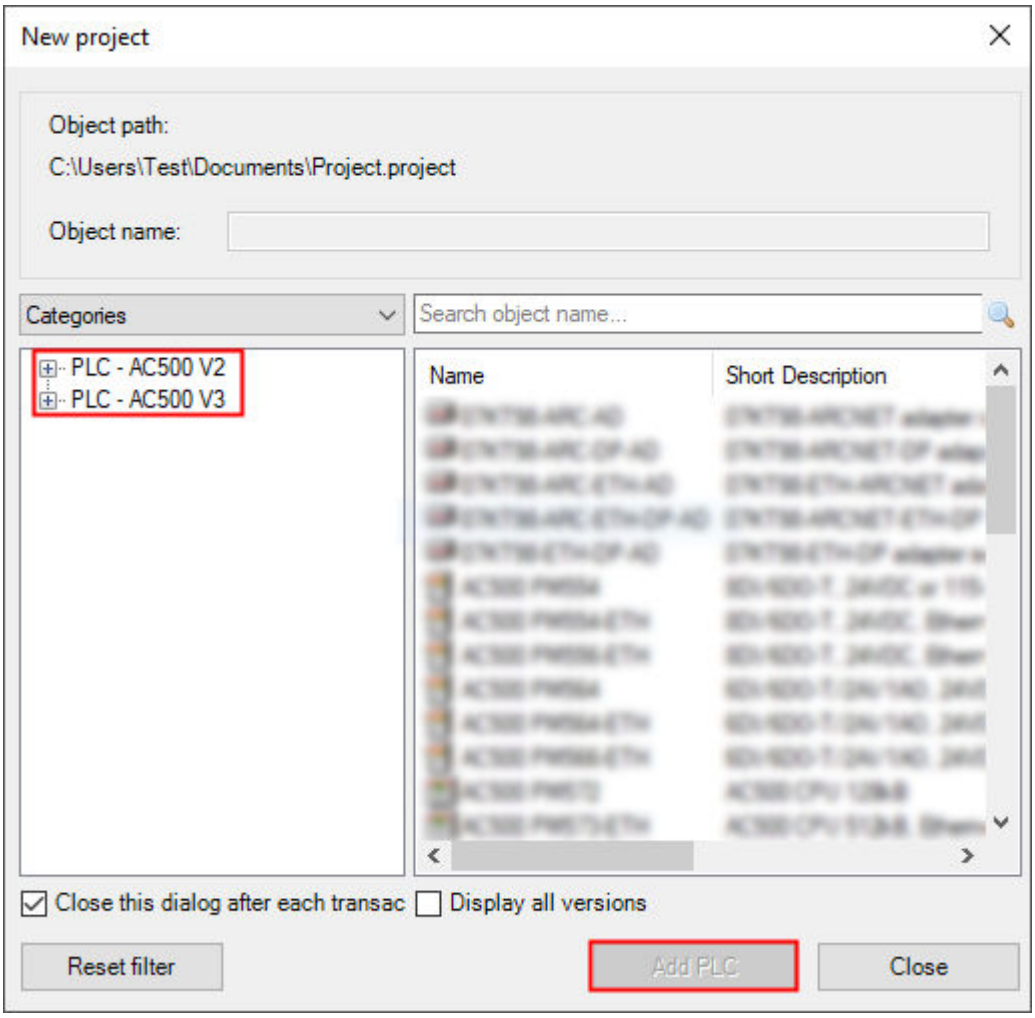
1. Launch Automation Builder either out of the desktop icon or out of the Windows menu.



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



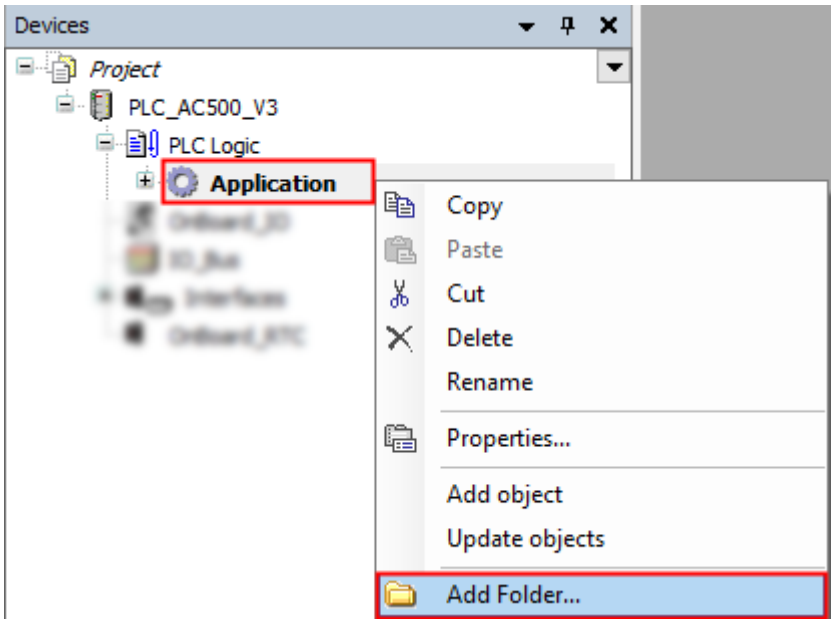
3. Select *“Projects”*.
4. Select *“AC500 project”*.
5. Fill in project name.
6. Choose a location to save the project to.
7. Select *“OK”*.
8. Select *“PLC - AC500 V3”*.
9. Select the CPU according to your hardware set-up.



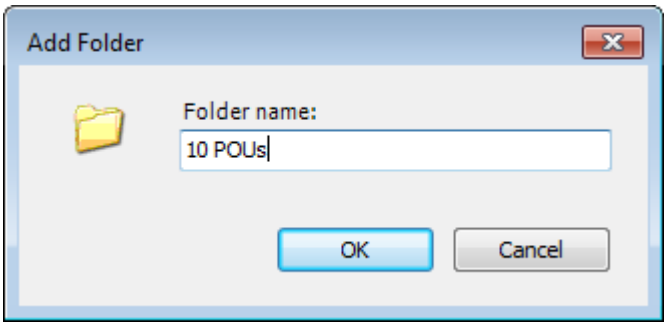
10. Select *"Add PLC"* to add the CPU to your application.

Create folders in the device tree

To optimize the project readability, you will create different folders to group similar objects. The folder names are exemplary. Because the device tree view follows an alphabetical order, we use number prefixes to determine the order.



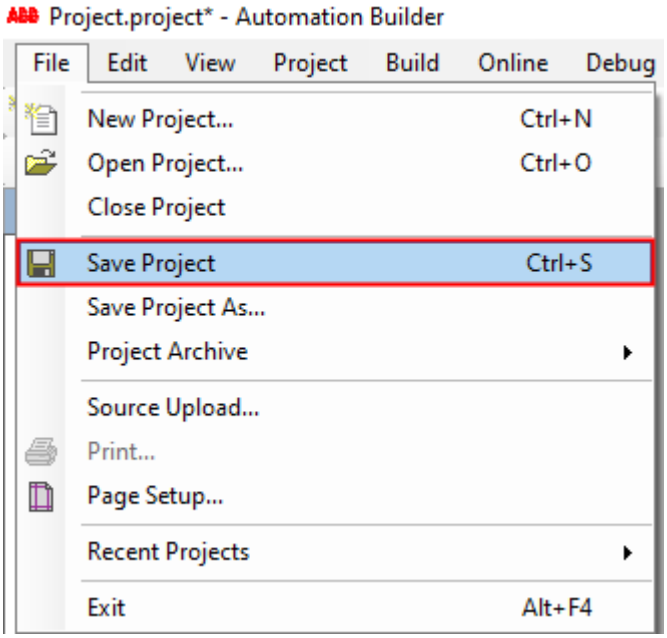
1. Right-click *“Application”*.
2. Select *“Add Folder”*.



3. Type in "10 POUs". This is a name example. Here, the intention is to see this folder as a last one.

   The folder "10 POUs" is for program organization units (POU). POUs are objects of type program, function or function block that are used to create a user program.

Save the project



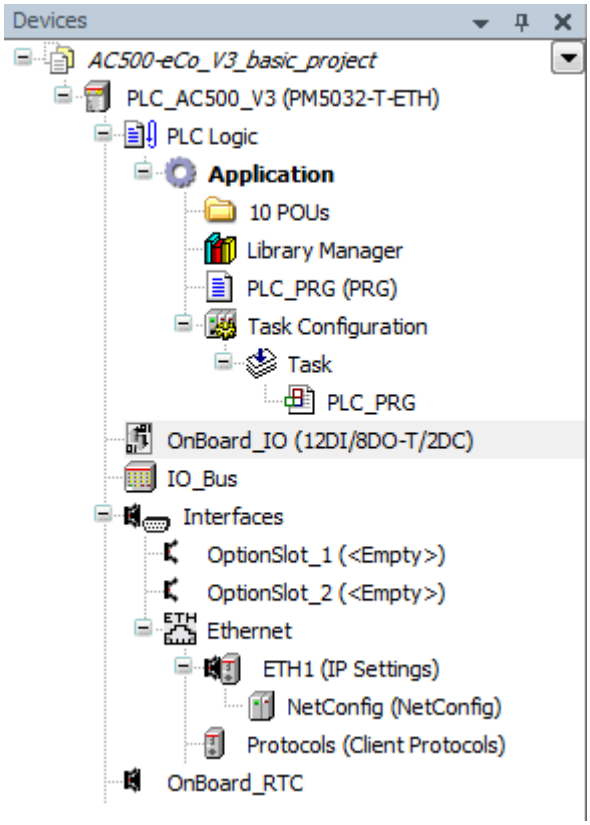
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

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

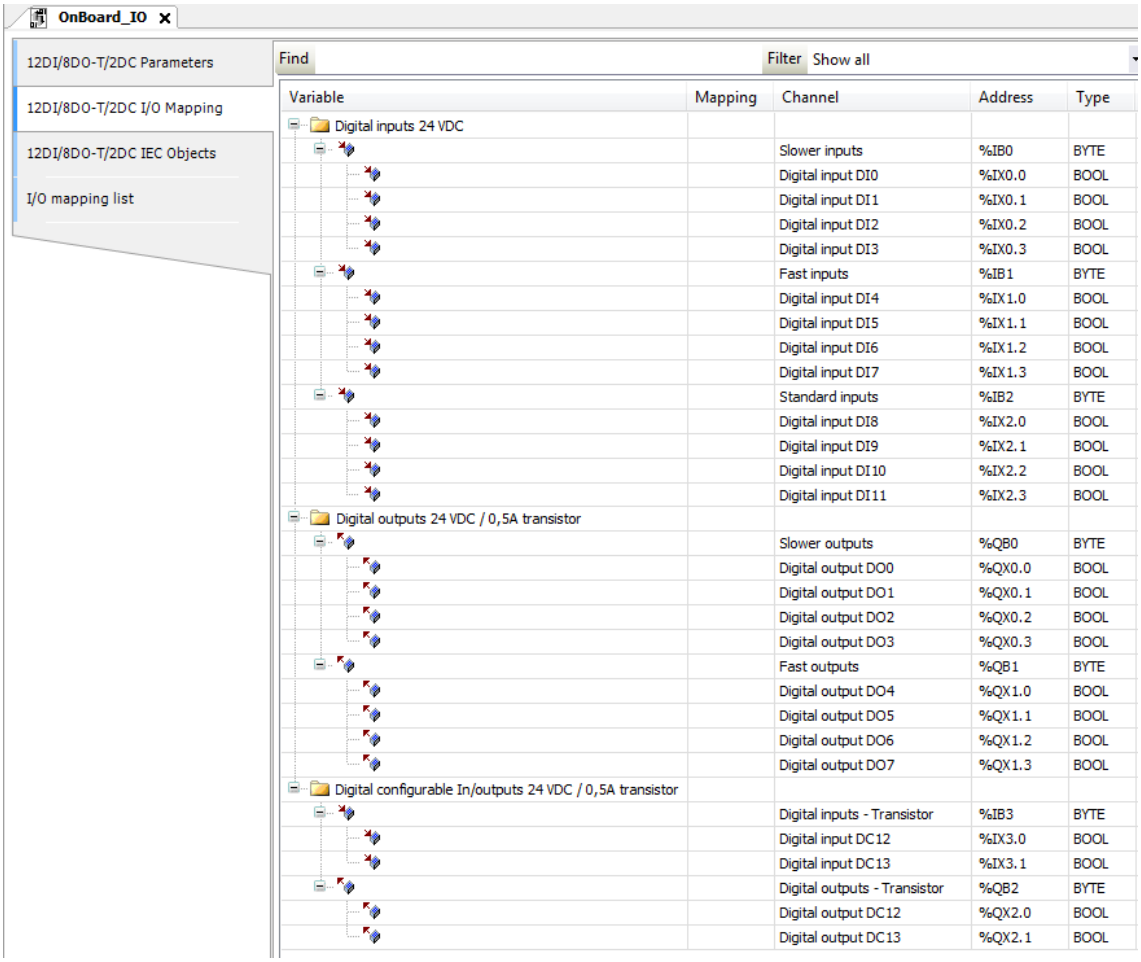
Configure the onboard I/O channels

Onboard I/O variable mapping



1. Double-click *“OnBoard_IO”* in the device tree.

⇨ A tab opens in the editor view.

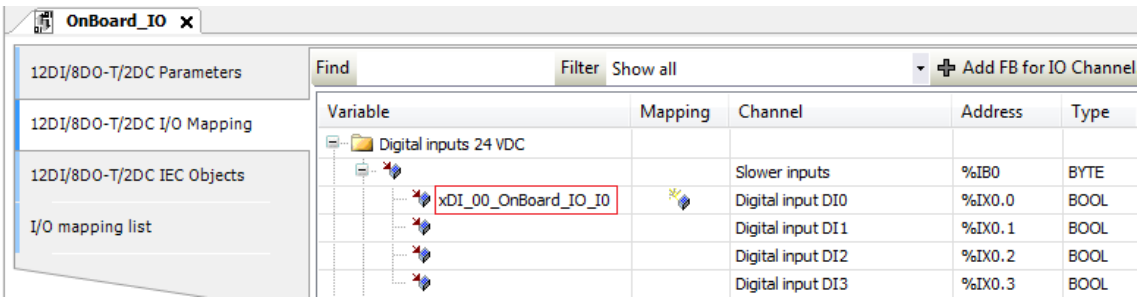


2. Select *“12DI/8DO-T/2DC I/O Mapping”*.

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

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

Handle the digital input variables

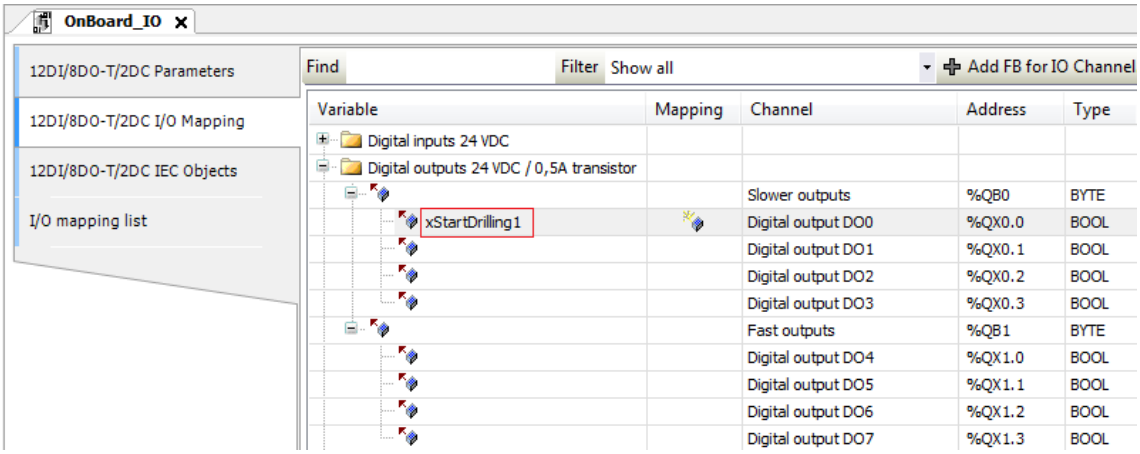


1. Open the list of the digital inputs.

2. Fill in the variable names:

Channel	Type	Variable
Digital input DI0	BOOL	xDI_00_OnBoard_IO_I0

Handle the digital output variables



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

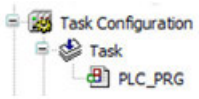
Channel	Type	Variable
Digital output DO0	BOOL	xStartDrilling1

Programming and compiling

Task configuration

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

For this project you will use only one cycling task.



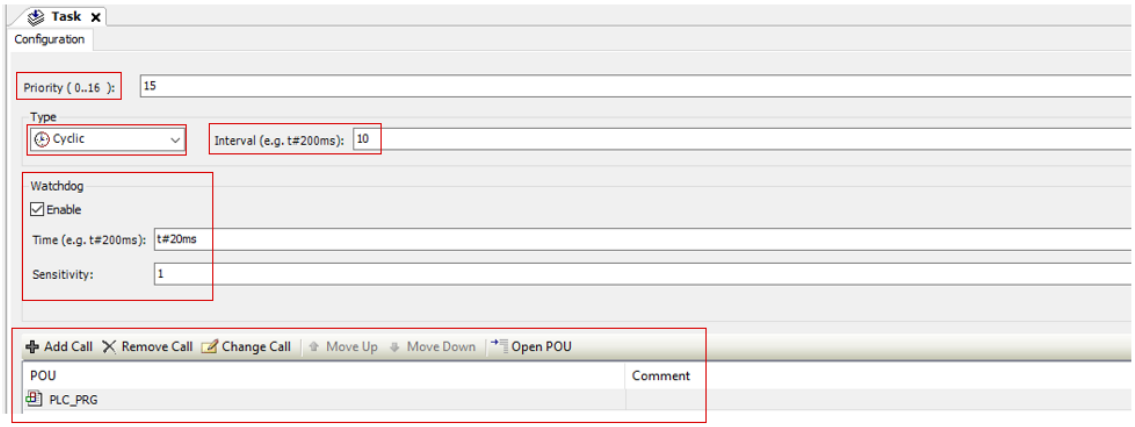
In the device tree, you see the objects *“Task configuration”* and *“Task”*. Both created automatically with the project.

For this project you will use only one cycling task.

▷ Double-click *"Task"* in the device tree.

⇨ A tab opens in the editor view.

For this project you will use only one cyclic task. Keep the default settings for the task.

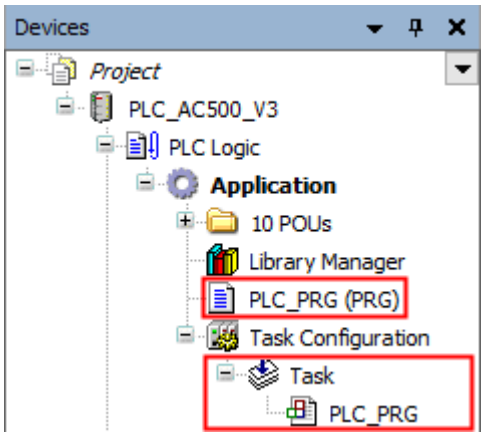


Priority	This is how the CPU prioritizes the task, when more than one task is defined. Priority 0...15 = real time tasks, priority 16 = non-real time task.
Type	In the CPU you can run tasks dependent on the demands of the process
Interval	For cyclic tasks you can set the cyclical execution time. It is usually set in milliseconds with IEC time syntax
Watchdog	To keep track of the time it takes to complete the task
Calls	You can call in one or more program POUs in one single task

Main program PLC_PRG

In the default task configuration ↬ *(shown in chapter 1.2.18.2.2.4.1 Task configuration on page 131)*, there is one call of a POU (program organization unit) i.e. "PLC_PRG".

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



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

To optimize the project readability, you will work with the previously created folder "10 POUs" and add the created subroutines (POUs) to this folder. The subroutines will be created in FBD (Function Block Diagram) editor.

Boolean logic "NOT"

Application example "driller"

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

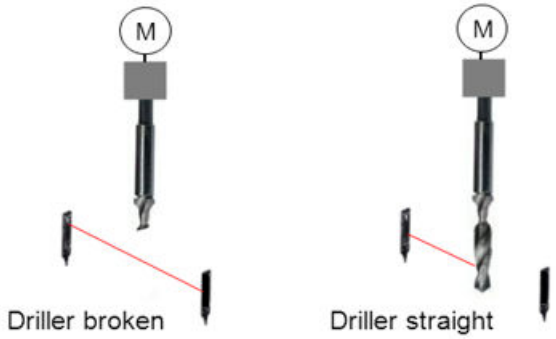


Table 8: Required behavior

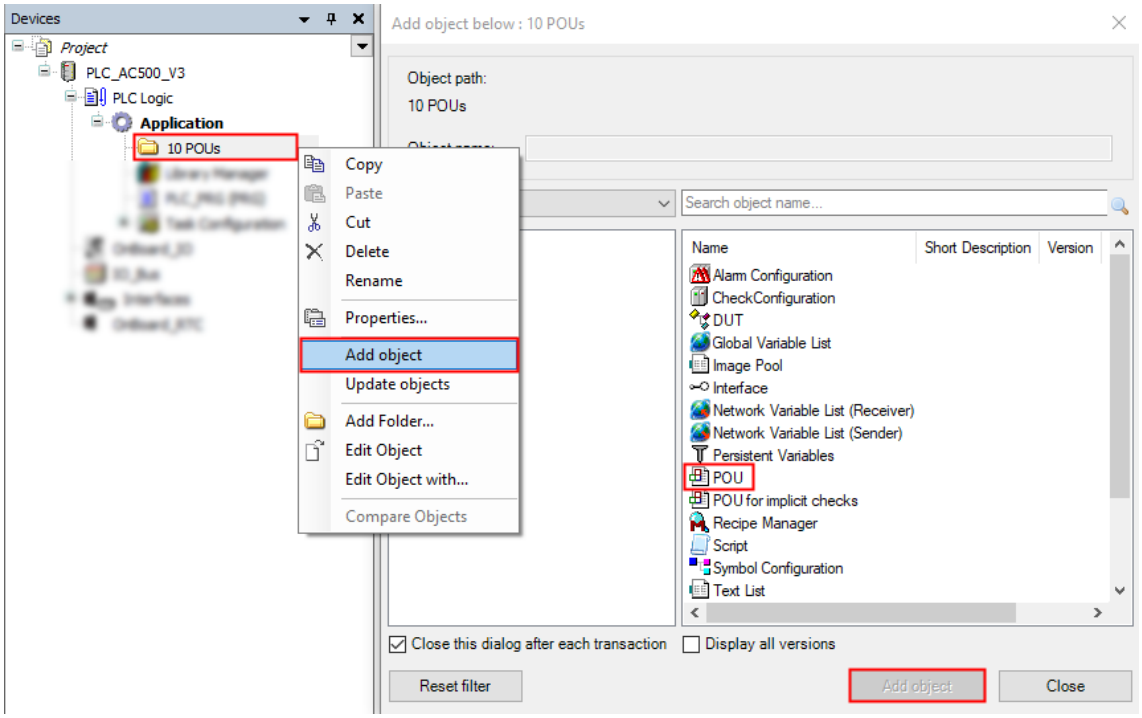
Signal from photo sensor	Required signal of motor ON
FALSE	TRUE
TRUE	FALSE

Table 9: Hardware set-up

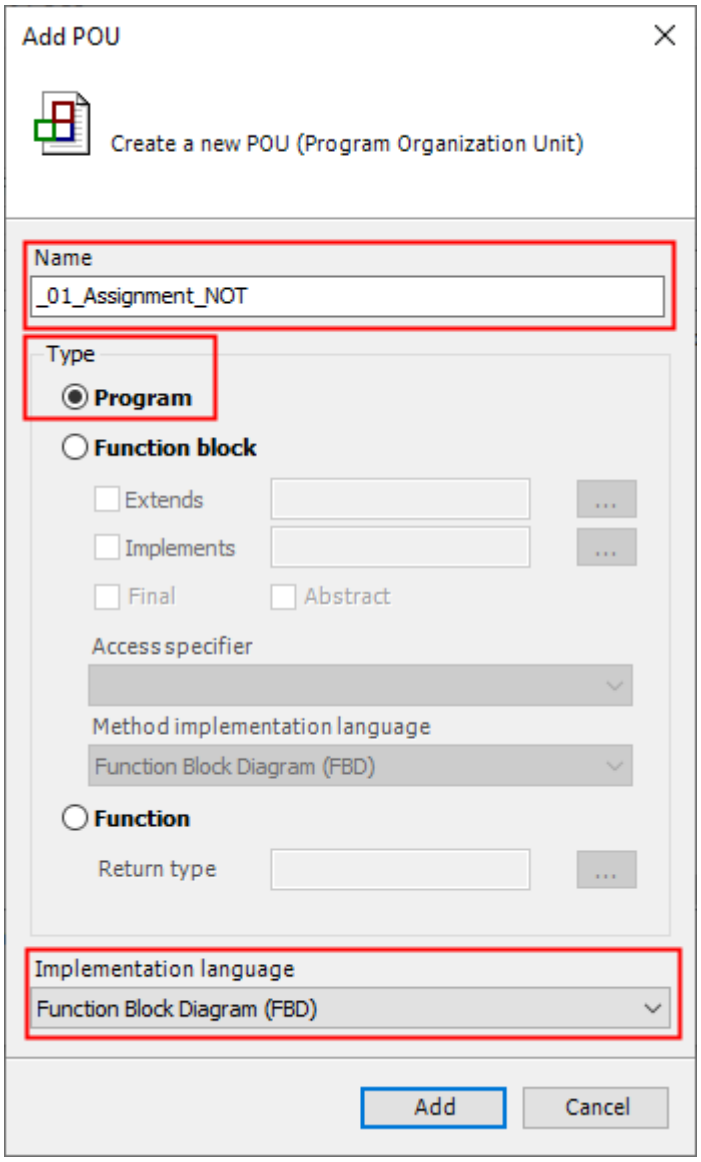
Element	HW channel	Symbol	Description
Switch I1	OnBoard_IO_I0	xDI_00_OnBoard_IO_I0	Photo sensor
LED output DO0	OnBoard_IO_O0	xStartDrilling1	Motor on

Implementation

Create a new program POU in the project



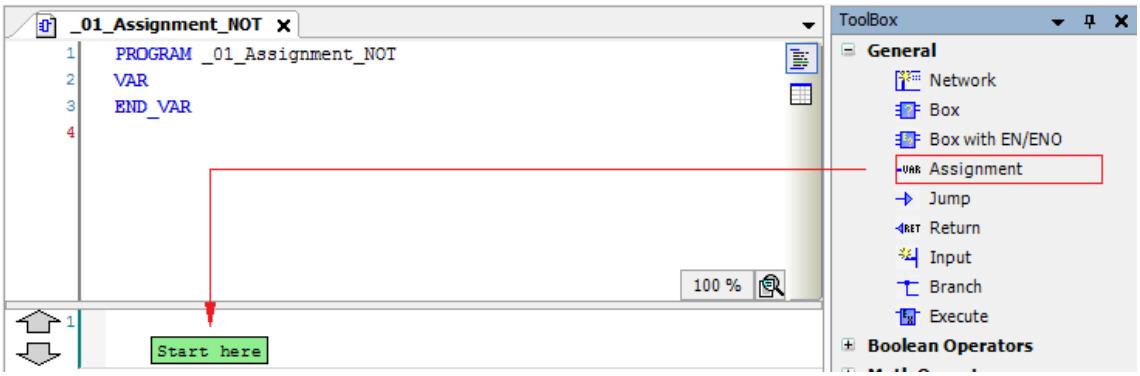
1. Right-click *“10 POUs”*.
2. Select *“Add object”*.
3. Select *“POU”*.
4. Select *“Add object”*.



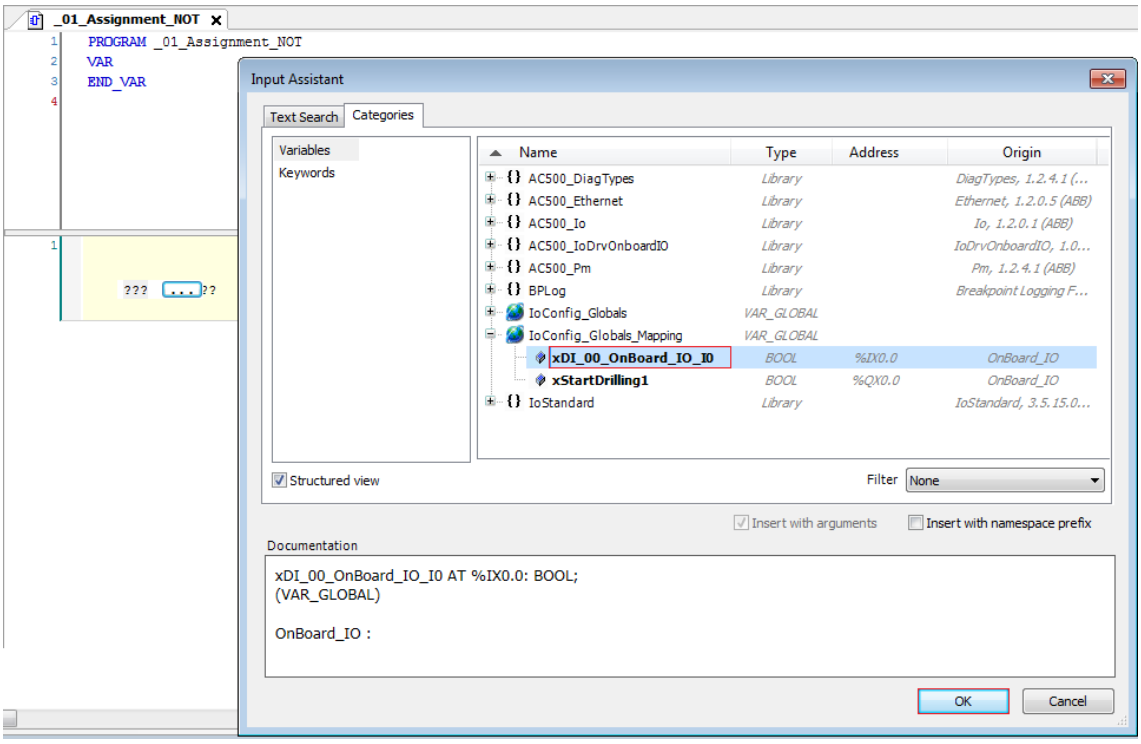
5. Enter *"_01_Assignment_NOT"*.
6. Select *"Program"*.
7. Select *"Function Block Diagram (FBD)"*.
8. Select *"Add"*.
   ⇨ POU has been added.

Assign the hardware DI signals to local variables

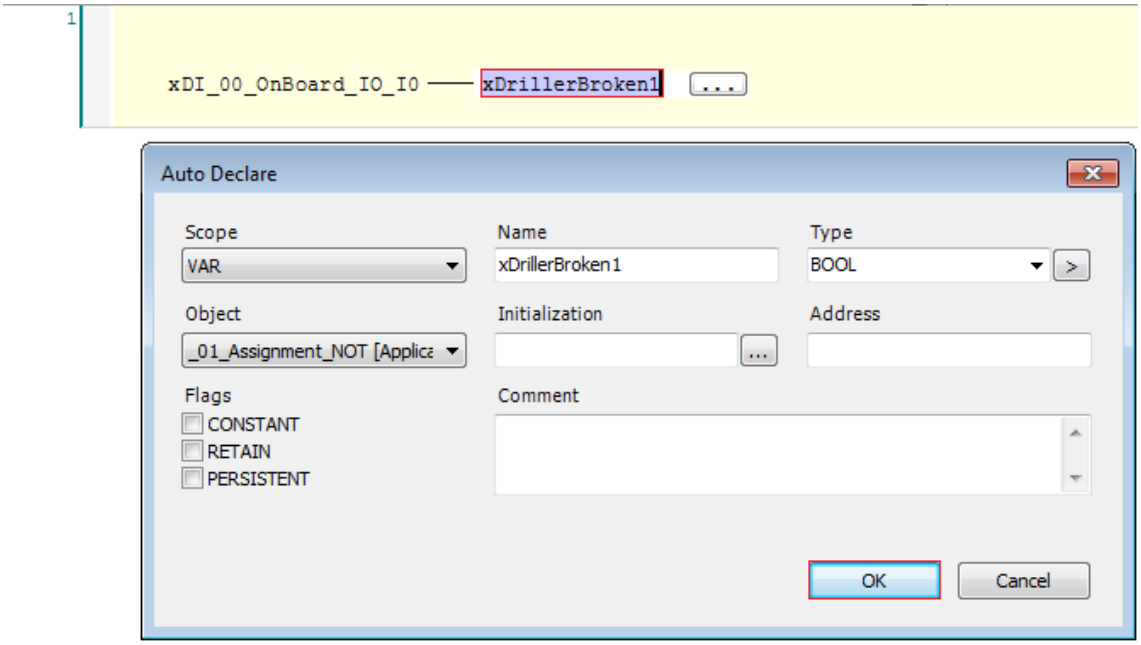
1. Double-click POU*"_01_Assignement_NOT"* in the device tree.



2. Select *"Assignment"* from the ToolBox.
3. Drag and drop *"Assignment"* into the "Start here" field in network *"1"*.



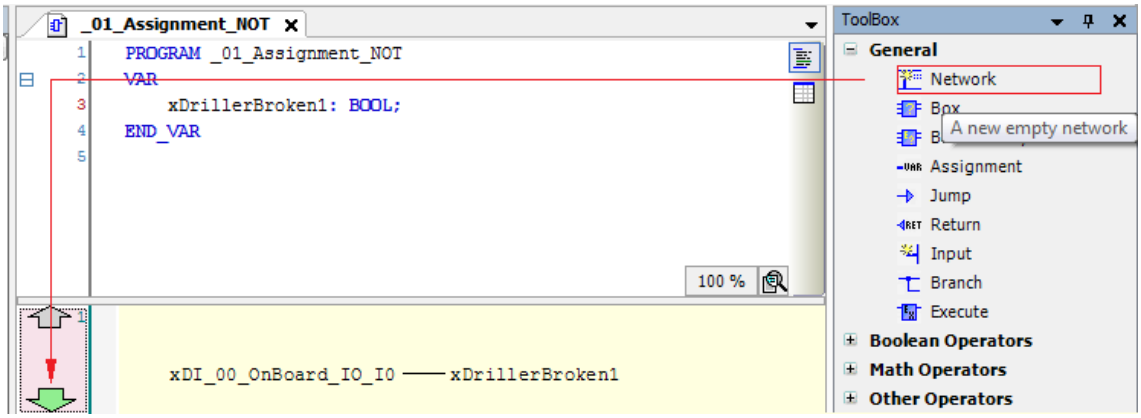
4. Select *"???"* on the left side of the assignment, then select *"..."*.
5. Open the *"Io Config_Globals_Mapping"* mapping list and select "xDI_00_OnBoard_IO_I0".
6. Select *"OK"* to add this variable to the left side of the assignment connector.



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

   ⇨ *"Auto Declare"* opens.

   You see the written variable name and the data type BOOL. The scope is "VAR". It means it is a local variable within this POU.
10. Select *"OK"* to accept the entries.



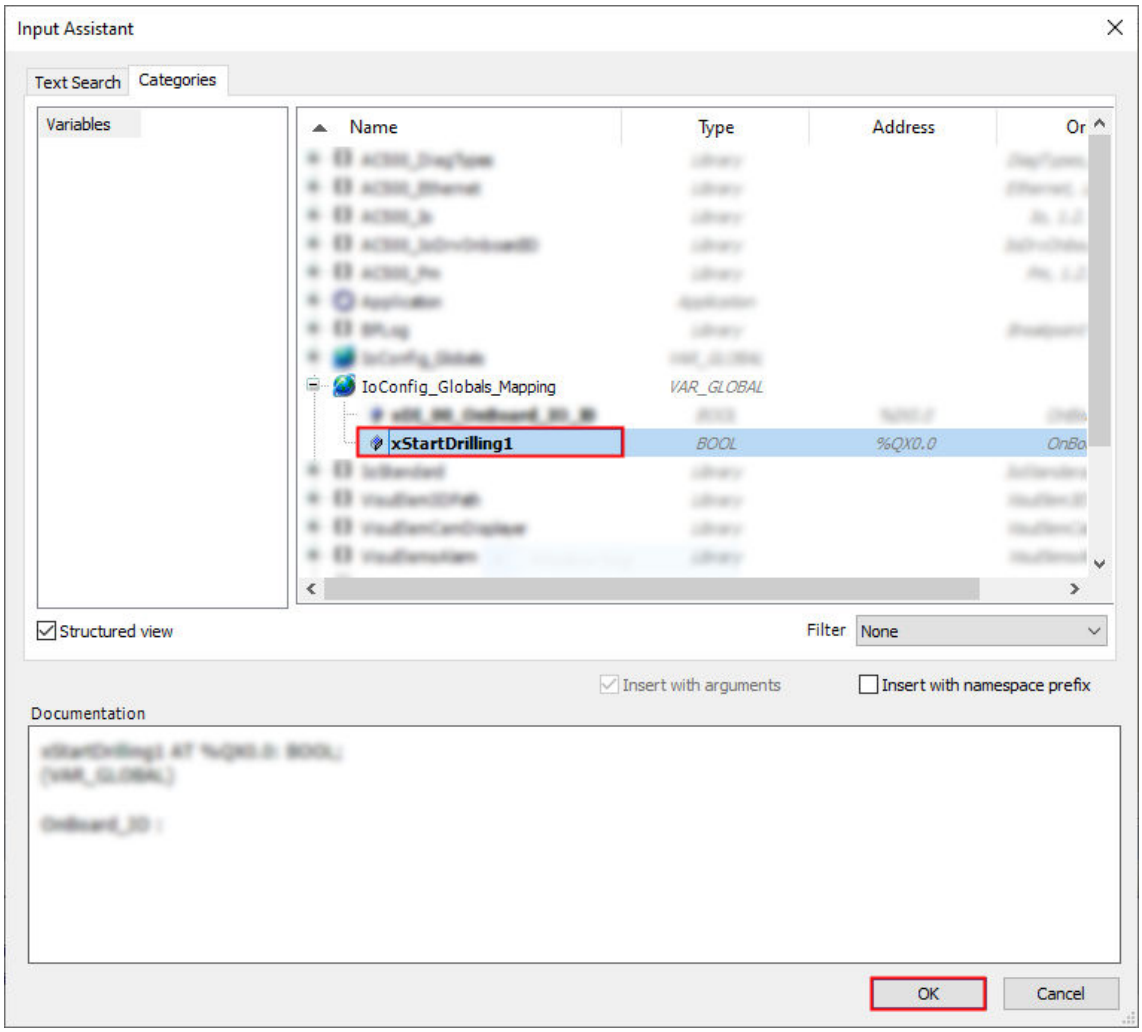
11. Drag and drop *"Network"* from the ToolBox to the down-arrow of network 1.

    ⇨ You added a network *"2"* below network 1.

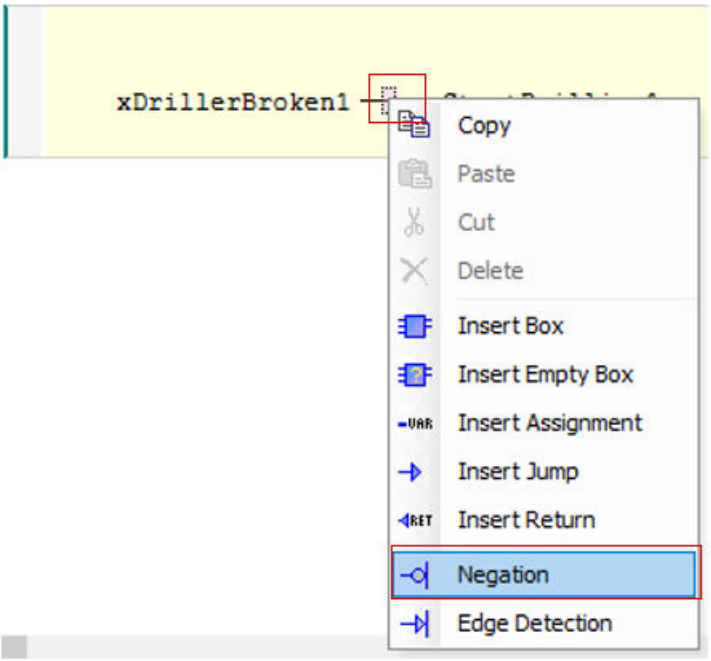
Add assignments and a Boolean NOT to the DO signals

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

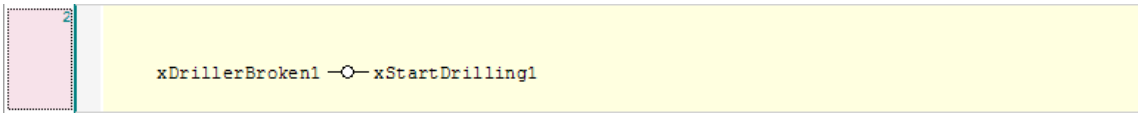
   ⇨ *"Input Assistant"* opens.



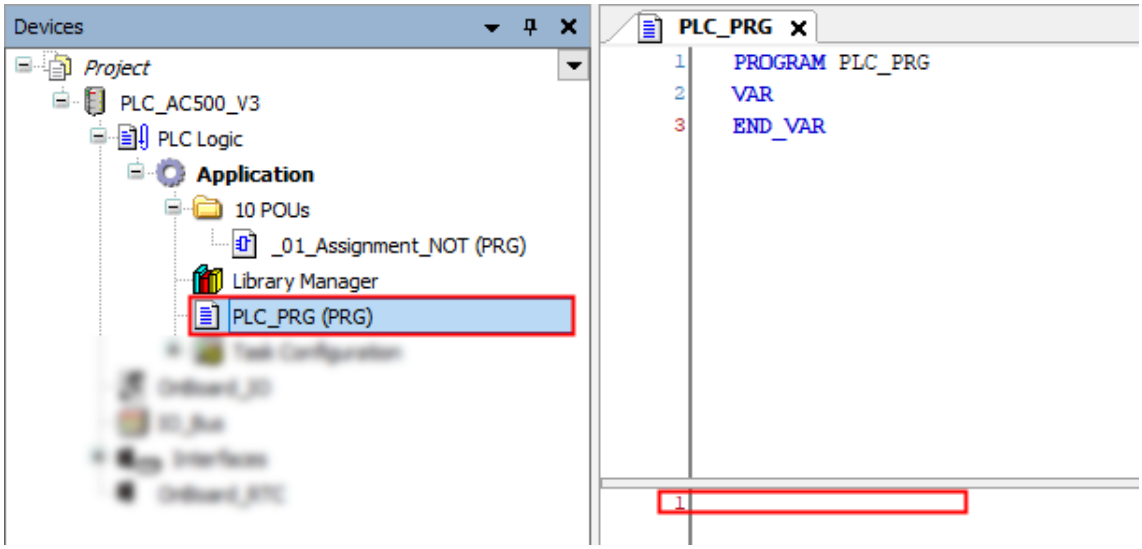
4. In the *"IoConfig_Globals_ Mapping"* variable list, select *"xStartDrilling1"*.
5. Select *"OK"* to close the dialog.



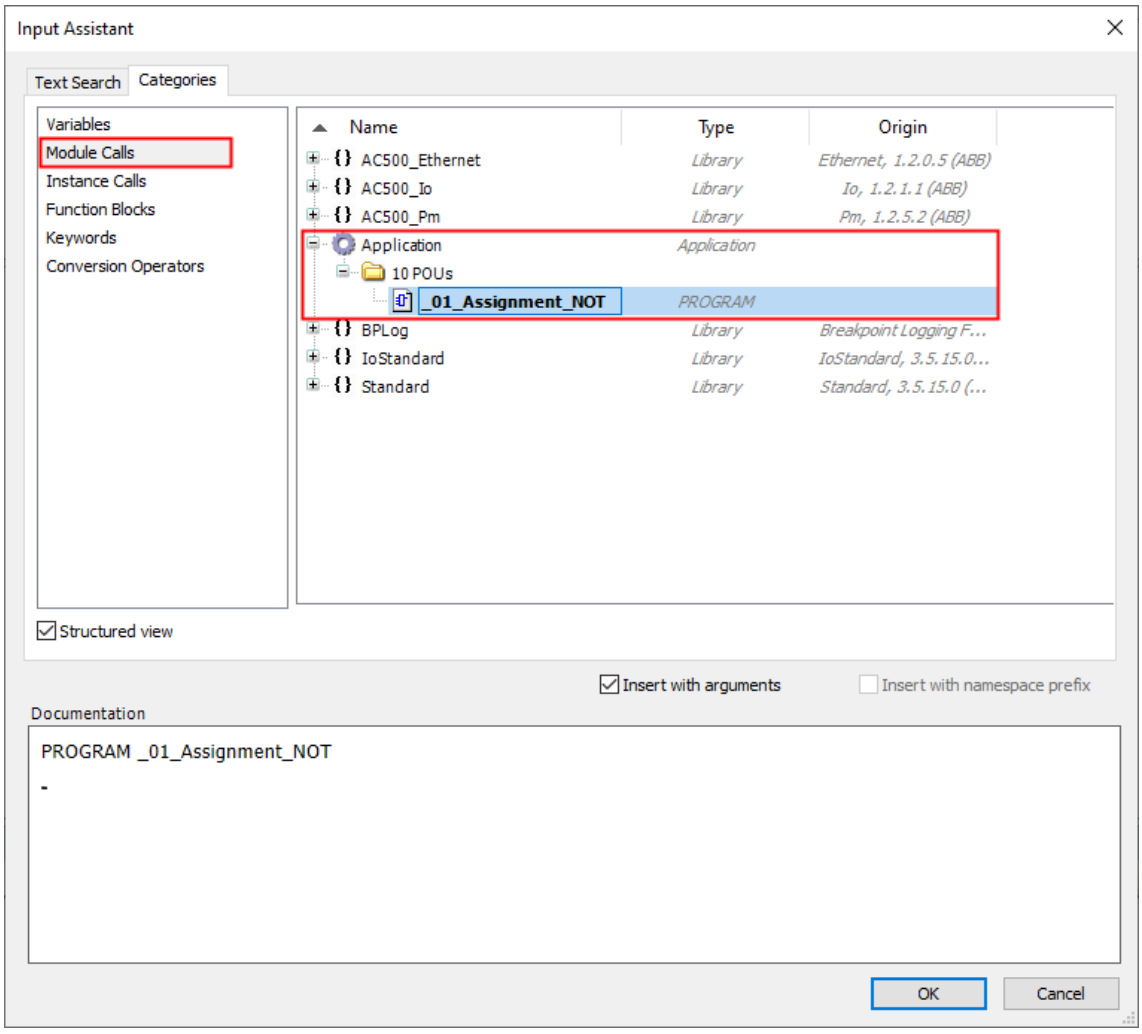
6. Right-click the center of assignment PIN.
7. Select *"Negation"* to add a negation to the assignment.



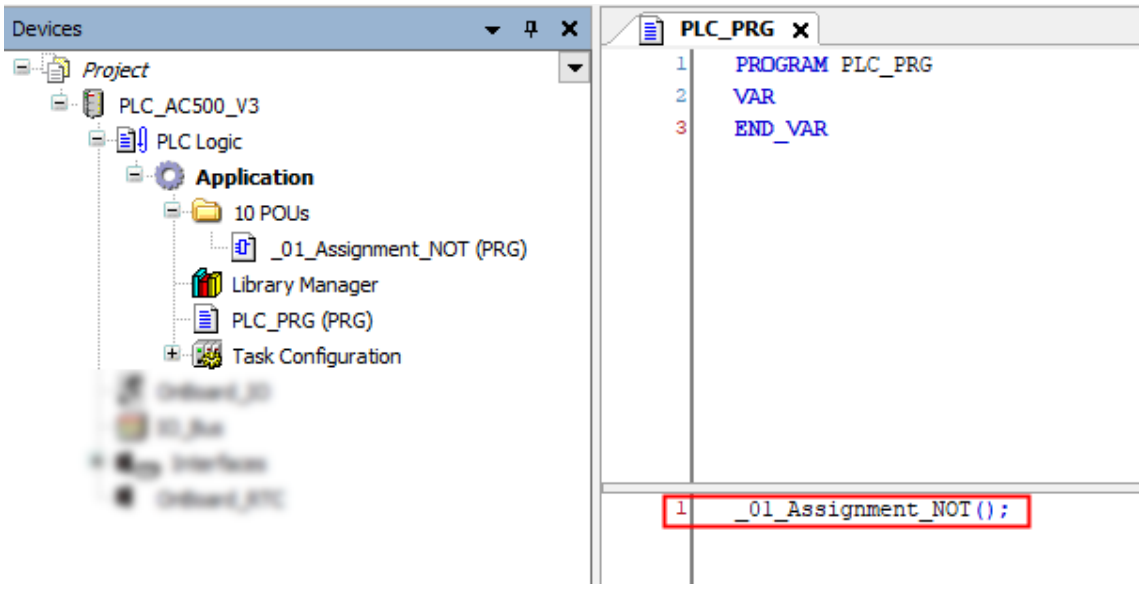
Call the POU in the PLC_PRG



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

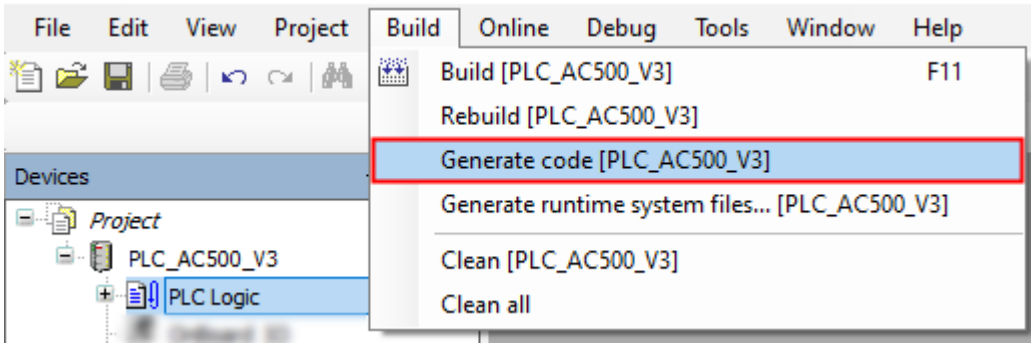


3. Select *“Module Calls”*.
4. Open *“Application”*.
5. Open *“10 POUs”* and select *“_01_Assignment_NOT”*.
6. Select *“OK”* to close the dialog.



Compile the project

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

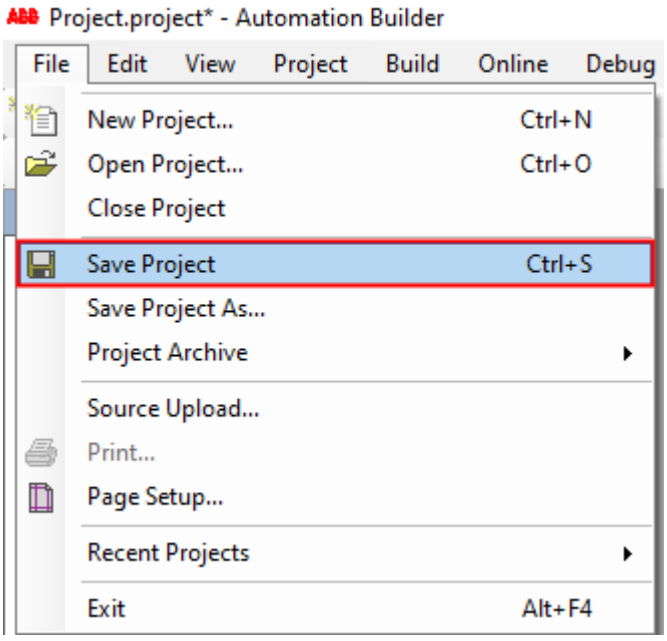


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



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

Alternatively, select the save icon in the tool bar.

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

Set-up the communication gateway

Set-up communication parameters

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

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

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

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

Subnet mask should be 255.255.255.0.

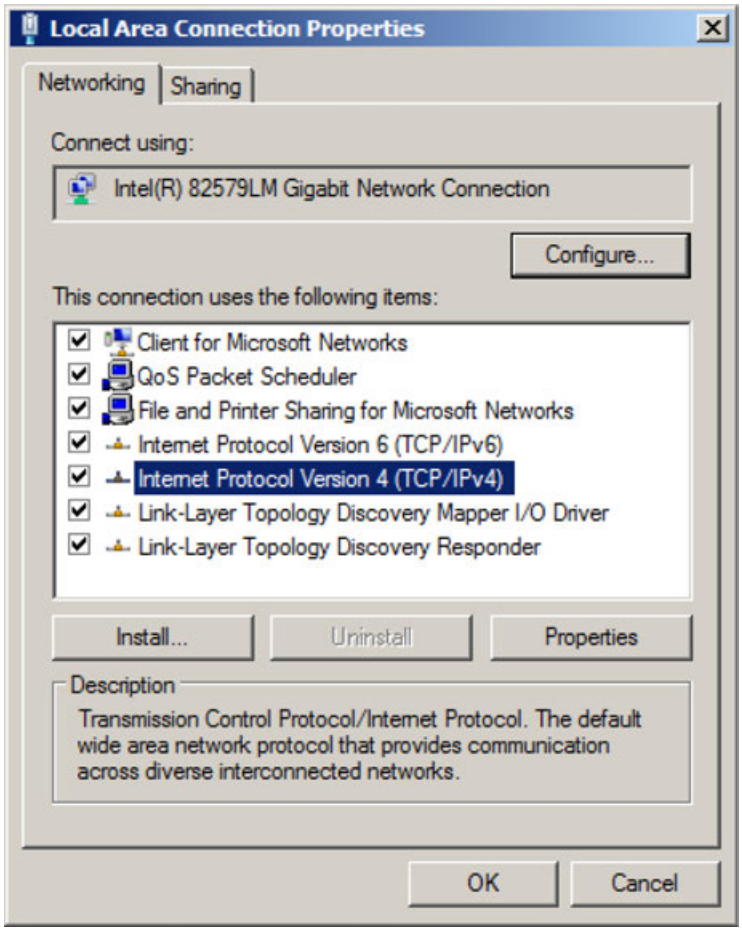
Change the IP address

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

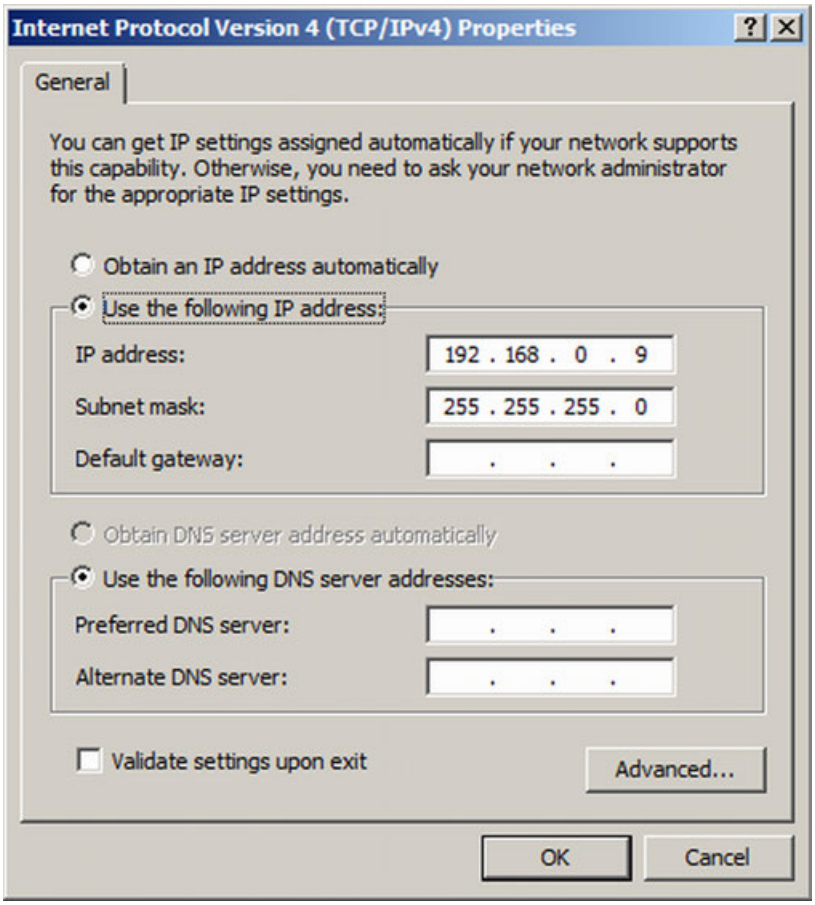
   ⇨

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

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



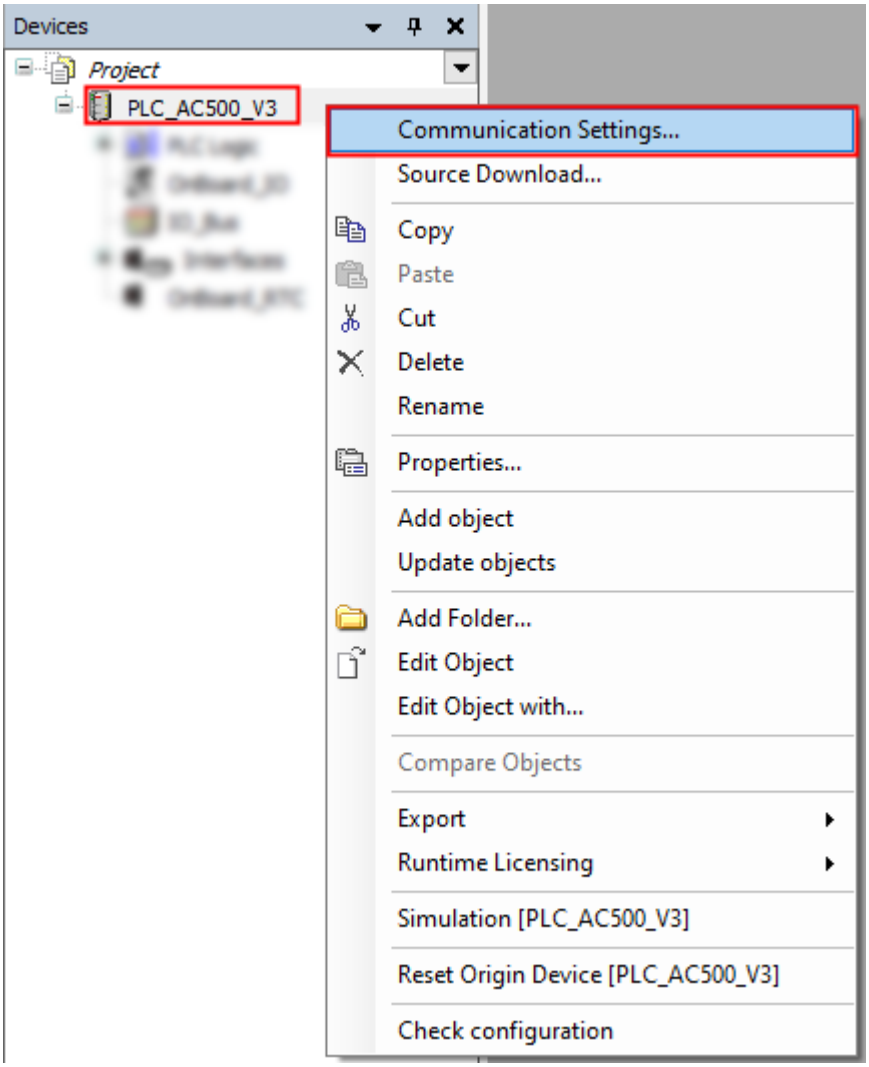
4. Double-click **Internet Protocol Version 4 (TCP/IPv4)**.



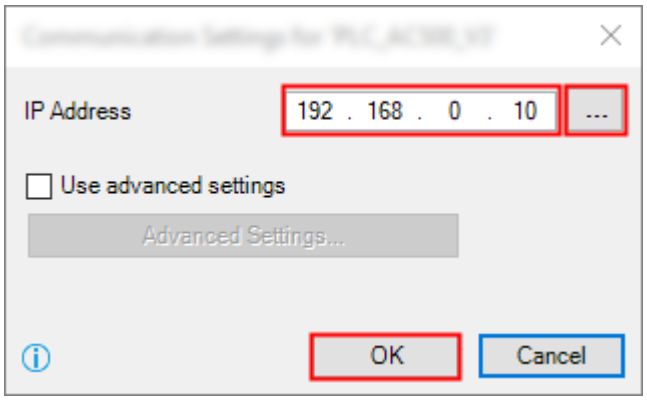
5. Enter your desired IP address and subnet mask.

Set-up the communication gateway

☑ CPU and PC are connected with an Ethernet cable.



1. In the Automation Builder device tree right-click *“PLC_AC500_V3”*.
2. Select *“Communication Settings”*.



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

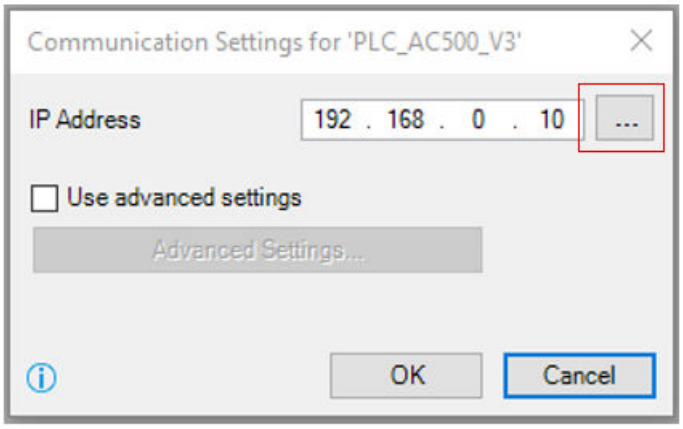
> *The standard (default) IP address of the port ETH1 is: 192.168.0.10*

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

Network scan

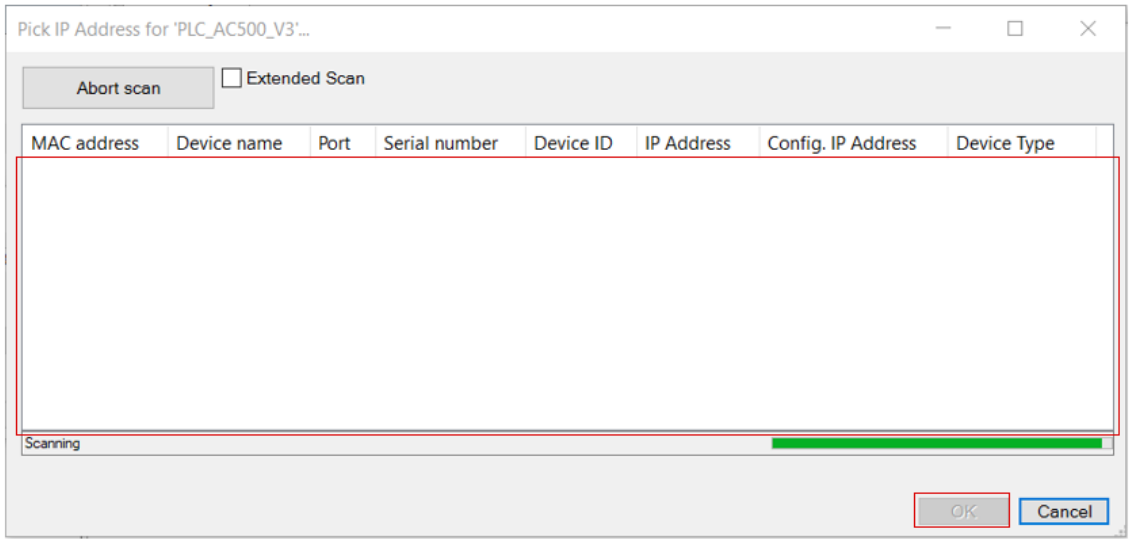
If you need to scan the network for the CPU or if you have multiple CPUs on the same network.

1. Right-click *"PLC_AC500_V3"* in the device tree.
2. Select *"Communication Settings"*.



3. Select *"..."*.

   ⇨ *"Pick IP Address for "PLC_AC500_V3""* opens.



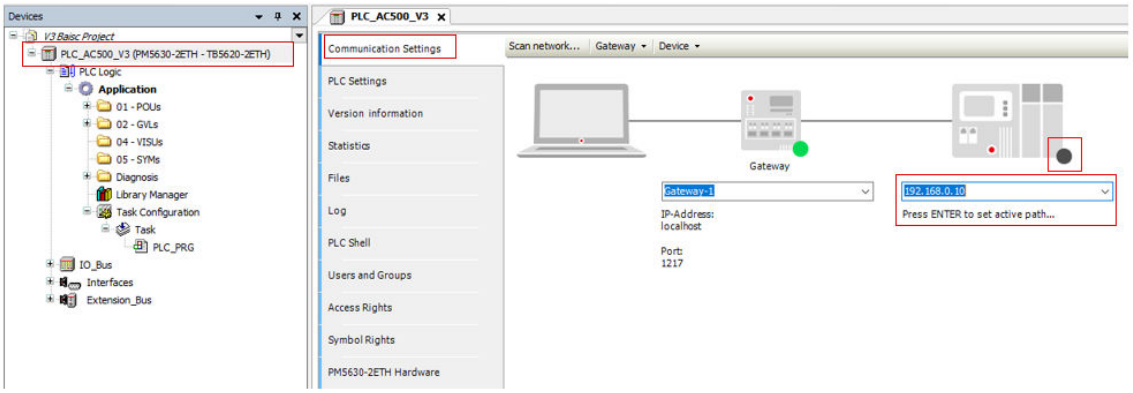
   The automatic scan runs.

   The results will appear in this field.

4. Select the CPU in the field and select *"OK"* to implement the needed communications gateway.

Check communication settings

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



1. Double-click *"PLC_AC500_V3"* in the device tree.

2. Select *“Communication Settings”*.

   ⇨ The selected IP address is shown.

3. If the IP address is not visible, enter the IP address manually.

4. To test the connection and/or to see the CPU information press *[Enter]* or click on the black dot next to the PLC figure.

AC500-eCo V3 firmware installation and update

The PLC firmware can be updated via Automation Builder.

> *This is also necessary for commissioning AC500-eCo V3 CPUs.*

A very new CPU has no pre-installed firmware. To guarantee the authenticity of delivered AC500-eCo firmware, V3 CPUs are delivered with a boot loader only. You need to download a valid firmware to the CPU. After download, the functionality of the CPU is given.

☑ An Automation Builder project with an AC500-eCo V3 CPU is open.

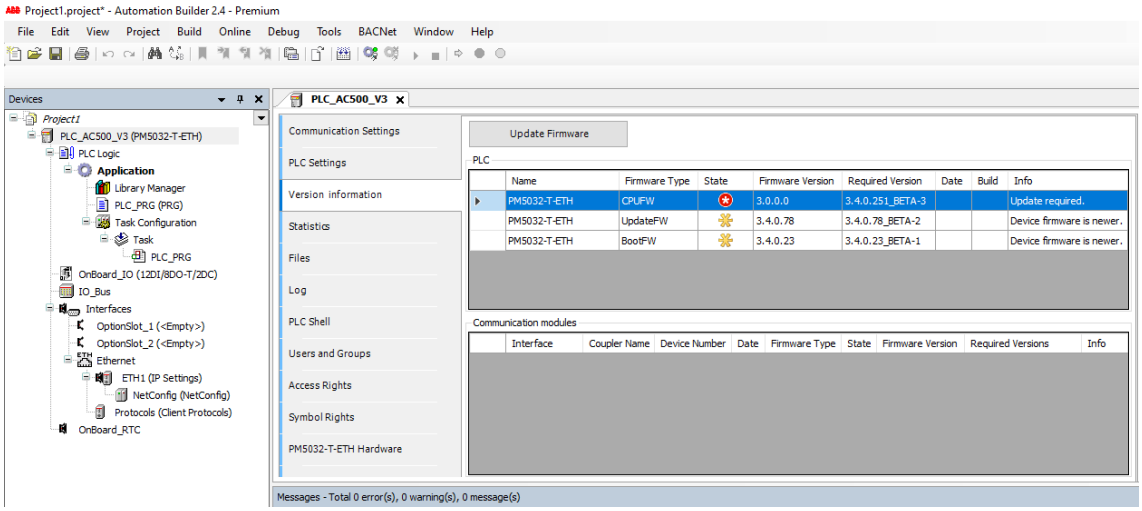
☑ CPU is in "stop" mode without firmware.

☑ The power LED is ON.

☑ For new modules: IP address is set. (The default IP address is 192.168.0.10)

1. Double-click CPU *“PLC_AC500_V3”*.

2. Select *“Version information”*.



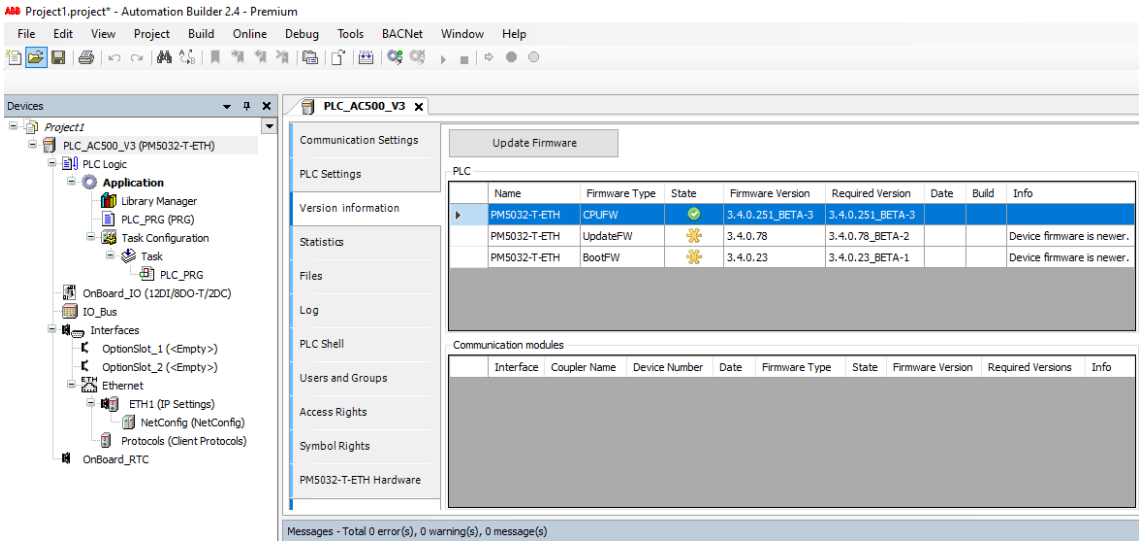
3. Select *[Update Firmware]*.

   ⇨ While the update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

4. Wait for the PLC to finish the update.

> **NOTICE!**
> Do not disconnect the power supply during the update process! The PLC could be damaged.

5. If necessary, refresh the version information by switching to another tab and back.
   ⇨ Successful firmware update:



Behavior of LEDs during firmware update

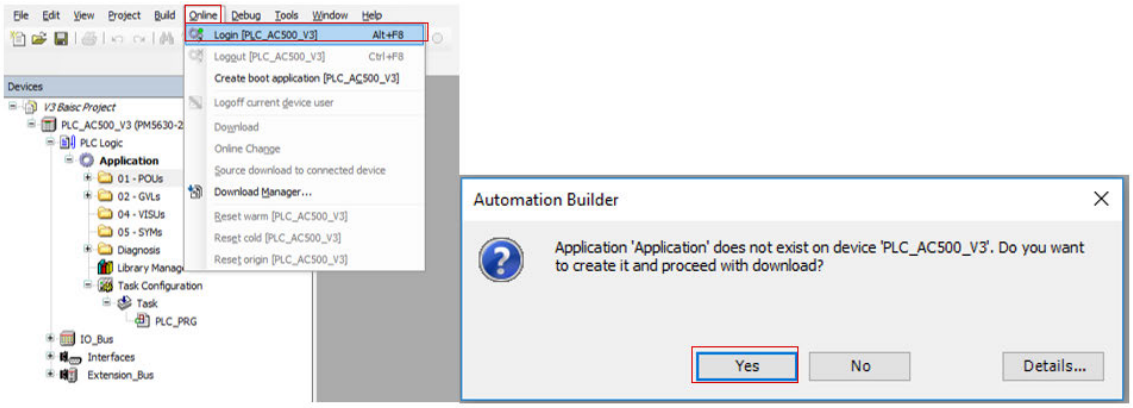
- CPU without firmware, only the power LED is on.
- While the firmware update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

LED	LED flashes	Status
RUN and ERR	Toggling	Update pending
RUN	Flashing slow	Done successful
ERR	Flashing slow	Done failed

- CPU with installed firmware, only the power LED is on.
- If the CPU is running, then the RUN LED is on.
- If the CPU is in STOP mode, the RUN LED is off.

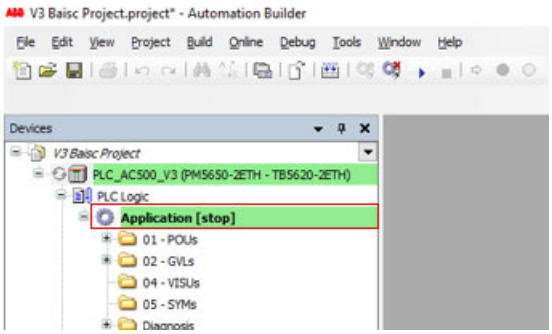
Log-in to CPU and download the program

Logging-in to the CPU will load the project into the AC500 V3 CPU. The first log-in will also load the hardware set-up.

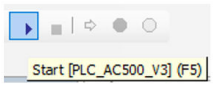


1. In the Automation Builder menu select *“Online ➔ Login [PLC_AC500_V3]”*.
   ⇨ A pop-up will appear.

2. Select *“Yes”* to download the application to the AC500V3 CPU.



⇨ PLC is in "stop" mode.



3. Start the PLC ↳ *Chapter 1.2.18.2.2.8.1 “Start the program execution” on page 149*.

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

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

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

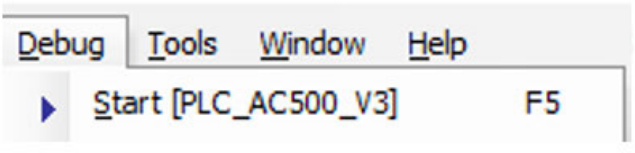
Test the program

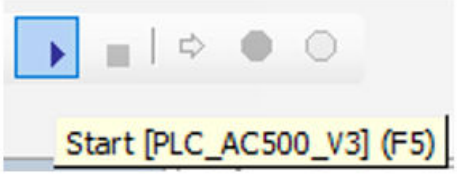
Start the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in "stop" mode.





▷ Select menu *“Debug ➔ Start [PLC_AC500_V3]”*.

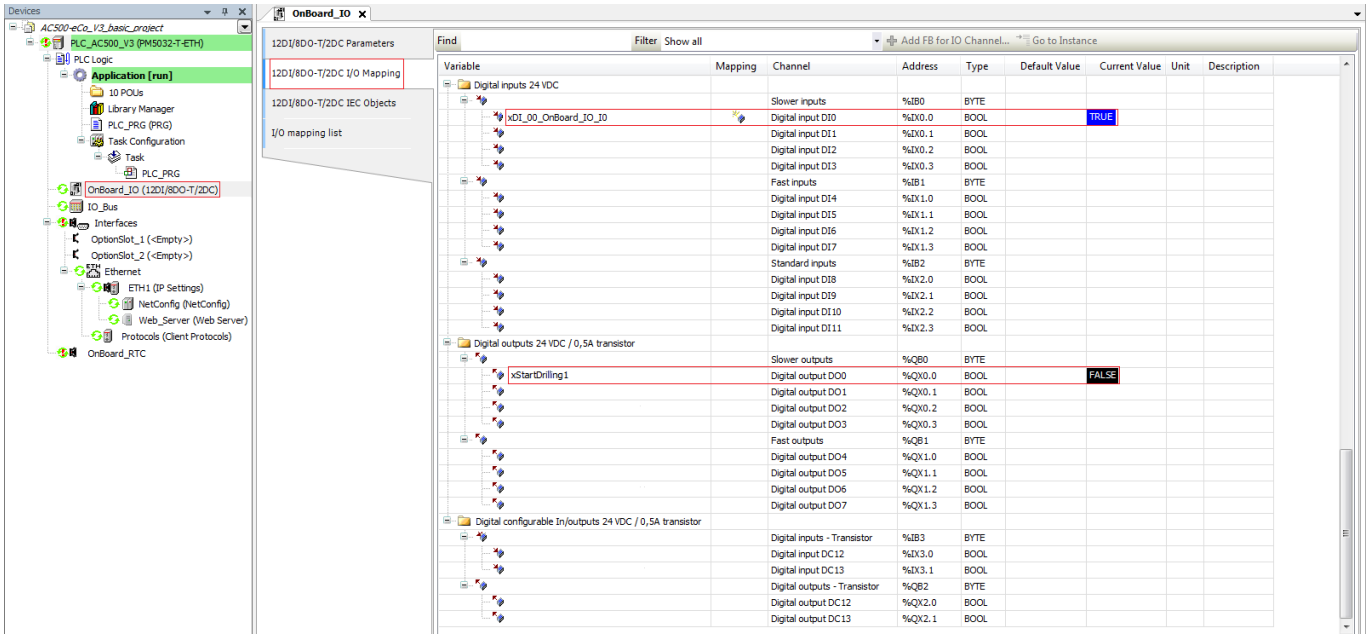
Alternatively, select the "start" icon in the tool bar.

Alternatively, press *[F5]*.

Test the function

▷ Operate the switch I1 and observe:

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

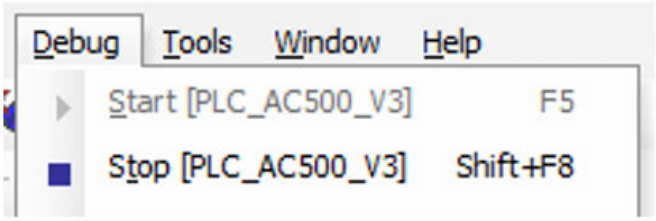


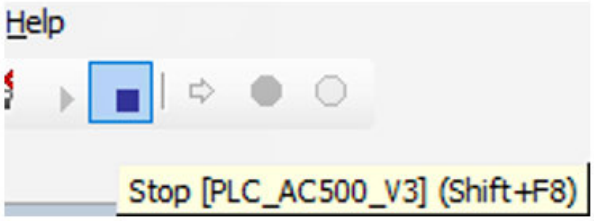
Stop the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in RUN mode.





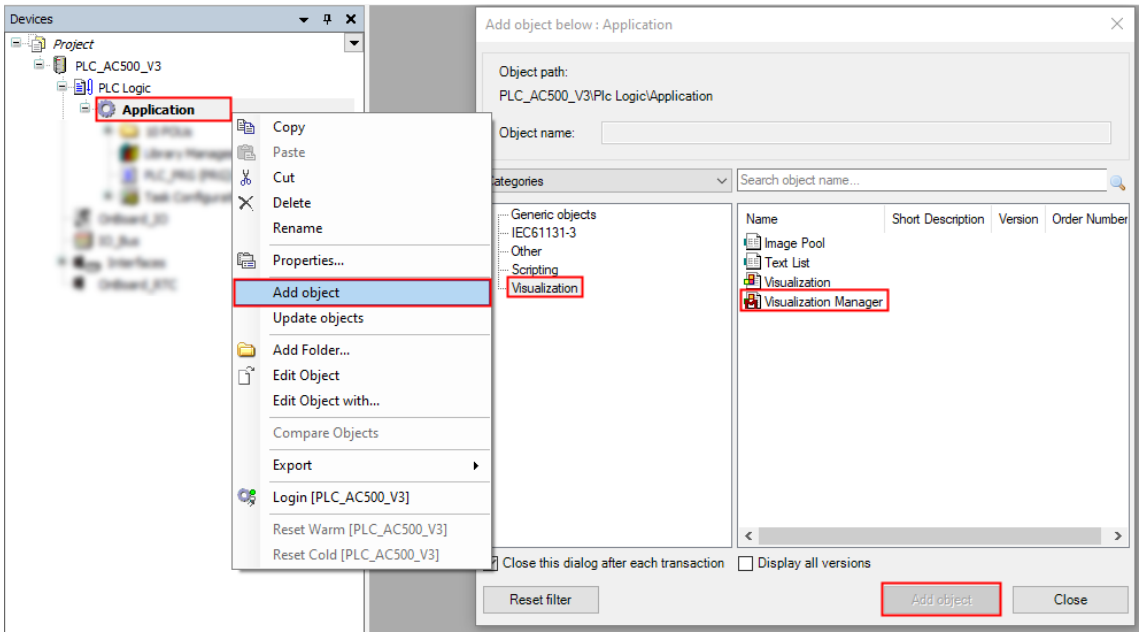
▷ Select menu *“Debug ➔ Stop [PLC_AC500_V3]”*

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

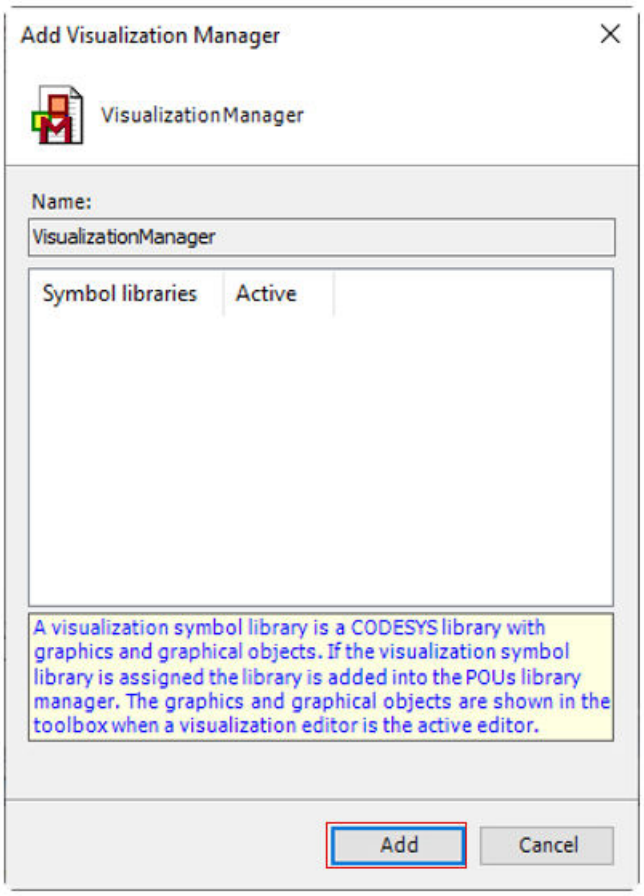
Alternatively, press *[Shift]* + *[F8]*.

Set-up visualization

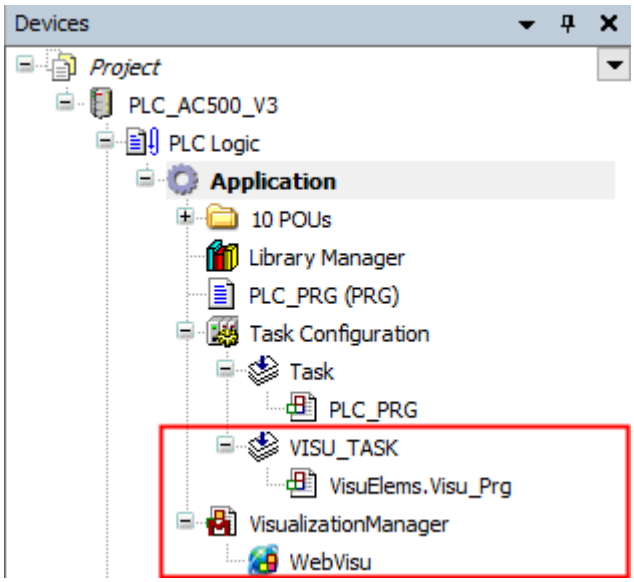
Add the VisualizationManager



1. Right-click *“Application”* in the device tree.
2. Select *“Add object”*.
3. Select *“VisualizationManager”*.
4. Select *“Add object”* to add the VisualizationManager to the project.
   ⇨ Dialog *“Add Visualization Manager”* opens.

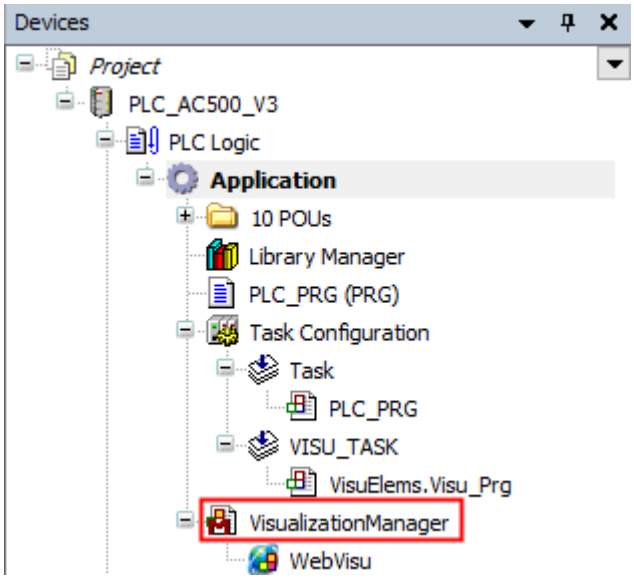


5. Select *“Add”*.

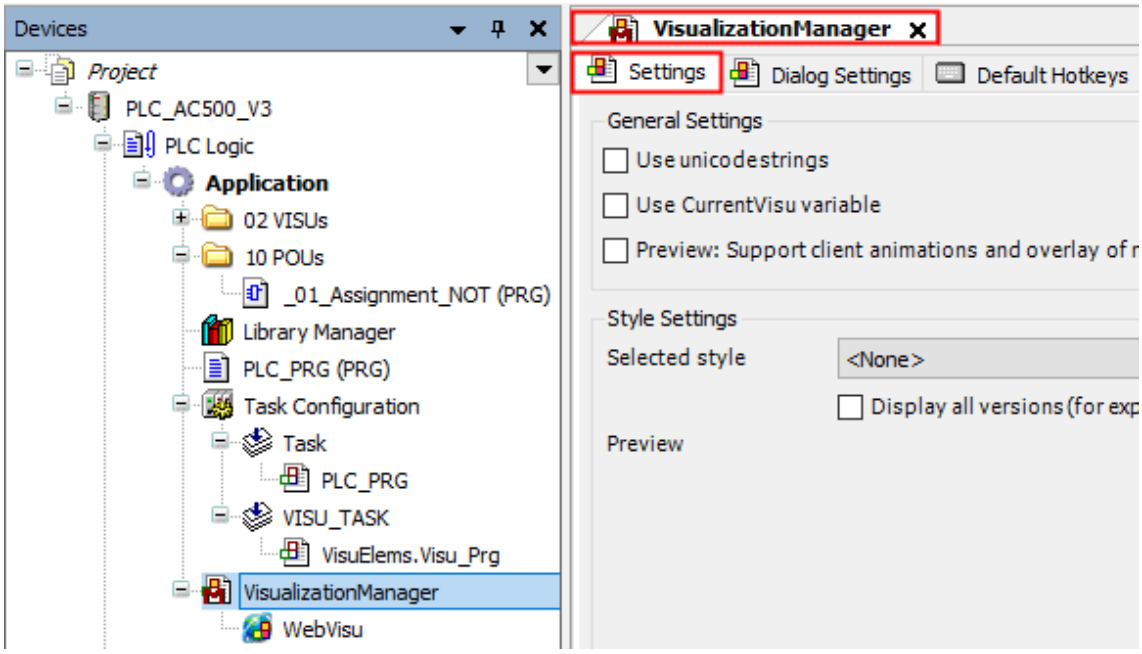


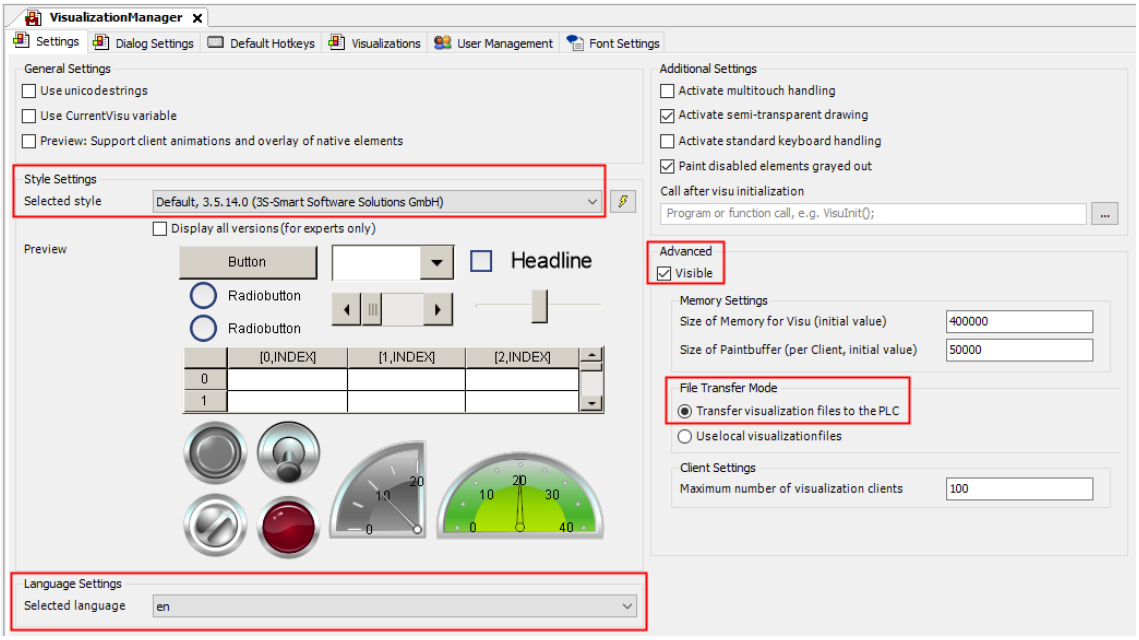
⇨ You added the objects *“VisualizationManager”* and *“VISU-TASK”* to the device tree.

Set-up the VisualizationManager



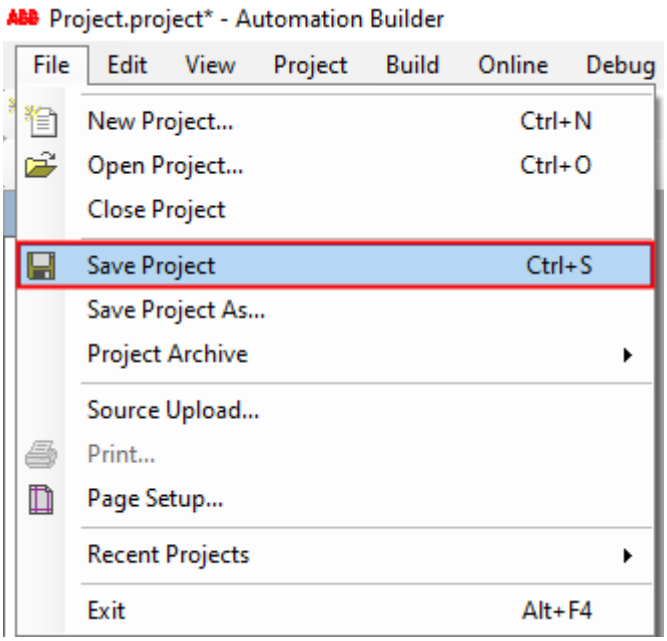
1. Double-click VisualizationManager in the device tree.
   ⇨ A tab opens in the editor view.





2. Select *“Settings”*.
3. Open the drop-down menu *“Selected style”*.
4. Select *“Default, x.x.x”* (exemplary).
5. Open the drop-down menu *“Selected language”*.
6. Select *“en”* for English language in the visualization.
7. Enable *“Visible”* for advanced settings.
8. Keep the file transfer to enable the visualization on the PLC (mandatory for web server function ↳ *Chapter 1.2.18.2.2.11 “Enable web visualization” on page 163*).

Save the project



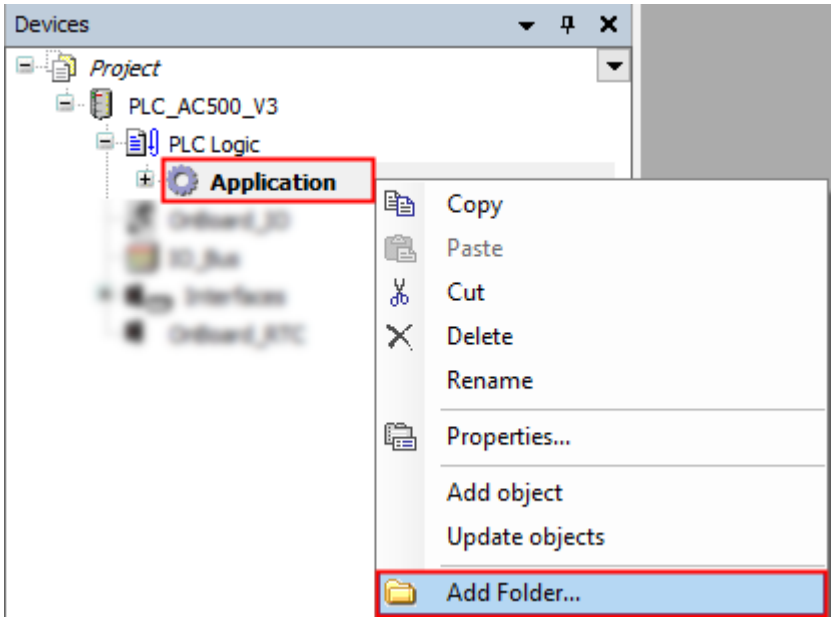
▷ Select menu *“File ➔ Save Project”*.

Alternatively, select the save icon in the tool bar.

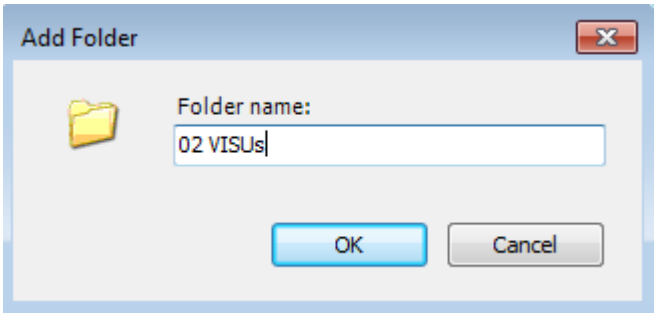
Alternatively, press *[Ctrl] + [S]*.

Create visualization

Add a folder for visualization screens

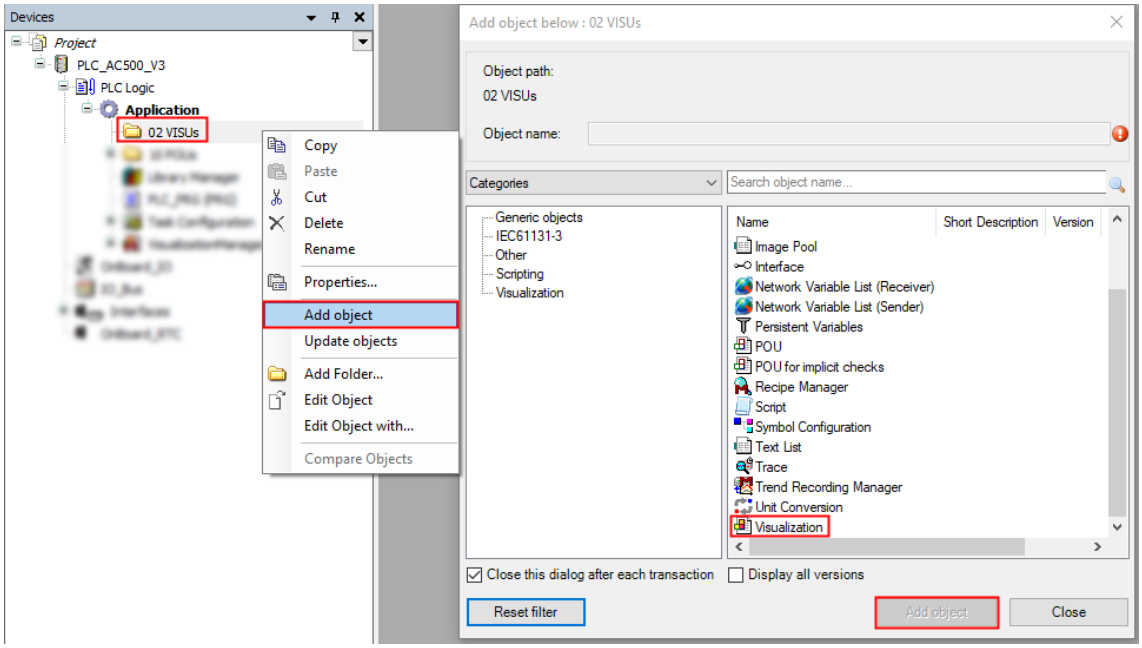


1. Right-click *“Application”* in the device tree.
2. Select *“Add Folder”*.

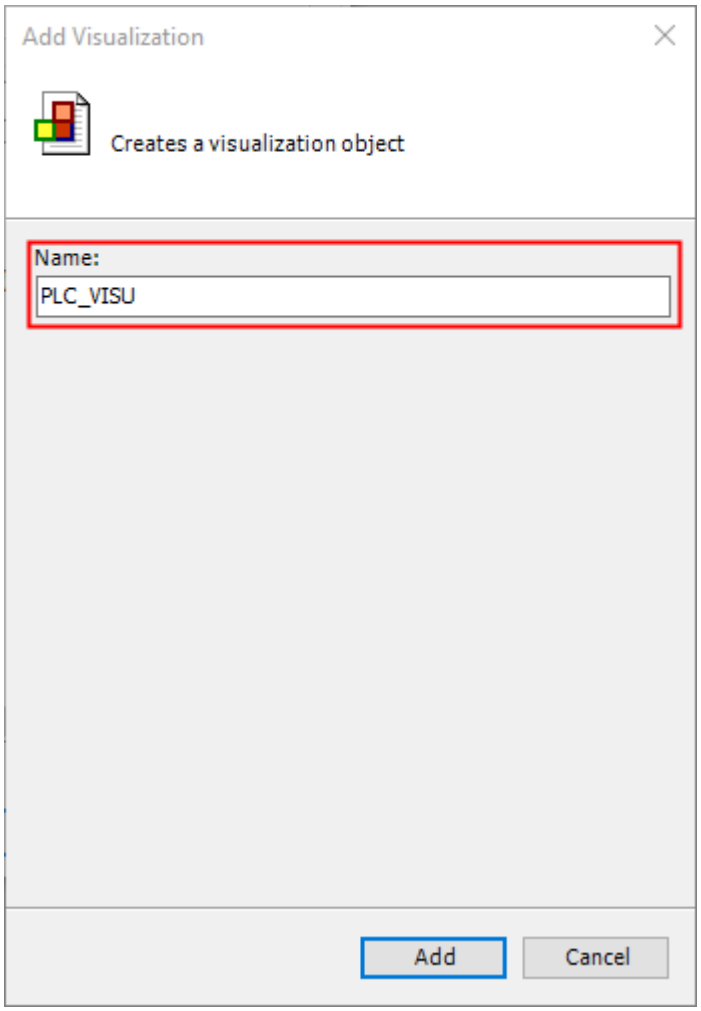


3. Type in "02 VISUs".
4. Select *“OK”* to add the folder.

Add a screen for "_01_Assignment_NOT" POU



1. Right-click *“02 VISUs”*.
2. Select *“Add object”*.
3. Select object *“Visualization”*.
4. Select *[OK]*.



5. Type in "PLC_VISU".
6. Select *“Add”*.
   ⇨ A tab opens in the editor view.

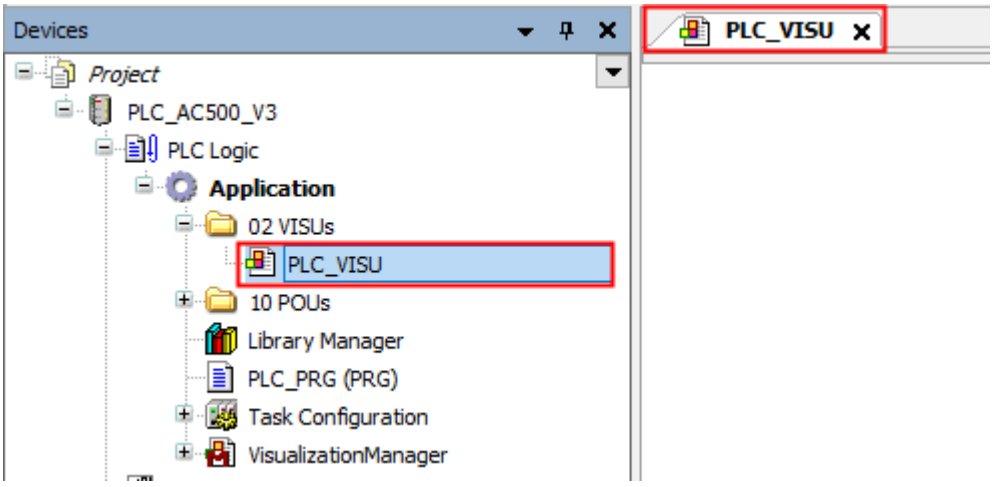


Fig. 10: PLC_VISU_tab

The name "PLC_VISU" has been chosen, because it is the default name for a home screen in a web visualization.

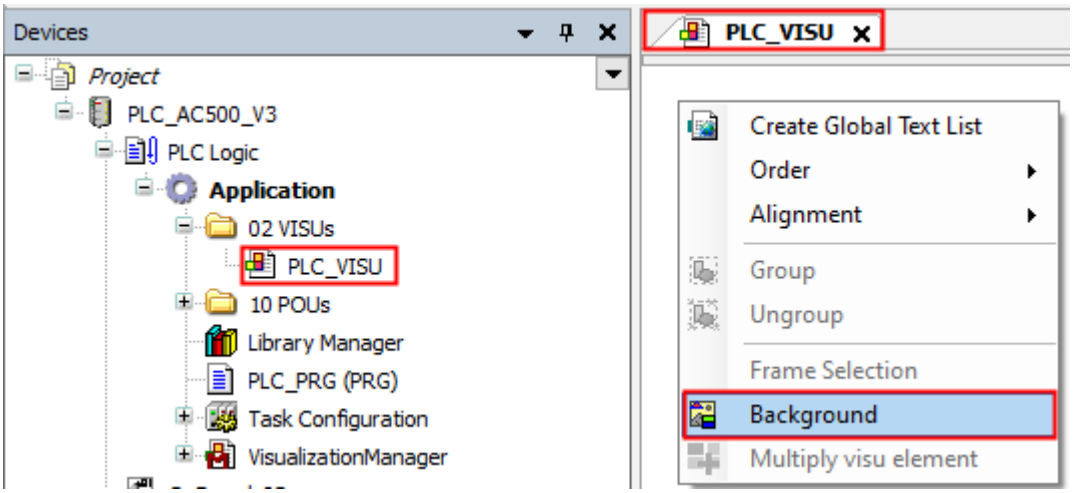
If you have more than one visualization object in your project, it will be useful to choose another name, e.g. "_01_Assignment_NOT_v". And to choose "PLC_VISU" as a home screen to access all available visualization screens.

The name of a visualization object can be modified afterwards.

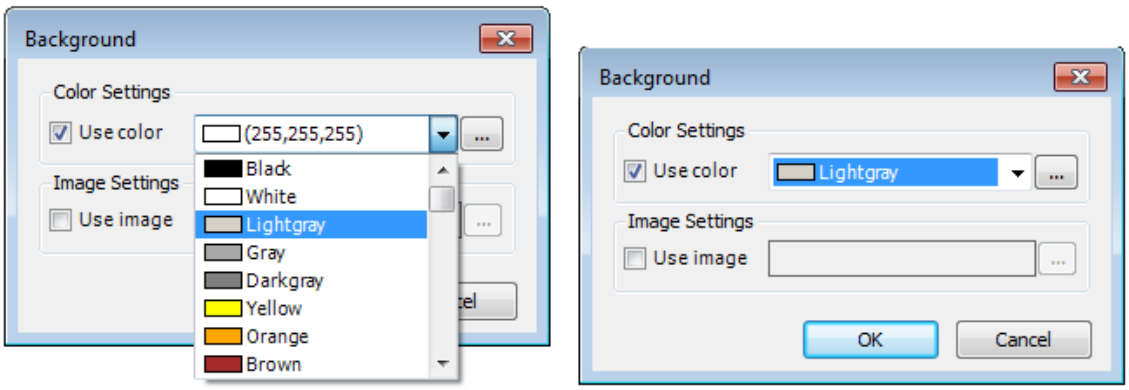
Creating and configuring of visualization

Change background color

1. Double-click *“PLC_VISU”* in the device tree.
   ⇨ A tab opens in the editor view.



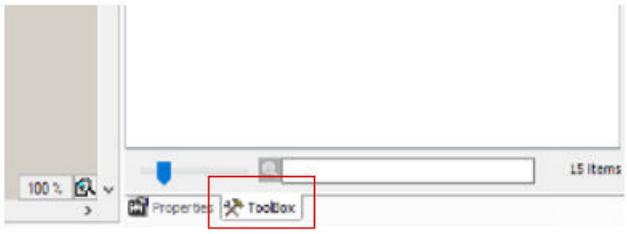
2. Right-click anywhere on the "PLC_VISU" editor page.
3. Select *“Background”*.



4. Enable the check box *“Use Color”*.
   ⇨ This enables the drop-down menu.
5. Select a color, e.g., *“Lightgray”*.
6. Select *[OK]* to add the color to "PLC_VISU".

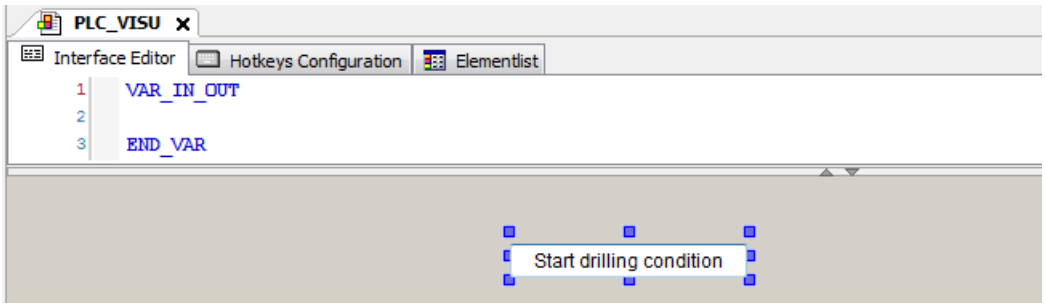
Add a screen title

1. Double-click on *“PLC_VISU”* in the device tree.



2. Select *“ToolBox”*.

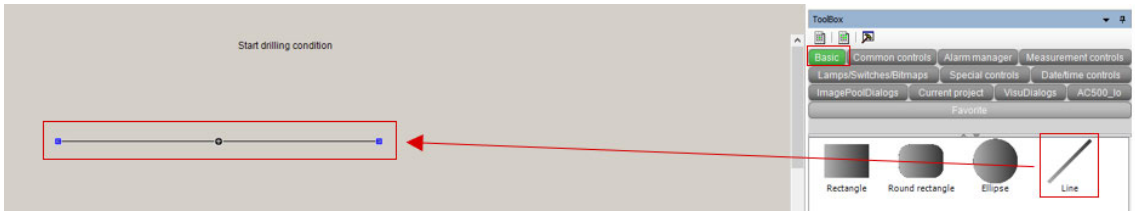
3. Select *“Common controls”*.
4. Drag and drop *“Label”* to the page.



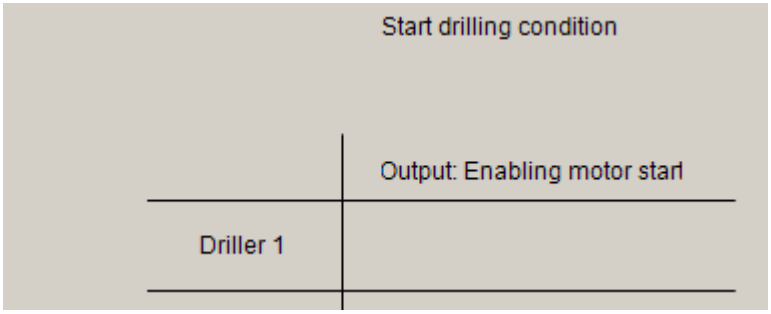
5. Type in "Start drilling condition".

Further lines and labels

1. Double-click on *“PLC_VISU”* in the device tree.



2. Select *“ToolBox”*.
3. Select *“Basic”*.
4. Drag and drop the line. Then drag the line to the needed length.

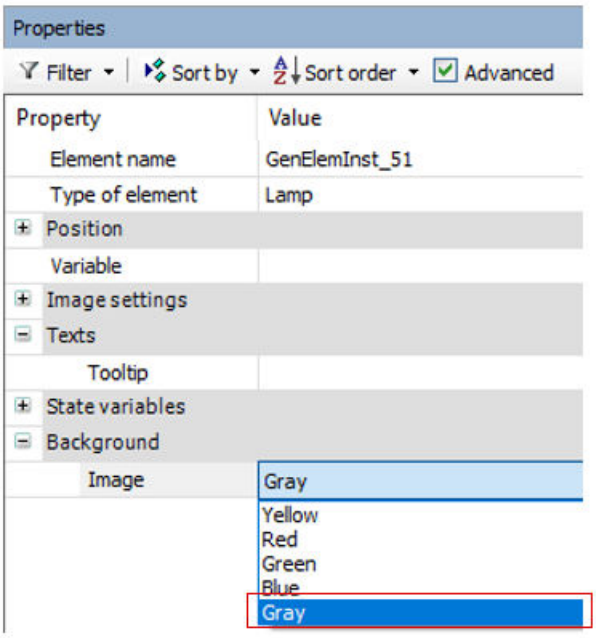


5. Follow the same procedure to create the other shapes and labels.

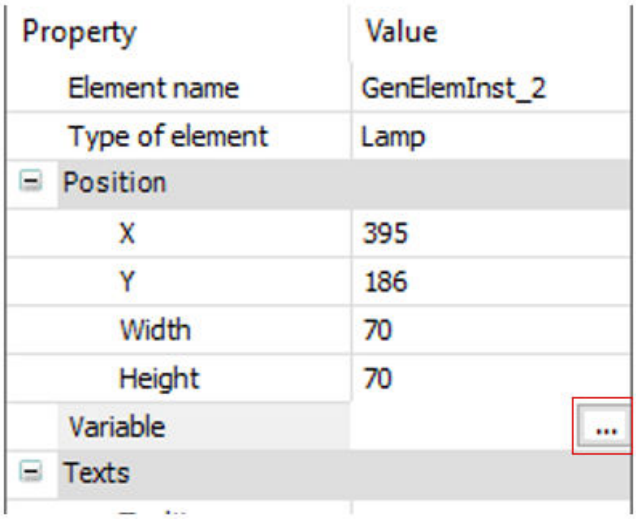
Lamp element for signal indication

1. Double-click on *“PLC_VISU”* in the device tree.

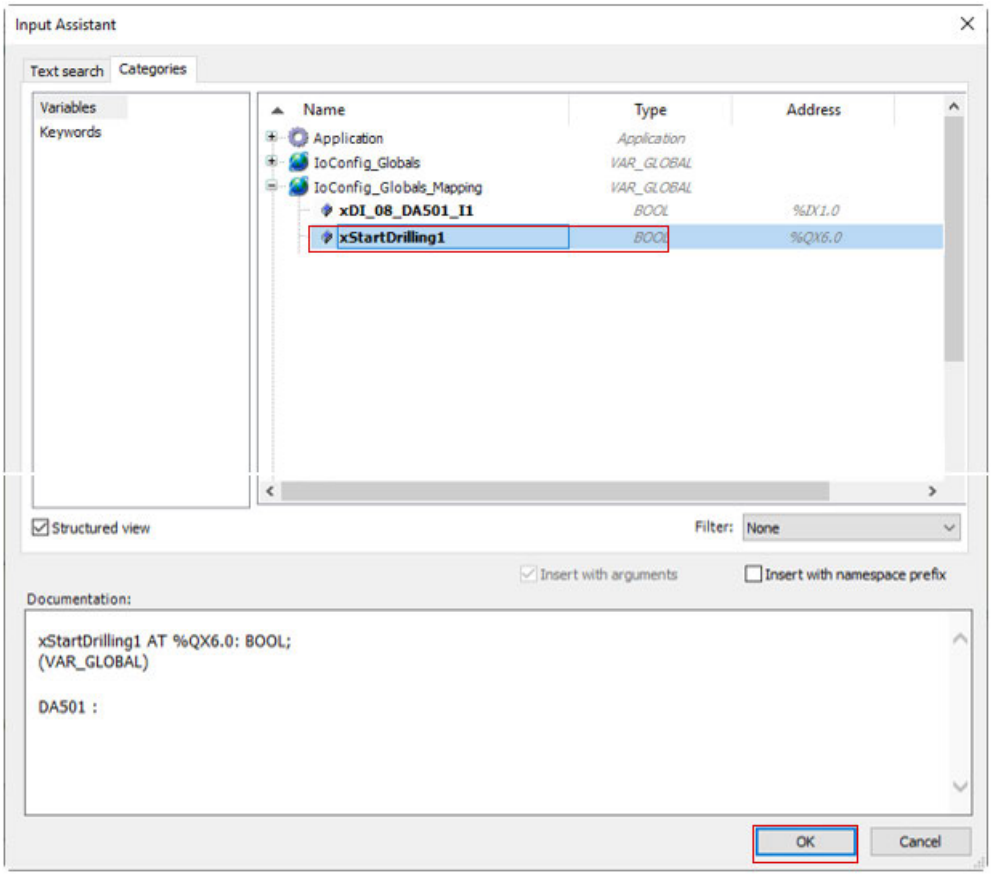
2. Select *“ToolBox”*.
3. Select *“Lamps/Switches/Bitmaps”*.
4. Drag and drop *“Lamp”* to the screen.
5. Adapt the size, if required.



6. Under *“Image”*, select *“Gray”*.



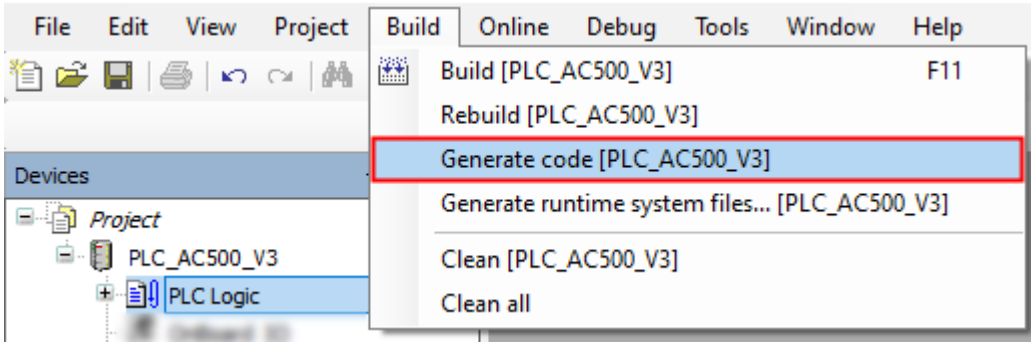
7. Double-click on *"Variable"* and select *"..."* to select a variable from the list.



8. Under *"IoConfig_Globals_Mapping"*, select *"xStartDrilling1"*.
9. Select *[OK]*.

Compile the project

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

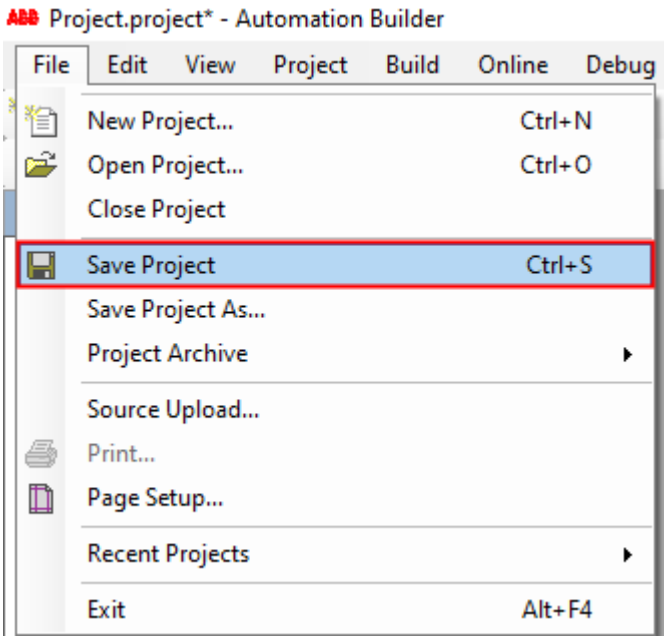


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



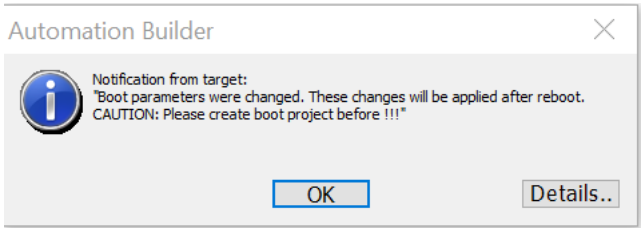
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

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

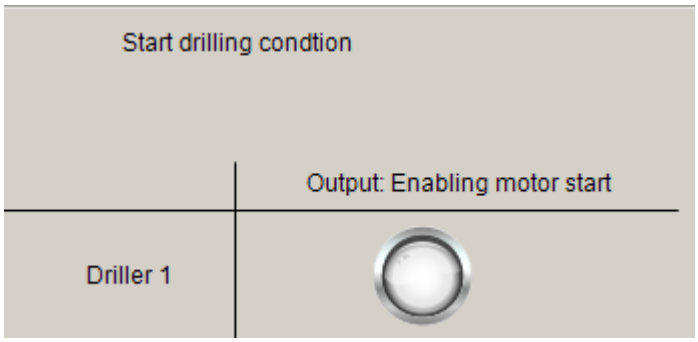
Loading the project to the CPU

1. Download the project to the CPU ↬ *as described in Chapter 1.2.18.2.2.7 , on page 148*.
2. Check the notification window at the end of the download. In case of message "Boot parameters were changed. These changes will be applied after reboot", a reboot of the CPU is required after creation of the boot project.



Test the program

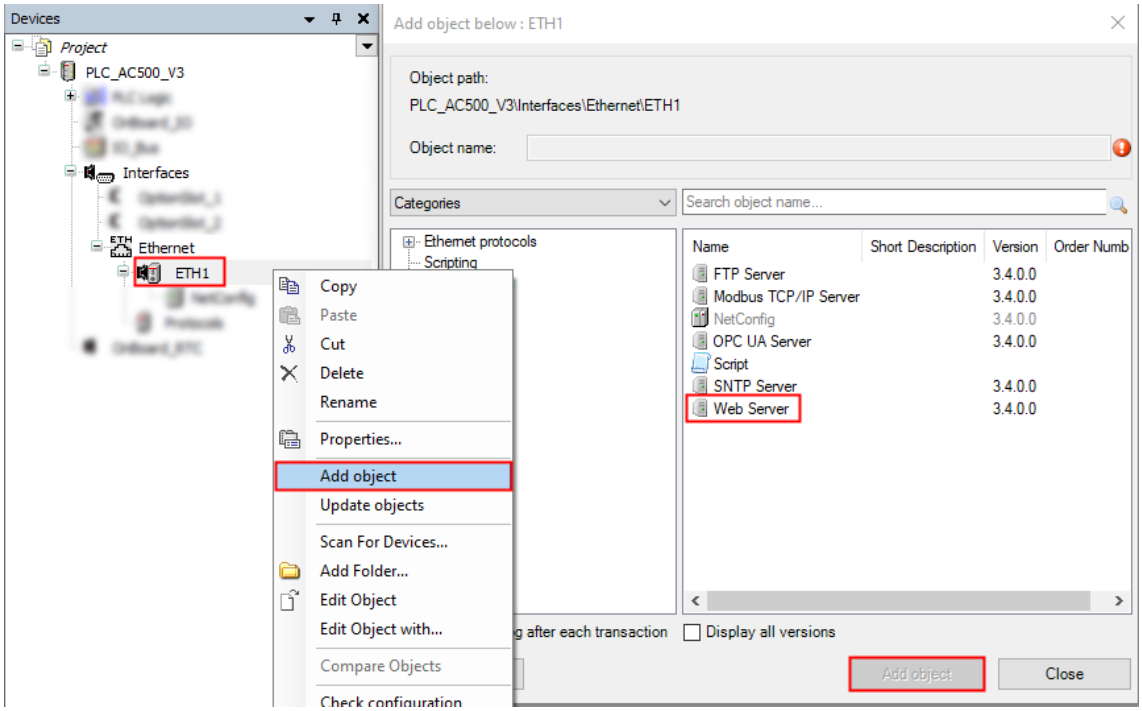
▷ Operate the switches and observe the visualization screen.



Enable web visualization

Add a web server object to the device tree

Ethernet ports can be configured for web server protocol. This description deals with ETH1 configuration for the web server

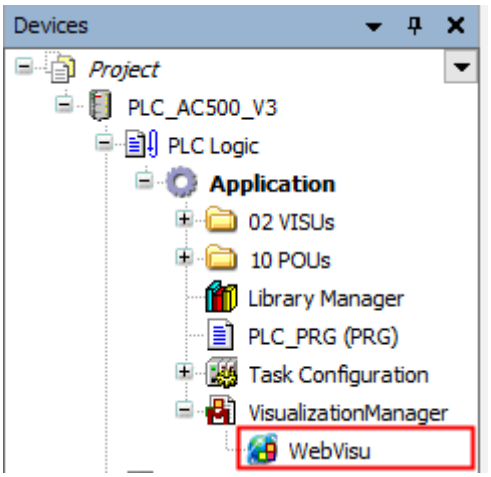


1. Right-click *“ETH1”* in the device tree.
2. Select *“Add object”*.
3. Select *“Web Server”*.

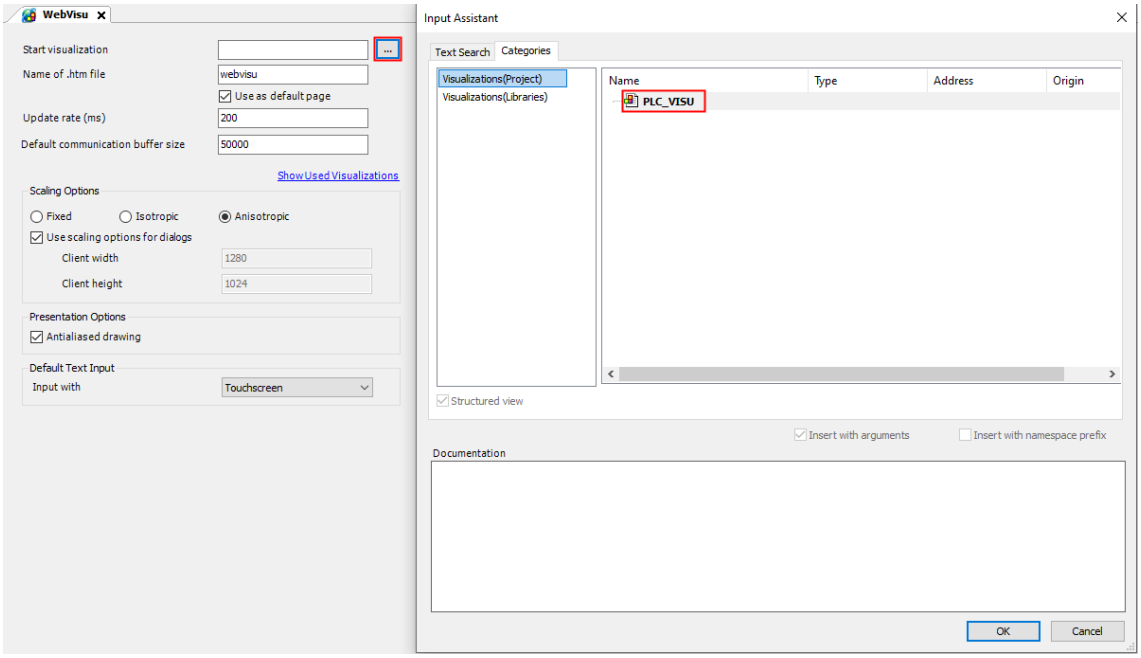
4. Select *“Add object”*.

   ⇨ You added and activated a web server on Ethernet port 1 on the AC500 V3 CPU.

Set-up the web server



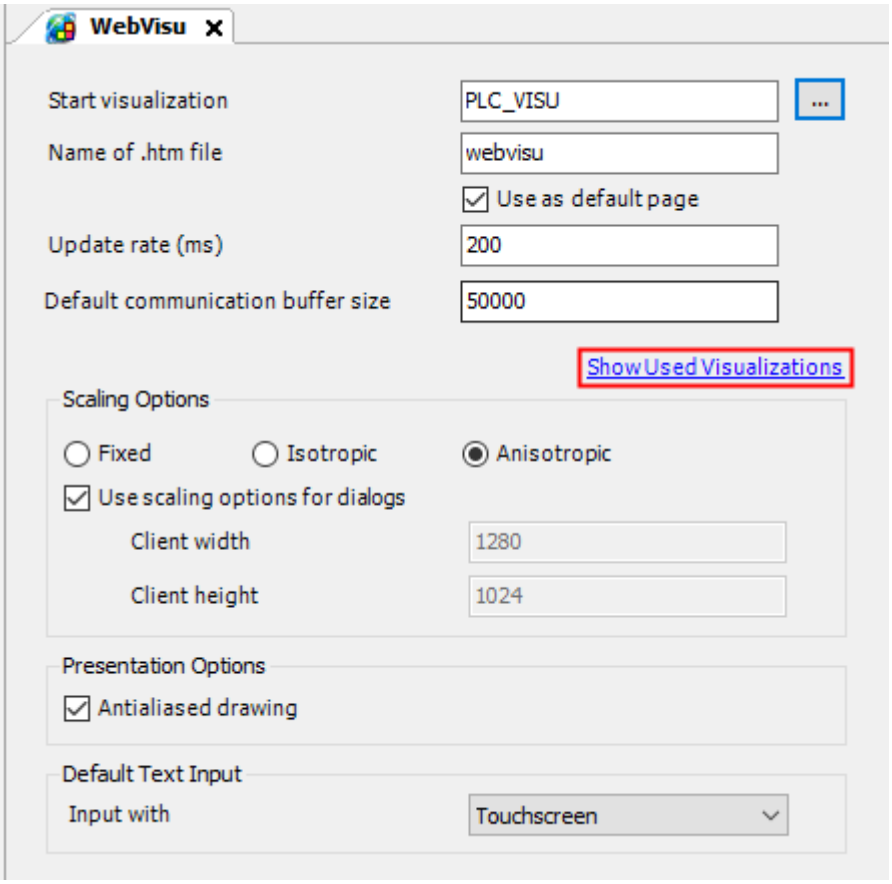
1. Double-click *“WebVisu”* in the device tree.



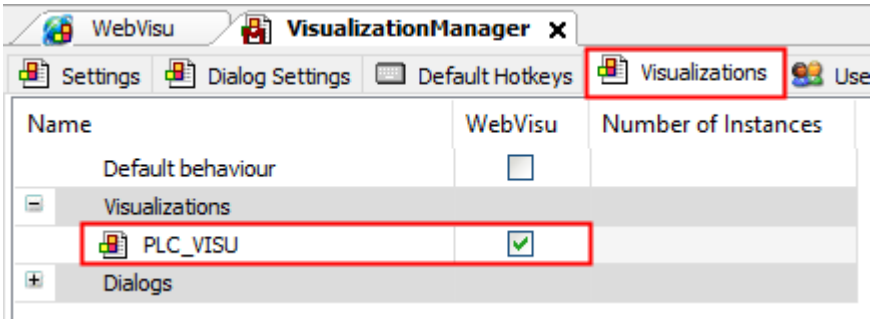
2. Under *“Start Visualization”*, select *“...”*.

   ⇨ A list opens.

3. Select the *“PLC_VISU”* screen from the list.
4. Keep all further settings with default values.



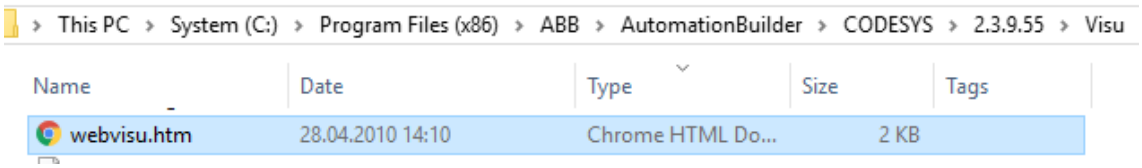
5. Select the link *"Show used visualizations"*.



⇨ The VisualizationManager editor and there the tab *"Visualizations"* opens. All screens and dialog elements created in the project are visible.

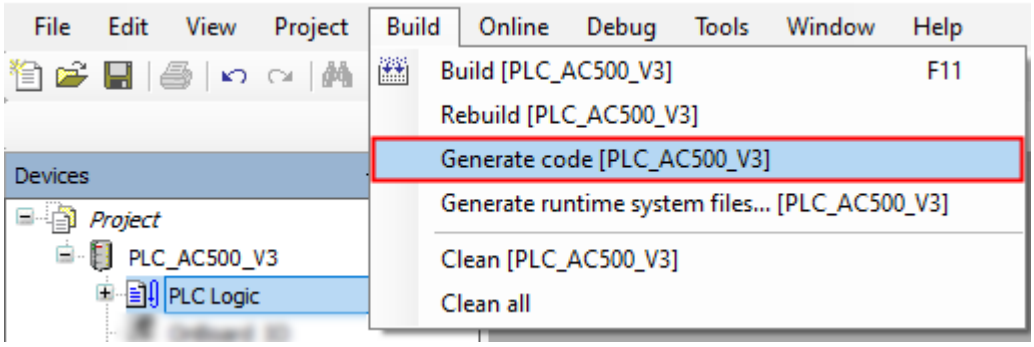
Here, you can select which screens are enabled or disabled for web visualization.

If you want to select another screen as a start visualization, you must modify the adequate parameter in the webvisu.htm file: `<param name="STARTVISU" value="PLC_VISU">`



Compile the project

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

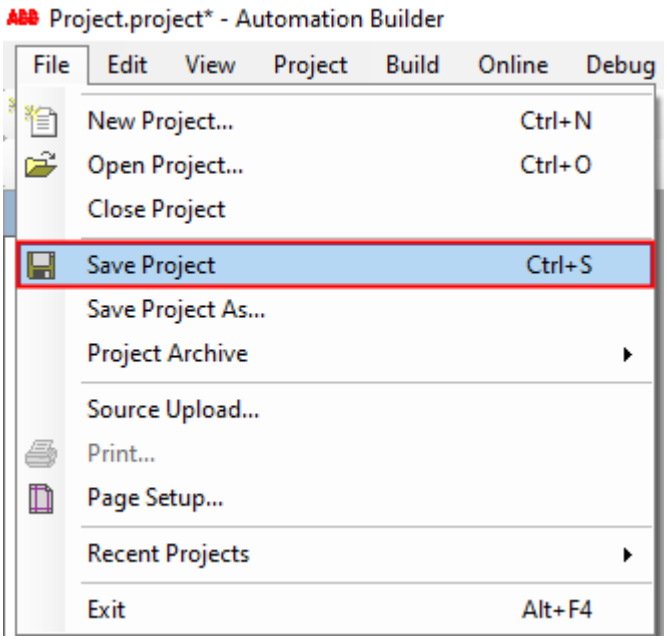


▷ Select menu *"Build ➔ Generate code"*.

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

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

Save the project



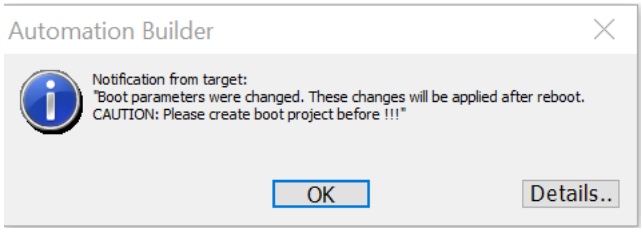
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

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

Loading the project to the CPU

1. Download the project to the CPU ↬ *as described in Chapter 1.2.18.2.2.7 , on page 148*.
2. Check the notification window at the end of the download. In case of message "Boot parameters were changed. These changes will be applied after reboot", a reboot of the CPU is required after creation of the boot project.



Create a boot project

By default, after project download, the boot project is created automatically.

Rebooting the CPU

▷ Reboot the CPU by switching OFF and ON the power supply. (The parameter for web server activation is a boot pamater which is loaded during boot of the CPU)

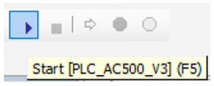
Test the web visualization

☑ You have downloaded the project and created the boot project.

☑ The CPU has been rebooted.

☑ You are logged in.

☑ CPU is in "stop" mode.



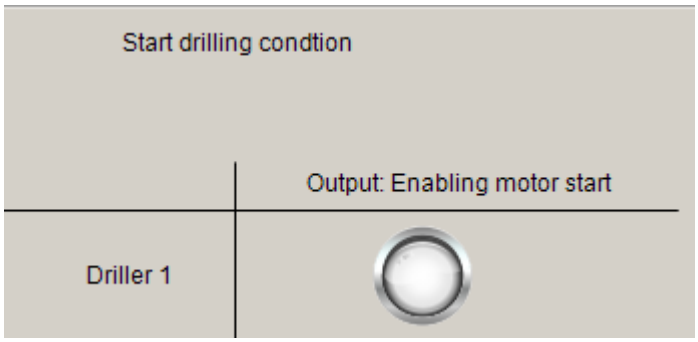
1. Start the project execution, e.g., from the tool bar.
2. Launch an internet browser.
3. Type in the URL field: *http://192.168.0.10/webvisu.htm*.

   192.168.0.10 is the IP address of CPU's ETH1 port.

   /webvisu.htm is the default htm file.

   ⇨ Web visualization will be loaded.

   The start screen "PLC_VISU" is displayed in a responsive view.



4. Test the function by operating switch I1.
5. Test the results for responsive view by changing the web browser window size.

Reset the CPU

Reset values and parameters

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

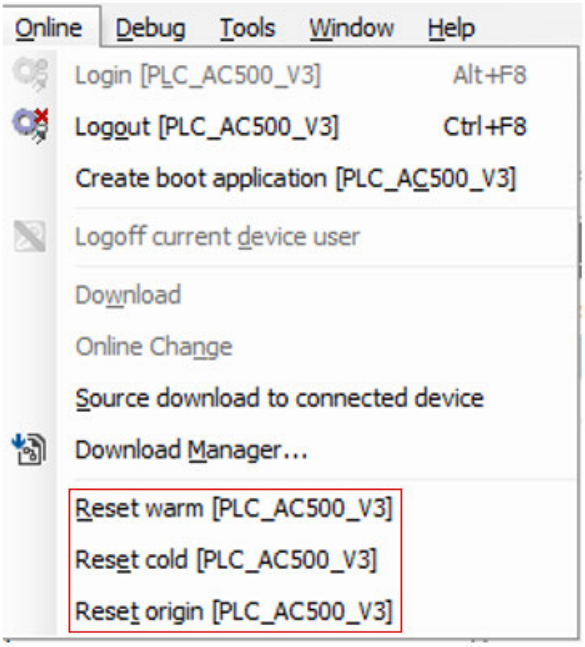


Fig. 11: Reset commands in “Online” menu

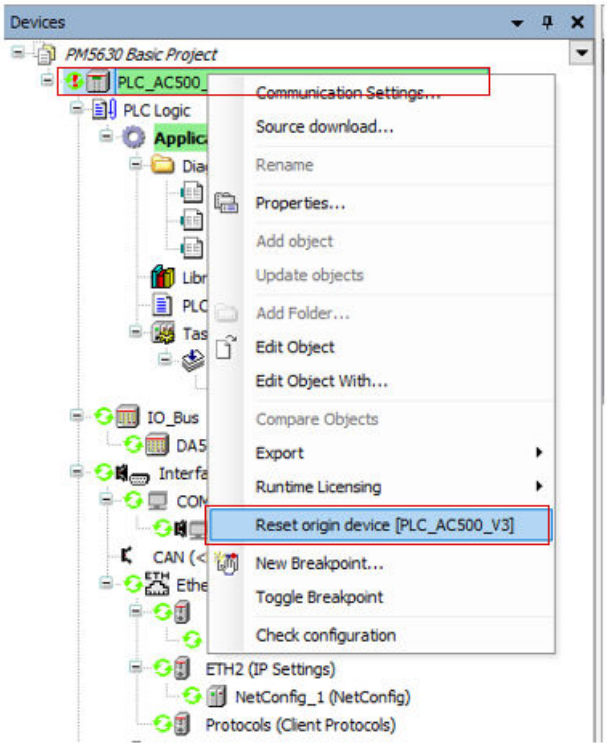
Reset warm	All variables are reset, except RETAIN PERSISTENT variables.
Reset cold	Causes initialization of all variables, except PERSISTENT variables. By recommended creation of remanent variables always with both properties: PERSISITENT and RETAIN, this command resets all variables, except PERSISTENT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

Table 10: Behavior of variables of type VAR (local or global) and variables of type PERSISTENT RETAIN

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

Complete reset of the CPU

To do a complete reset of the CPU thereby erasing the application from the RAM and flash EEPROM do the following.



1. Right-click the station object *“PLC_AC500_V3”* in the device tree.
2. Select *“Reset origin device [station name]”*.
   ⇨ The application is completely erased from the CPU (complete project from all memory areas).

1.3 Automation Builder installation manager

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

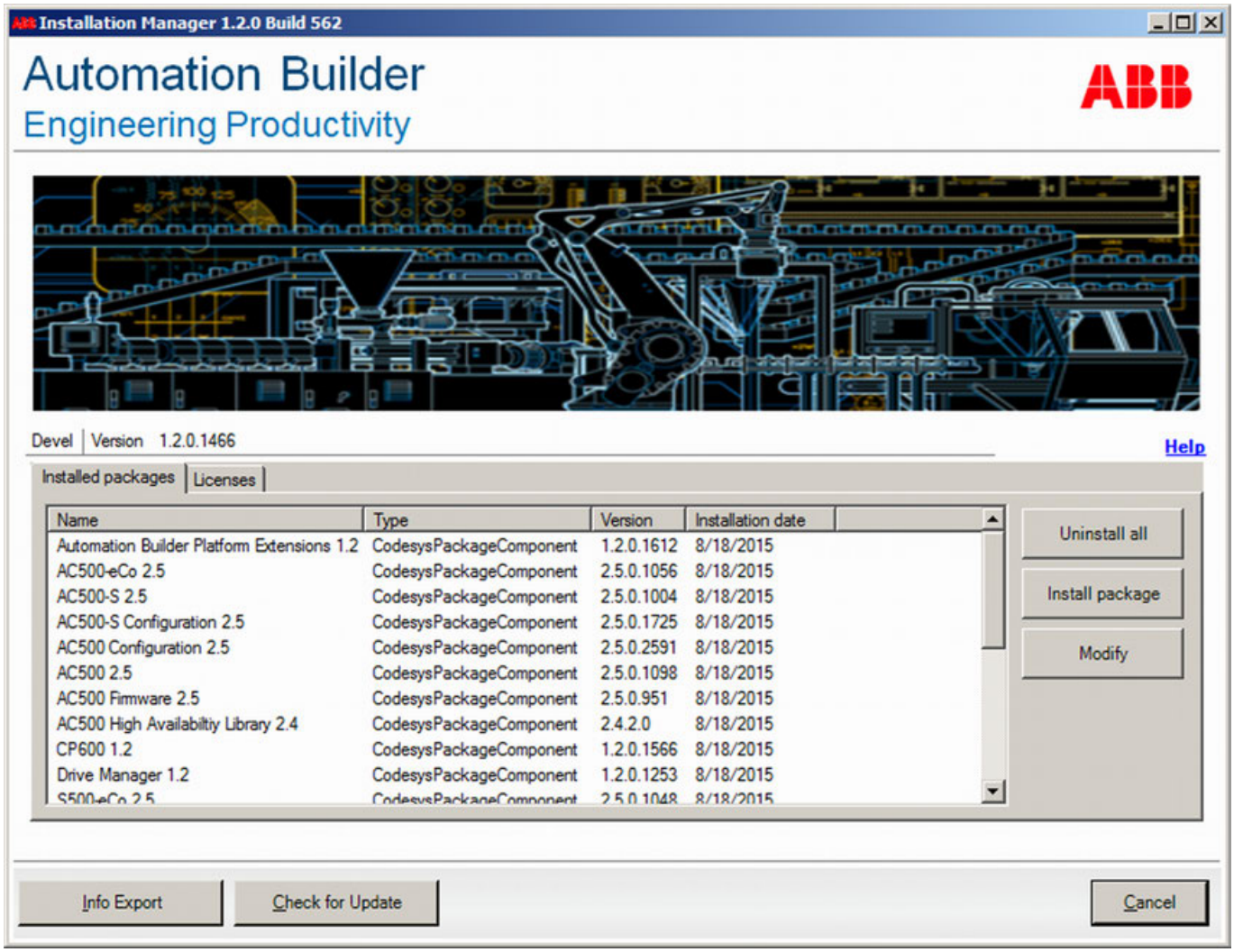
You can launch installation manager from the main menu of Automation Builder or from Windows start menu.

1. Open Automation Builder software.

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

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

   ⇨ Installation manager starts.



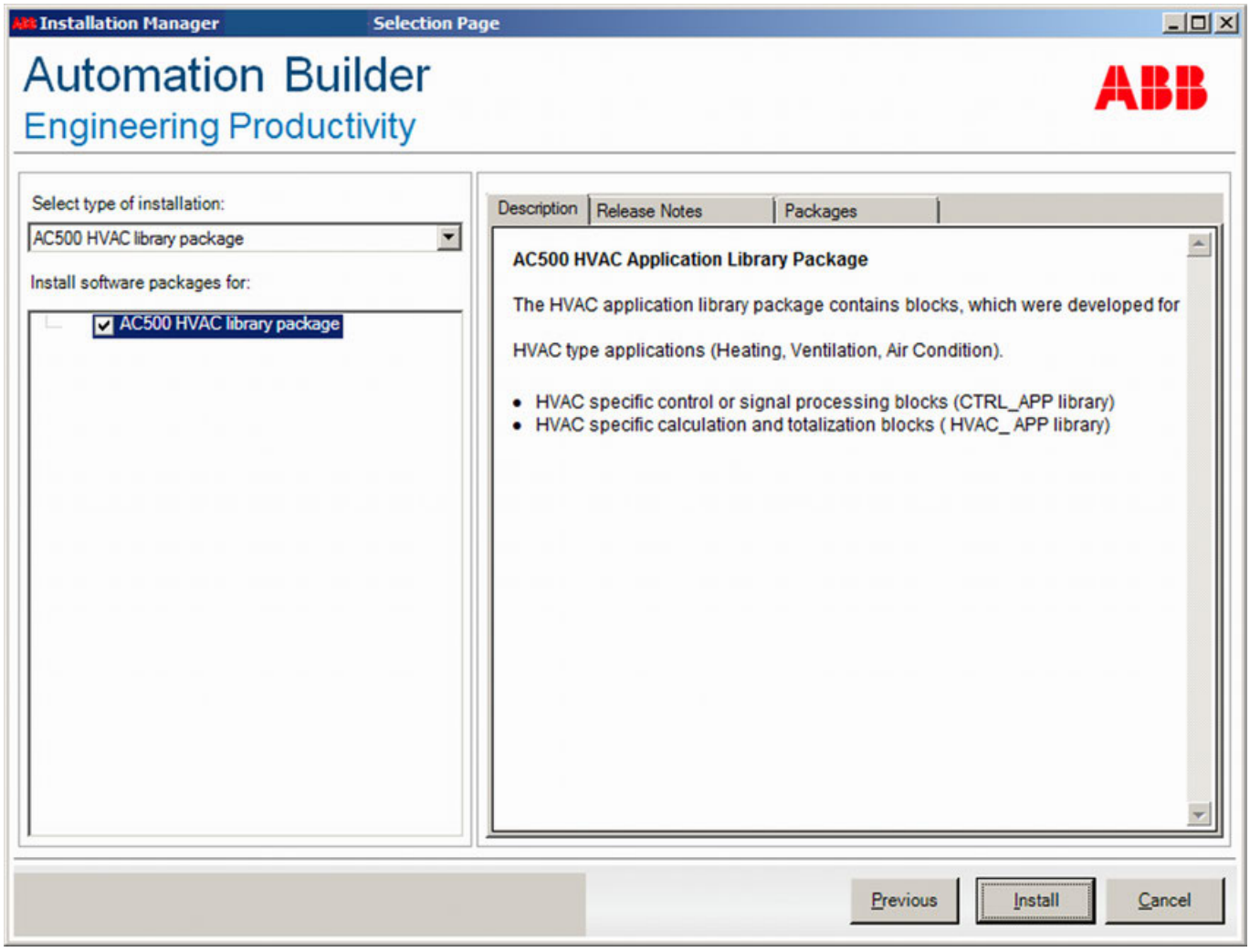
Options:

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

1.3.1 Installing customer specific package

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

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



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

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

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

   Click **Finish** to end the wizard.

1.3.2 Adding or removing installed software packages

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

   ⇨ The selection page opens.

   The selected software packages are installed already.

   The not selected software packages are not installed.

2. Select the software packages you want to install.

   Unselect the software packages you want to uninstall.

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

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

3. Click **Continue**.

   The following three cases are possible:

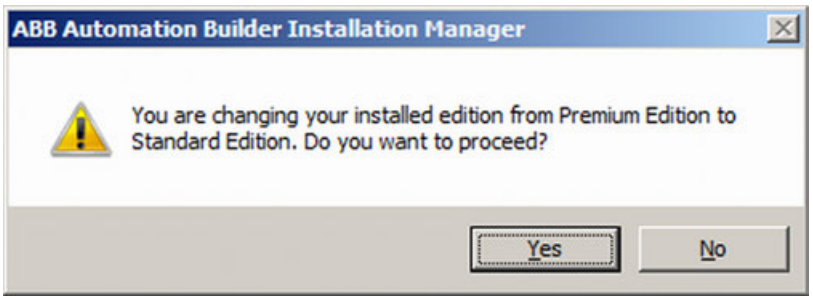
   - The selected software package starts downloading and installing.
   - The unselected software package will uninstall.
   - The unselected software package will uninstall first and then download and install the selected software package.

4. Successfully downloaded components are indicated with .

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

   Click **Finish** to end the wizard.

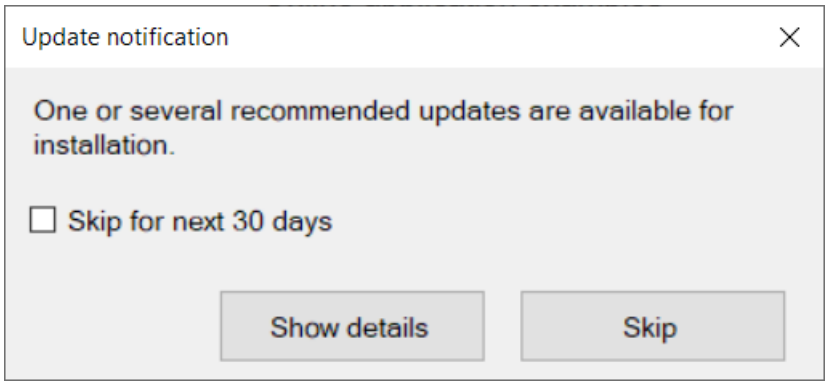
If you modify the type of installed edition, a warning message is displayed.



1.3.3 Automation Builder update notification

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

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

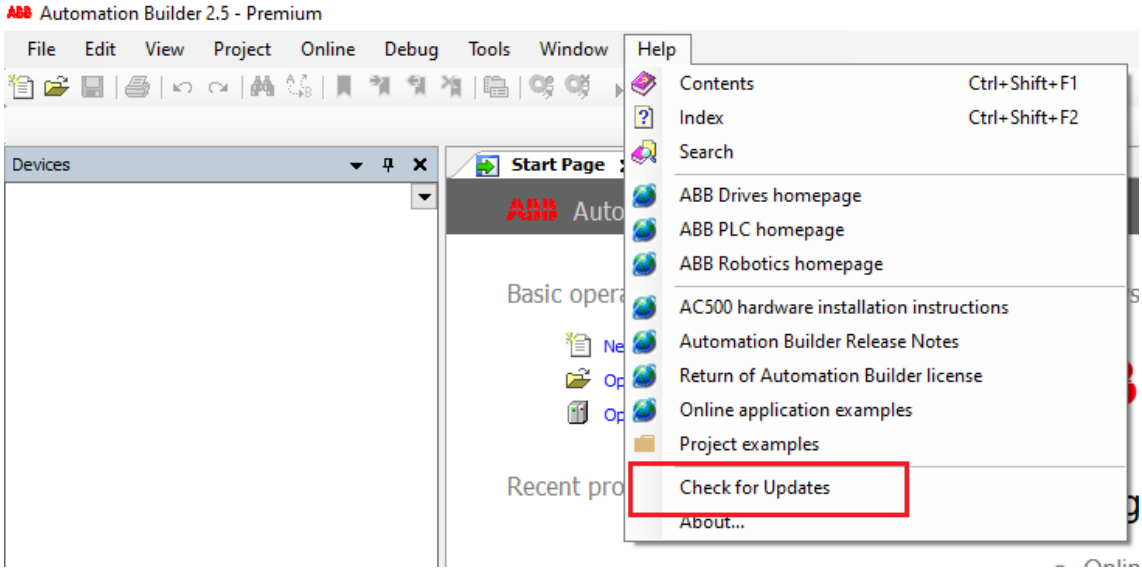


Skip of next 30 days + Skip:	Close the notification dialog. Notification dialog will not be shown for next 30 days.
Show details:	Show details will show the updates details page.
Skip:	Close the notification dialog. Next time launch of Automation Builder will show the notification dialog.

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

"Help" - "Check for Updates" menu item

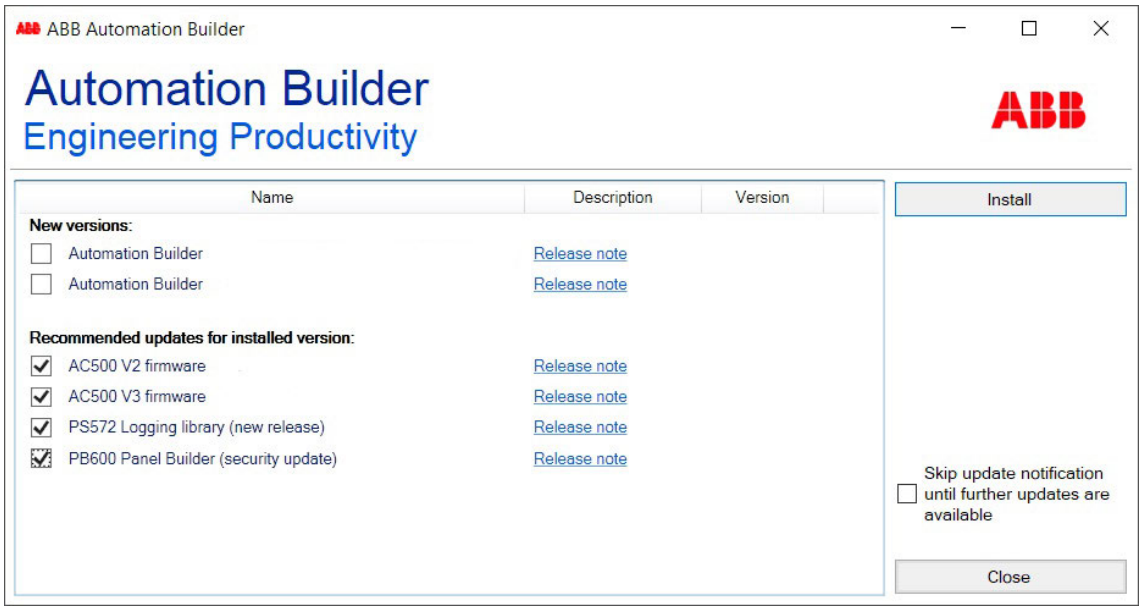
The *"Check for Updates"* menu item has been added to the *"Help"* menu. The user has the possibility to check for updates manually.



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

Automation Builder update details window

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



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

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

Installed updates in the Installation Manager start page

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

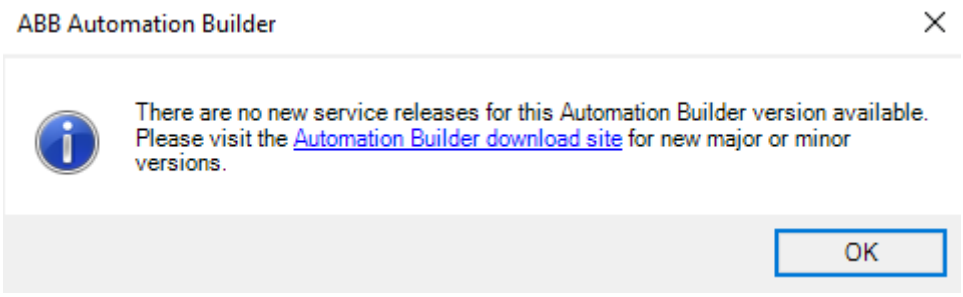
Installation Manager selection page

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

1.3.4 Checking for updates

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

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



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

> *Create a project archive before updating Automation Builder. Project archives contain all project data, including data that is not stored with a *.project file, e.g. device description files for third party devices.*
>
> *Chapter 1.6.6.1.1.7.1 “Creation of an archive ” on page 3642*

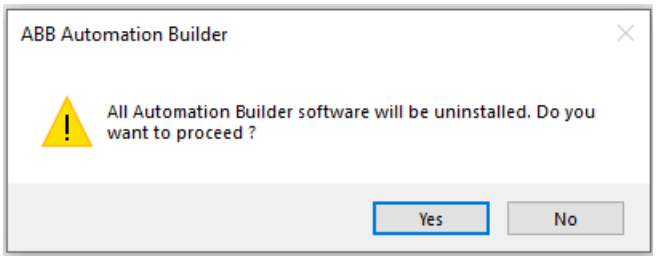
1.3.5 Uninstalling Automation Builder

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

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

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

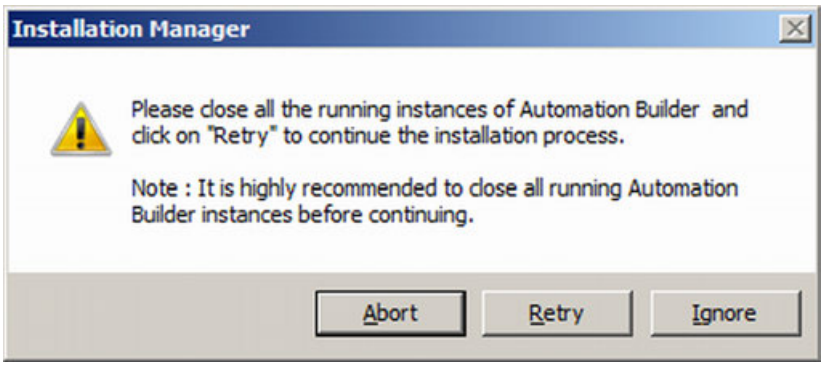
Click **Yes** to continue.



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

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

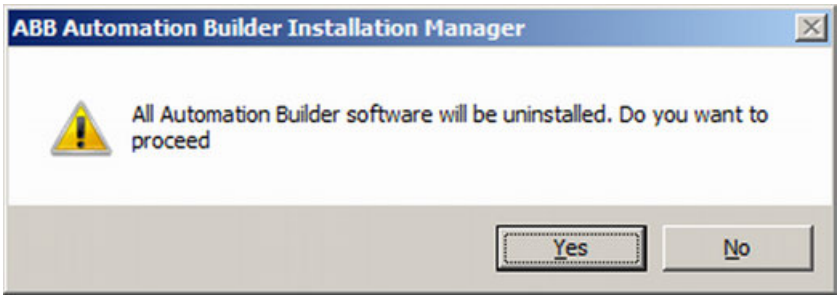
   With **Abort** uninstallation of the current package is stopped. Uninstallation is continued with the next package. With **Ignore**, uninstallation is forced. As this can lead to an erroneous uninstallation, we recommend you, *not* to use this option.



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

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

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



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

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

   Click **Finish** to end the wizard.

1.4 Programming with CODESYS

1.4.1 CODESYS Development System

Using CODESYS help

CODESYS Help is intended to assist you in using the CODESYS Development System easily and successfully. You will find quick answers to questions and solutions to problems.

Each help component consists of a concept section and a reference section.

In the concept sections, we explain in detail all topics that are relevant for creating CODESYS projects. The concepts are supplemented with instructions that lead you step-by-step to the intended result.

In the reference sections, we provide complete reference works for the user interface and programming of CODESYS.

The following formats of CODESYS Help are provided:

- CODESYS Offline Help: CHM-based CODESYS Help
- CODESYS Online Help: Web-based CODESYS Help

In the CODESYS options, you determine whether to use CODESYS Offline Help or CODESYS Online Help.

You can call the context-sensitive help directly from the user interface of the CODESYS Development System. In CODESYS, when you position the cursor over an object, menu command, or programming element, and then press the *[F1]* key, the respective help page opens. As an alternative, you can use the commands in the *"Help"* menu. This is a full-text search. The index search is possible in CODESYS Offline Help only.

Search operators for the offline help

- `AND`
  Used automatically, for example the input of the search terms `Device Diagnosis` has the same results as the input of `Device AND Diagnosis`
- The `*` placeholder is used automatically. However, the `*` character must not be used as a wildcard because in this case the `*` character will be searched for specifically.

Search operators for the online help

- `AND`
- `OR`
- `NOT`
  Example: `abc NOT abcd`: The search result includes all help pages that contain `abc` and excludes the pages with `abcd`.
- `ANDNOT`
  `ANDNOT` is the combination of the search operators `AND` nd `NOT`.
- `ANDMAYBE`
  Example: The search for `abc ANDMAYBE xyz` finds the help pages that contain `abc` and `xyz`, and all pages that contain only the string `abc`.
- Placeholders
  – `*`: Replaces any number of characters
  – `?`: Replaces exactly one character

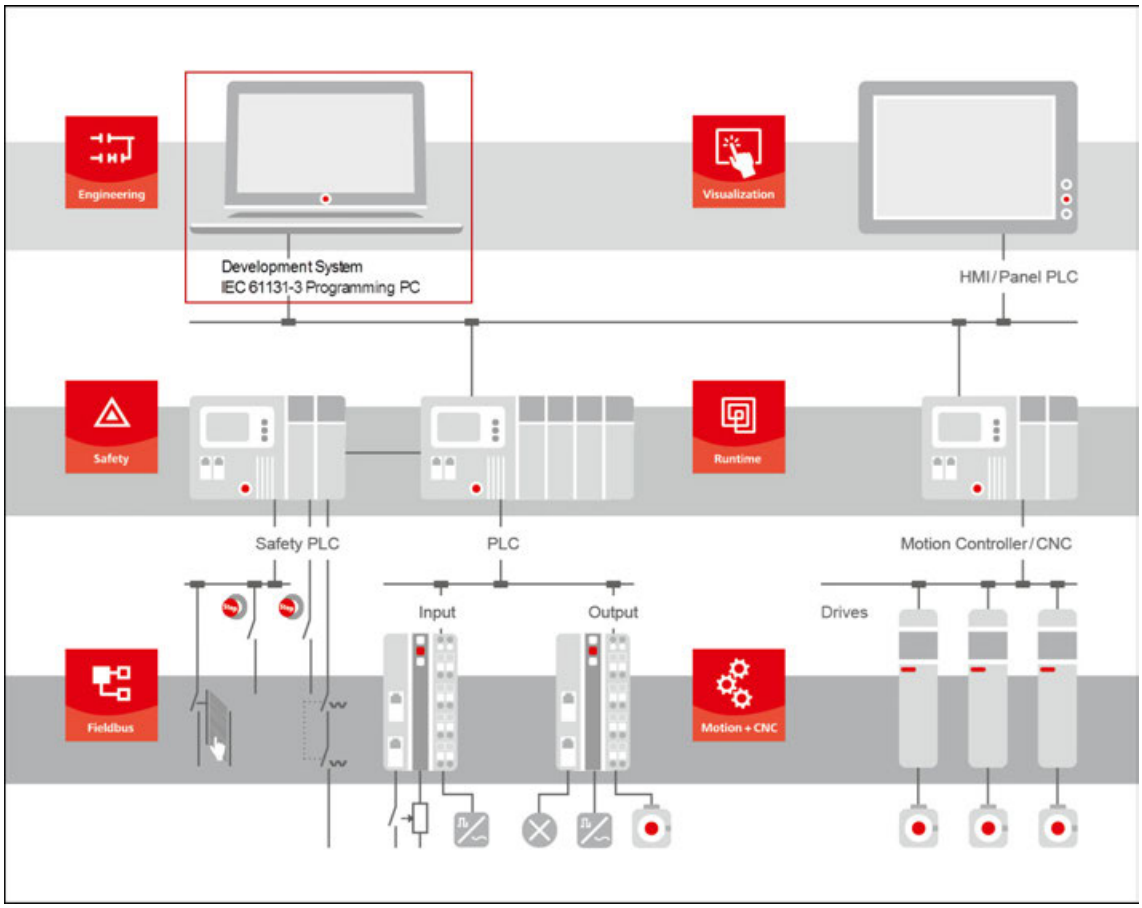
In the online help, you can use parentheses to group together multiple search operators for complex search queries. Example: `((profinet AND cycle) OR (Ethernet/IP AND cycle)) ANDNOT IRT`

See also

- *Chapter 1.4.1.20.4.13.10 "Dialog 'Options' - 'Help'" on page 1194*
- *Chapter 1.4.1.20.3.10.1 "Command 'Contents'" on page 1078*
- *Chapter 1.4.1.20.3.10.2 "Command 'Index'" on page 1078*
- *Chapter 1.4.1.20.3.10.3 "Command 'Find'" on page 1078*

CODESYS System overview

The CODESYS Development System IEC 61131-3 programming tool forms the core of the CODESYS software platform for tasks in industrial automation technology. With additional, integrated solutions for motion control, visualizations, and fieldbus connections, the usual practical requirements are covered in one system.



Features

The free CODESYS Development System is a IEC 61131-3 programming platform for automation devices with control tasks. It provides diverse and comfortable engineering solutions to support you in your developing tasks:

	For this see in this Online Help:
Project configuration through wizards.	↳ *Chapter 1.4.1.2 “Creating and Configuring a Project” on page 186*
Adaptability of the user interface.	↳ *Chapter 1.4.1.1.2 “Customizing the user interface” on page 180*
Creation of professional IEC 61131-3 controller applications wit a host of standard features.	↳ *Chapter 1.4.1.8 “Programming of Applications” on page 222*
User-friendly programming with mouse and keyboard in all IEC 61131-3 languages. Appropriate editors for FBD, LD, IL, ST, SFC, additionally the variants CFC and Extended CFC.	↳ *Chapter 1.4.1.19.1 “Programming Languages and Editors” on page 460*
Input assistance for the input and configuration of data.	↳ *Chapter 1.4.1.8.5 “Using input assistance” on page 260*

	For this see in this Online Help:
Support of object-oriented programming. Real object-oriented programming (OOP) fully compliant with the IEC 61131-3 standard in all IEC 61131-3 languages, without any additional tools. Inheritance of POUS to similar application parts to reduce development time and errors. Object-orientation is not a must: Functional and object-oriented programming can be used and mixed as required.	*Chapter 1.4.1.8 “Programming of Applications” on page 222*
Comprehensive project comparison, also for graphic editors.	*Chapter 1.4.1.4 “Comparing projects” on page 195*
Library concept for an easy reutilization of application.	*Chapter 1.4.1.16 “Using Libraries” on page 448*
Debugging and online features for the fast optimization of the application code and to speed up testing and commissioning.	*Chapter 1.4.1.11 “Testing and Debugging” on page 394*
Integrated compilers for many different CPU platforms for optimizing the controller performance.	*Chapter 1.4.1.20.4.12.2 “Dialog 'Project Environment' - 'Compiler Version'” on page 1182* *Chapter 1.4.1.20.4.11.3 “Dialog Box 'Project Settings' - 'Compileoptions'” on page 1173*
Security features for the protection of the source code and the operation of the controller.	*Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197* *Chapter 1.4.1.8.17 “Encrypting an application” on page 294* *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*
Field bus support and programming of devices from different manufacturers.	*Chapter 1.4.1.7 “Configuring I/O Links” on page 213*
Extensibility and adaptability without leaving the framework.	

Additionally:

Many seamlessly integrated tools for different kinds of automation tasks, for example CODESYS Visualization, CODESYS SoftMotion, CODESYS Application Composer.

Please always note the possibility to extend the functionalities by "AddOn"-Packages, provided in the CODESYS Store.

Customization of the user interface language

In the *“Option ➔ International Settings”* dialog you can customize the language of the user interface of the development system. This change will take effect the next time you start CODESYS. You can adjust the help language separately.

If you start CODESYS from the command line, you can add a parameter to adjust the user interface language.

See also

- *Chapter 1.4.1.20.4.13.13 “Dialog 'Options' – 'International Settings'” on page 1195*
- *Chapter 1.4.1.15 “Using the Command-Line Interface” on page 442*

1.4.1.1 Configuring CODESYS

CODESYS Development System allows to configure the behavior, the appearance, the content of the menus and the arrangement of the windows individually. In the *"Tools"* menu you find dialogs for customizing the user interface and to setup the CODESYS options.

See also

- *Chapter 1.4.1.20.4.14 "Dialog 'Customize'" on page 1205*
- *Chapter 1.4.1.20.4.13 "Dialog 'Options'" on page 1186*

1.4.1.1.1 Setting CODESYS options

You can configure the behavior and appearance of the CODESYS Development System in the different tabs of the *"Options"* dialog. The dialog opens by clicking *"Tools ➔ Options"*. Here you can configure the default settings for different editors and functionalities. These settings apply throughout CODESYS.

The settings are stored in your current user profile on your local system. For use on other systems, option settings, either user-specific or machine-specific (computer), can be exported to an XML file.

In V3.5 SP13 and later, CODESYS checks whether an older version is already installed when the development system is started for the first time. If this is the case, then the "Import Assistant" dialog opens for transferring the CODESYS options set with the older version.

See also

- *Chapter 1.4.1.20.3.8.17 "Command 'Options'" on page 1071*
- *Chapter 1.4.1.20.3.8.18 "Command 'Import and Export Options'" on page 1072*
- *Chapter 1.4.1.20.4.1 "Dialog 'Import Assistant'" on page 1149*
- *Chapter 1.4.1.1.2.1 "Customizing menus" on page 180*
- *Chapter 1.4.1.1.2.4 "Customizing keyboard shortcuts" on page 183*
- *Chapter 1.4.1.1.2.2 "Customizing toolbars" on page 182*

1.4.1.1.2 Customizing the user interface

In CODESYS, you can customize the user interface by changing the window layout as well as the appearance of menus and commands according to your requirements.

Customizing menus

You can customize the menu commands of the CODESYS user interface. In a configuration dialog, you can add or remove menus.

Removing menus and commands

1. Choose the command *"Tools ➔ Customize"*.
   ⇨ The *"Customize"* dialog box opens. The *"Menu"* tab is visible.

2. Select a menu in the menu tree or a command in a menu.
3. Click *“Delete”*.
   ⇨ The menu or command is deleted from the menu tree.
4. Click *“OK”*.
   ⇨ The dialog box closes and the menu is customized.

Adding menus

1. Choose the command *“Tools ➔ Customize”*.
   ⇨ The *“Customize”* dialog box opens. The *“Menu”* tab is visible.
2. Scroll to the end of the menu tree.
3. Select the blank symbol (□).
4. Click *“Add Popup Menu”*.
   ⇨ The *“Add Popup Menu”* dialog box opens.
5. Type a name for the new menu in the *“Default text”* field.
   If localization is unnecessary, then skip to step 9.
6. Click *“Add Language”*.
   ⇨ A drop-down list opens with available languages.
7. Choose the required language.
   ⇨ The language is added to the list of languages.
8. Click into the *“Text”* field and type the language-specific text.
9. Click *“OK”*.
   ⇨ The new menu is added at the bottom of the menu tree.
10. Change the menu order by clicking *“Move up”* and *“Move down”*. Click *“OK”* to close the *“Customize”* dialog box.

The new menu is displayed only when it contains a command.

Adding commands

1. Choose the command *“Tools ➔ Customize”*.
   ⇨ The *“Customize”* dialog box opens. The *“Menu”* tab is visible.
2. Expand the branch of the menu where the new command should be added.
3. Select the blank symbol (□).
4. Click *“Add Command”*.
   ⇨ The *“Add Command”* opens dialog box.
   The dialog box lists all commands grouped by category.
5. Select the command to be added. Click *“OK”*.
   ⇨ The new command is added to the menu tree.

6. Change the menu order by clicking *“Move up”* and *“Move down”*. Click *“Add separator”* to add a border between commands. Click `OK` to close the *“Customize”* dialog box.

   ⇨ The new command is now available in the menu.

See also

- *Chapter 1.4.1.20.4.14.1 “Dialog 'Customize' - 'Menu'” on page 1206*
- *Chapter 1.4.1.1.2.2 “Customizing toolbars” on page 182*

Customizing toolbars

You can customize the toolbars of the CODESYS user interface. In a configuration dialog, you can add or remove toolbars.

Removing toolbars and commands

1. Choose the command *“Tools ➔ Customize”*.

   ⇨ The *“Customize”* dialog box opens.

2. Choose the *“Toolbars”* tab.
3. Select a toolbar or a command from a toolbar tree.
4. Click *“Delete”*.

   ⇨ The toolbar or command is deleted.

5. Click *“OK”*.

   ⇨ The dialog box closes and the toolbar or command is removed.

Adding toolbars

1. Choose the command *“Tools ➔ Customize”*.

   ⇨ The *“Customize”* dialog box opens.

2. Choose the *“Toolbars”* tab.
3. Select the blank toolbar.
4. Click *“Add Toolbar”*.

   ⇨ The cursor blinks in the new toolbar.

5. Type a name.
6. Change the toolbar order by clicking *“Move up”* and *“Move down”*. Click *“OK”* to close the *“Customize”* dialog box.

CODESYS displays the new toolbar only when it contains a command.

Adding commands

1. Choose the command *“Tools ➔ Customize”*.

   ⇨ The *“Customize”* dialog box opens.

2. Choose the *“Toolbars”* tab.
3. Expand the tree of the toolbar where the new command should be added.
4. Select the blank symbol ( ).
5. Click *“Add Command”*.
   ⇨ The *“Add Command”* dialog box opens.
   The dialog box lists all commands grouped by category.
6. Select the command to be added. Click *“OK”*.
   ⇨ The new command is added to the toolbar tree.
7. Change the toolbar order by clicking *“Move up”* and *“Move down”*. Click *“Add separator”* to add a border between commands. Click *“OK”* to close the *“Customize”* dialog box.
   ⇨ The new command is available in the toolbar.

See also

- *Chapter 1.4.1.20.4.14.3 “Dialog 'Customize' - 'Toolbars'” on page 1207*
- *Chapter 1.4.1.1.2.1 “Customizing menus” on page 180*

Customize command icon

CODESYS provides the capability of assigning customized icons to commands.

1. Select the command *“Tools ➔ Customize”*.
   ⇨ The *“Customize”* dialog box opens.
2. Click the *“Command icons”* tab.
3. Select the category *“Help”* from the list on the left.
   ⇨ All commands in this category are listed on the right.
4. Select the command *“Information”*.
5. Click *“Assign”*.
   ⇨ A dialog box opens for selecting the icon file (*.ico).
6. Select an icon file.
7. Click the *“Open”* button.
   ⇨ The icon is assigned to the selected command.
8. Click *“OK”*.

See also

- *Chapter 1.4.1.20.4.14.2 “Dialog 'Customize' - 'Command Icons' ” on page 1206*

Customizing keyboard shortcuts

CODESYS provides the capability of executing commands directly via keyboard shortcuts. You can customize or extend predefined keyboard shortcuts.

1. Choose the command *“Tools ➔ Customize”*.
   ⇨ The *“Customize”* dialog box opens.

2. Choose the *“Keyboard”* tab.
3. Select the category *“Help”* from the list on the left.
   ⇨ All commands in this category are listed on the right.
4. Select the command *“Search”*.
5. Click into the field *“Press Shortcut Keys”*.
6. Press *[Ctrl]+[Shift]+[S]*.
   ⇨ CODESYS adds the key combination to the field.
7. Click *“Assign”*.
   ⇨ The keyboard shortcut is assigned to the command.
8. Click *“OK”*.
   ⇨ You can call the *“Search”* command by pressing *[Ctrl]+[Shift]+[S]*.

See also

- *Chapter 1.4.1.20.4.14.4 “Dialog Box 'Customize' - 'Keyboard' ” on page 1207*

Changing the window layout

In CODESYS, you can easily customize the layout of different views to your individual needs.

1. Drag the view by the caption bar or by the tab.
   ⇨ Arrows are shown to mark possible destinations. Example: 
2. Drag the view to one of the arrows.
   ⇨ The destination is displayed as a blue-shaded area.
3. Release the left mouse button.
   ⇨ The window is inserted into the selected destination.

The window can also be placed outside of the CODESYS programming interface.

See also

- *Chapter 1.4.1.1.2.6 “Resizing windows” on page 184*
- *Chapter 1.4.1.1.2.7 “Auto-hiding windows” on page 185*
- *Chapter 1.4.1.1.2.8 “Switching between windows” on page 185*

Resizing windows

1. Move the mouse pointer over the border between two windows or views.
   ⇨ The cursor becomes a left-right arrow.
2. Drag the border to another position.

You can resize detached views by moving the frame lines.

See also

- *Chapter 1.4.1.1.2.5 “Changing the window layout” on page 184*
- *Chapter 1.4.1.1.2.7 “Auto-hiding windows” on page 185*
- *Chapter 1.4.1.1.2.8 “Switching between windows” on page 185*

Auto-hiding windows

Hiding windows

When you hide a view, it is minimized to a tab in the frame of the user interface. When you move the pointer over the tab, the window is shown automatically.

1. Click into the window to be hidden.
2. Click *“Window ➔ Auto Hide”*.

   Or click the PIN symbol ( ) in the upper right corner of the view.

   ⇨ The window is hidden and only visible by a small tab on the edge of the main window.
3. Move the mouse pointer over the tab.

   ⇨ The window is shown as long as the mouse pointer hovers over the tab.

Showing windows

1. Click the tab of the hidden window.
2. Clear the check box *“Window ➔ Auto Hide”*.

   Or click the PIN symbol ( ) in the upper right corner of the view.

   ⇨ The window is permanently shown.

See also

- *Chapter 1.4.1.1.2.5 “Changing the window layout” on page 184*
- *Chapter 1.4.1.1.2.6 “Resizing windows” on page 184*
- *Chapter 1.4.1.1.2.8 “Switching between windows” on page 185*

Switching between windows

It is possible to switch directly between the currently opened views and the editor windows.

1. Press the keystroke combination *[Ctrl]+[Tab]*. Continue pressing the *[Ctrl]* key.

   ⇨ An overview opens with all active views and editors.
2. Continue pressing the *[Ctrl]* key and select a window using the arrow keys.
3. Release the *[Ctrl]* key.

   ⇨ The selected view or editor is activated.

See also

- *Chapter 1.4.1.1.2.5 “Changing the window layout” on page 184*
- *Chapter 1.4.1.1.2.6 “Resizing windows” on page 184*
- *Chapter 1.4.1.1.2.7 “Auto-hiding windows” on page 185*

1.4.1.2 Creating and Configuring a Project

What is a project?

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

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

See also

- *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*
- *Chapter 1.4.1.20.2 “Objects” on page 818*
- *Chapter 1.4.1.20.4 “Dialogs” on page 1149*
- *Chapter 1.4.1.20.3.4.13 “Command 'Project information'” on page 1007*
- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*

1.4.1.2.1 Opening a V3 Project

You can open projects, library projects, or project archives in CODESYS which have been created with different installations. When a project is opened, it is automatically checked whether or not the active installation is appropriate to load the project. At this time, deficiencies can be detected, such as missing add-ons or deprecated installations. You can correct these deficiencies. Then you can load the project with an appropriate installation.

The following actions are possible to correct deficiencies:

- Update existing add-ons and install missing add-ons
- Start another installation which is appropriate for the project
- Install an additional CODESYS version with the appropriate state

Loading a write-protected project

Moreover, you can load and read write-protected projects. You have to specify an appropriate location where you have the necessary write permissions only when you save the file.

Loading a project with access restrictions

You can load restricted projects only if you have the access credentials, such as user name and password.

Loading a project with a security key

You have selected a project which is protected by a security key. If the security key is not plugged into the computer, then you are prompted to plug it in. Otherwise CODESYS opens the project without any information about the protection.

Loading a backup of a project

Backups are created when the *"Automatically save"* project option is selected. When CODESYS is not ended properly after a change, the project is saved as a backup.

When you have selected a project, the *"Auto Save Backup"* dialog opens first when loading. There you can handle the backup.

See also

- *Chapter 1.4.1.2.2 "Opening a V2.3 project" on page 187*
- *Chapter 1.4.1.20.4.13.16 "Dialog 'Options' – 'Load and Save'" on page 1196*
- *Chapter 1.4.1.5.1 "Setting up write protection" on page 201*
- *Chapter 1.4.1.5.2 "Assigning Passwords" on page 202*
- *Chapter 1.4.1.20.3.1.2 "Command 'Open Project'" on page 957*

See also

- Help on CODESYS Installer

1.4.1.2.2 Opening a V2.3 project

A CoDeSys V2.3 project can be converted into a CODESYS V3 project only if the CODESYS V2.3 Converter package is installed in CODESYS V3. The package is available in the CODESYS Store.

Requirement: CODESYS is started (or a project is already open). You should be aware of the restrictions described below the following instructions.

1. Click *"File ➔ Open Project"*.
2. In the *"Open Project"* dialog, click any CoDeSys V2.3 project or project archive in the file system. For searching, you can set the file filter on the bottom right corner of the dialog.
   ⇨ If another project is still open, CODESYS instructs you to close it accordingly. After that the CoDeSys V2.3 converter automatically starts.
3. The V2.3 converter checks that the project can be compiled without errors. If so, then it processes the project automatically.
4. NOTE: If the project contains visualization objects with placeholder variables that the converter cannot resolve, the respective visualizations are shown as a group in place of the visualization references.
5. Device conversion: When a device (target system) is referenced in the project to be opened and no conversion rules are defined for the device, then the *"Device Conversion"* dialog opens. Specify here whether and how the converter should replace the previous device reference with a current one.
   ⇨ For replacement, the converter added the new device in the place of the old one in the device tree of the converted project.

6. Library conversion: if a library, for which no conversion rule has so far been defined, is referenced in the project to be opened, then the *“Conversion of Library Reference”* dialog opens. Specify here whether and how the converter should replace the existing library reference with a current one. If you select a library for which the project information is missing, then the *“Enter Project Information”* dialog opens in order to specify this information.

   ⇨ The converter loads the adapted project. Note: The redefined library references are to be found in the global Library Manager in the POUs view.

Restrictions when reusing a CoDeSys V2.3 project in CODESYS

Compilation:	The project has to be compilable without errors in CoDeSys V2.3. Note: CODESYS stills issues warnings in V3 when compiling. These are caused by implicit conversions, which can lead to a loss of information (for example through sign changes). CODESYS checks "case" statements against the switch variable: `CASE USINT OF INT` is not checked in CoDeSys V2.3, but it issues an error message when imported into V3.
Controller configuration:	The *“Controller Configuration”* of a CoDeSys V2.3 project cannot be imported into V3. You have to recreate the device configuration and re-declare the variables used in the controller configuration.
Network variables:	For network variables, CODESYS creates V3 GVL objects and imports the variable declarations. However, the network properties are not imported. See the description of the network variable exchange for this.
Libraries:	All variables and constants that are used in a library also have to be declared in the library. It must be possible to compile the library in CoDeSys V2.3 without errors.

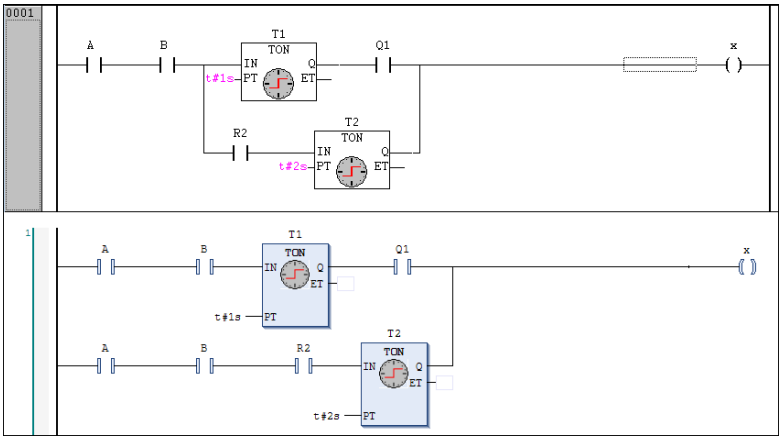
Syntactic and semantic restrictions since CoDeSys V2.3:	● `FUNCTIONBLOCK` is not a valid keyword instead of `FUNCTION_BLOCK`.
	● `TYPE` (declaration of a structure) must be followed by a “:”.
	● `ARRAY` initialization** must have parentheses.
	● `INI` is no longer supported (you have to replace this in the code by the Init method).
	● In function calls it is no longer possible to mix explicit with implicit parameter assignments. Therefore the order of the parameter input assignments can be changed:
	`fun(formal1 := actual1, actual2); // -> error message`
	`fun(formal2 := actual2, formal1 := actual1); // same semantics as the following line:`
	`fun(formal1 := actual1, formal2 := actual2);`
	● CoDeSys V2.3 pragmas are not converted. They produce an warning in V3.
	● The `TRUNC` operator now converts to the data type `DINT` instead of `INT`. CODESYS automatically adds a corresponding type conversion for a CoDeSys V2.3 import.

Visualization:				
Placeholders and their replacement	Placeholders	VAR_INPUT	Usage	Replacement
	`PLC_PRG.$LocalVar$.aArr[0]`	`localVar: MyStruct;`	`localVar.aArr[0]`	`localVar := PLC_PRG.myStructVar`

	`$Var$.aArr[0]`	`Var : MyStruct;`	`Var.aArr[0]`	`Var := PLC_PRG.myStructVar`
	`PLC_PRG.myStructVar.aArr[$Index$]`	`Index : INT;`	`PLC_PRG.myStructVar.aArr[Index]`	`Index := 0`

Problematic placeholders	• Placeholders within a text:
	Text: `$axle$-Axis`
	Correction:
	`localVar : STRING;`
	Text: `%s-Axis`
	Text variable: `localVar`
	• Placeholder describes only one part of a variable name:
	`axis$axis$spur$spur$.fActPosition`
	Correction:
	Define only one placeholder for the `axis$axis$spur$spur$` placeholder.
	`axis_spur : MyFunctionBlock;`
	Then directly transfer the corresponding instance of the function block.
	`axis_spur := PLC_PRG.axis1spur2;`
	• Placeholder is replaced by an expression:
	`$Expression$ -> PLC_PRG.var1 + PLC_PRG.var2`
	Correction:
	You must transfer the expression to an auxiliary variable and then transfer this auxiliary variable as an instance.
	• The placeholder describes a program name: $Program$.bToggle -> PLC_PRG.bToggle D
	The converter cannot transfer this form of setting placeholders in V3. However, you will rarely use it in practice.
	• Placeholder is replaced by different types:
	`$Var$`
	-> replacement 1 : `PLC_PRG.n` (INT)
	-> replacement 2 : `PLC_PRG.st` (STRING)
	Correction:
	Define two different placeholders in the interface for this.
	• The visualization is located in a library. You replace the placeholder later from any desired project when you use the visualization there.
	Correction:
	Here you have to replace the `TYPE_NONE` data types manually. However, there is also the possibility for you to integrate the library in a project and the placeholder is correctly replaced. If you now import this project, the data type is also determined correctly in the library.			

Non-importable elements:	Trend, ActiveX – the import is not possible, because the implementation differs a great deal. In V3, a corresponding warning is issued and a corresponding manual reproduction is required.			
Programming languages	ST, IL, FBD:	No restrictions		

	LD:	CODESYS imports function blocks with parallel branches in such a way that the part before the branch is repeated for each branch. This corresponds to the generated code that CoDeSys V2.3 creates for parallel branches. 
	SFC:	• Step variables explicitly declared by the user must be declared locally in the SFC editor. You may not declare them as VAR_INPUT, VAR_OUTPUT or VAR_INOUT, because CODESYS cannot automatically adapt the calls. Explanation: Steps no longer use Boolean variables for the management of the internal states in V3, but also structures of the type `SFCStepType`. • Identifier: the following identifiers may not begin with an underscore character: – Names of IEC actions in the tree – Variables that are called in an IEC association list – Names of transitions that have been programmed out Explanation: In V3 the implicit variables that CODESYS creates for actions are given an underscore character as prefix. An invalid identifier with a double underscore character would result.
	CFC:	• Large boxes: The layout of large boxes can lose quality due to an import. The boxes may overlap one another too much. (Correction planned). • Macros: Macros cannot be imported. (Correction planned).

See also

- *Chapter 1.4.1.20.2.21 “Object 'Project Information'” on page 919*
- *Chapter 1.4.1.20.4.4 “Dialog 'Device Conversion' ” on page 1151*
- *Chapter 1.4.1.20.4.2 “Dialog 'Library Reference Conversion'” on page 1150*

1.4.1.2.3 Configuring a Project

You can configure your CODESYS project using the following dialogs:

- *“Project Settings”*: Basic settings on the behavour of editors and of the compiler, on user management etc.
- *“Project Information”*: Adding of individual and tagging information to the project
- *“Project Environment”*: Defining which versions of the external and internal modules should be used, with the aim of achieving up-to-dateness and compatibility with each other.

See also

- ↳ *Chapter 1.4.1.20.4.11 “Dialog 'Project Settings'” on page 1170*
- ↳ *Chapter 1.4.1.20.2.21 “Object 'Project Information'” on page 919*
- ↳ *Chapter 1.4.1.20.4.12 “Dialog 'Project Environment'” on page 1182*

Retrieving and Editing Project Information

You can use the *“Project Information”* object to retrieve information about your project and the associated file, and edit certain information.

The object contains information about

- File attributes
- Meta-information, such as manufacturer, title, or author
- Properties with keys
- Statistics
- Licensing
- Signing: This way of signing translated libraries is deprecated, and for security reasons should only be used if compatibility with older versions is required. If this method is used, then later you can use a public key token to verify that the library was last signed by the library vendor. As a library vendor, it is therefore crucial that you make the public key used available to the customer, for example in the documentation.

CODESYS saves the project information as an object within the project. When you transfer a project to another system, the *“Project Information”* object is transferred with it. There is no need for a project archive.

You can use property keys to access the project information externally via function blocks. For a library project, you can also query information about the licensing.

Editing meta-information

1. Click *“Project ➔ Project Information”*.
   ⇨ The *“Project Information”* dialog opens.
2. Click the *“Summary”* tab.
3. Specify your data in the input fields (example: `0.0.0.1` in the *“Version”* input field).
   ⇨ CODESYS creates a property with a key for each given value and manages them on the *“Properties”* tab. For a library project, CODESYS still uses the properties and sorts later in the library repository.

   If you select the option for CODESYS to create a functions block for these properties, then you can access the properties programmatically.

Creating functions for accessing properties

1. Click *“Project ➔ Project Information”*.
   ⇨ The *“Project Information”* dialog opens.
2. Select the *“Automatically generate 'Project Information' POUs”* option.

Example

Requirement: The following property is defined.

```
Key = nProp1
Type= number
Value= 333
```

1. Select the *“Automatically generate 'Project Information' POUs”* option.
2. Declare a property of the type `DINT`, for example `showprop : DINT;`.
3. Call the function `GetNumberProperty`: `showprop := GetNumberProperty("nProp1");`

   ⇨ You are granted access to the value in the application.

Note: The functions that are created with the “Automatically generate 'Project Information' POUs” option can be used only if the runtime supports the `WSTRING` data type. If this is not the case, then instead you can apply the “Automatically generate 'Library Information' POUs” option. You can use the functions created in this way at least in the application to access properties. These functions are not registered in the runtime.

Licensing library projects

If your project is a library project, then you can activate the library licensing in use here. The CODESYS Security Key is a dongle.

☑ Requirement: The project is a library project.

1. Click *“Project ➔ Project Information”*.

   ⇨ The *“Project Information”* dialog opens.
2. Click the *“Licensing”* tab.
3. Select the *“Activate dongle licensing”* option.
4. Specify the dongle data in *“Firm code”*, *“Product code”*, *“Activation URL”*, and *“Activation mail”*.

   ⇨ The library is licensed.

Creating private key files

1. Click *“Project ➔ Project Information”*.
2. Click the *“Signing”* tab.
3. Click the *“Create Private Key File”* button.

   ⇨ The *“Create Private Key File”* dialog opens.
4. Select a safe location, e.g. `D:\for lib developers only\mycomp_libkey.libpk` and exit the dialog with *“Save”*.

See also

- *Chapter 1.4.1.20.2.21 “Object 'Project Information'” on page 919*

Making project settings

You can configure settings that affect the behavior of CODESYS and that of certain editors in the *"Project Settings"* object. The settings are valid throughout the project and are applied immediately for active editors. You can also access the dialog boxes of the object with the command *"Project ➔ Project Settings"*.

CODESYS saves the project settings as an object directly in the project. If you then transfer a project to another system, the *"Project Settings"* object is also transferred with it, without a project archive being required.

See also

- *Chapter 1.4.1.10.7 "Downloading source code to and from the PLC" on page 393*
- *Chapter 1.4.1.8.12.2 "Analyzing code statically" on page 283*
- *Chapter 1.4.1.20.3.4.14 "Command 'Project Settings'" on page 1007*
- *Chapter 1.4.1.20.4.11.1 "Dialog 'Project Settings' - 'SFC'" on page 1171*
- *Chapter 1.4.1.20.4.11.2 "Dialog 'Project Settings' - 'Users and Groups'" on page 1172*
- *Chapter 1.4.1.20.4.11.3 "Dialog Box 'Project Settings' - 'Compileoptions'" on page 1173*
- *Chapter 1.4.1.20.4.11.4 "Dialog Box 'Project Settings' - 'Compiler Warnings'" on page 1173*
- *Chapter 1.4.1.20.4.11.5 "Dialog 'Project Settings' – 'Source Download'" on page 1174*
- *Chapter 1.4.1.20.4.11.6 "Dialog 'Project Settings' - 'Page Setup'" on page 1175*
- *Chapter 1.4.1.20.4.11.7 "Dialog 'Project Settings' - 'Security'" on page 1176*
- *Chapter 1.4.1.20.4.11.8 "Dialog 'Project Settings' - 'Static Analysis Light'" on page 1177*
- *Chapter 1.4.1.20.4.11.9 "Dialog 'Project Settings' - 'Visualization'" on page 1180*
- *Chapter 1.4.1.20.4.11.10 "Dialog 'Project Settings' - 'Visualization Profile'" on page 1181*

1.4.1.3 Exporting and Transferring Projects

Export and import functions are available to you for the exchange of the data from CODESYS projects with other programs.

An exchange of CODESYS projects between CODESYS development systems takes place by way of a copy of the project file (*.project) or project archive (*.projectarchive).

See also

- *Chapter 1.4.1.3.1 "Exporting and importing projects" on page 193*
- *Chapter 1.4.1.3.2 "Transferring Projects" on page 194*

1.4.1.3.1 Exporting and importing projects

CODESYS offers commands for the export and import of objects to and from a file. Two possibilities are available to you here:

- Export to or import from a CODESYS XML file (*.export)
  This format is completely compatible with the CODESYS project format. The objects are saved in a machine-readable XML format.
- Export to or import from an XML file in the PLCopen format (*.xml)
  You can use this format to exchange information with other programs (for example program editors or documentation tools). PLCopen XML defines a subset of the elements known in CODESYS. 100% compatibility is thus not guaranteed.

Exporting projects

Requirement: A project is open in CODESYS.

1. Select the command *"Project ➔ Export…"* or *"Project ➔ Export PLCopenXML"*
2. Select the objects that you wish to export in the dialog box *"Export"* or *"Export PLCopenXML"*.
3. Click on *"OK"*.

4. Enter the file name and the location and click on *“Save”*.

Importing projects

Requirement: A project is open in CODESYS.

1. Select the command *“Project ➔ Import…”* or *“Project ➔ Import PLCopenXML”*.
2. In the dialog box *“Import”* or *“Import PLCopenXML”*, select the export file that you wish to import.
   ⇨ A dialog box opens and displays the objects in a tree structure, which can be inserted at this point.
3. Select the object in the object tree, under which the objects to be imported are to be inserted.
4. Select the objects and click *“OK”*.
   ⇨ The objects are added to the existing object tree.

See also

- *Chapter 1.4.1.20.4.13.19 “Dialog 'Options' - 'PLCopenXML'” on page 1198*
- *Chapter 1.4.1.20.3.4.26 “Command 'Export PLCopenXML'” on page 1015*
- *Chapter 1.4.1.20.3.4.27 “Command 'Import PLCopenXML'” on page 1015*
- *Chapter 1.4.1.20.3.4.25 “Command 'Import'” on page 1015*

1.4.1.3.2 Transferring Projects

If you wish to transfer a project to another computer and connect from there to the same PLC, without an online change or download being required, observe the following points.

- Make sure that the project requires only fixed versions of libraries (exception: interface libraries), visualization profile and compiler.
- Make sure that the boot application is up to date.

Then create a project archive, which you unpack on the other computer.

Transferring a project to another system

Requirement: A project is open on computer “PC1” that you transfer to another computer “PC2” and reconnect from there to the same controller.

1. Make sure that only libraries with fixed versions are integrated in the project, with the exception of pure interface libraries. To do this, open the *“Library Manager”* and check all entries that have a “*” instead of a fixed version specification.
2. Make sure that a fixed compiler version is set in the project settings. To check, select *“Project ➔ Project Settings”* and the *“Compiler Options”* category.
3. Make sure that a fixed visualization profile is defined in the project settings. To check, select *“Project ➔ Project Settings”* and the *“Visualization Profile”* category.

4. Make sure that the application that is presently open is the same as that which is presently in use on the PLC. This means that the “boot application” must be identical to the project in the programming system. To check, look at the project name in the title bar of the programming system window: If an asterisk is displayed behind the name, this means that the project has been modified, but not yet saved. It is then possible that the application and boot application do not correspond!

   In this case, first create a (new) boot application. It depends on the PLC and the application properties, whether this takes place automatically during the download of the application. For explicit creation, select the command *“Online ➔ Create boot application”*. Then execute a download with the help of the commands *“Online ➔ Login”* and *“Online ➔ Load”*.

   After that, start the application on the controller with the command *“Debug ➔ Start”*.

   ⇨ Now the desired application is running on the PLC, to which you wish to reconnect from the same project later on PC2.

5. Generate a project archive: Select *“File ➔ Project Archive ➔ Save/Send Archive”*. In the *“Project Archive”* dialog box, also select the following information:

   - *“Download information files”*
   - *“Library profile”*
   - *“Referenced devices”*
   - *“Referenced libraries”*
   - *“Visualization profile”*

   Save the project archive in a place that is accessible by PC2.

6. Log out from the controller: To do this, select *“Online ➔ Logout”*. You can stop and restart the PLC without reservations, before you reconnect from PC2.

7. Extract the project archive to PC2: Select *“File ➔ Project Archive ➔ Extract Archive”* and open the archive saved above. In the *“Extract Project Archive”* dialog box, activate the same information as described above when generating the archive.

8. Open the project and log in to PLC “xy” again.

   ⇨ CODESYS does not demand an online change or download; the project runs.

See also

- *Chapter 1.4.1.20.4.11.3 “Dialog Box 'Project Settings' - 'Compileoptions'” on page 1173*
- *Project Settings - Visualization Profile*
- *Chapter 1.4.1.20.3.6.4 “Command 'Create Boot Application'” on page 1032*
- *Chapter 1.4.1.20.3.1.8 “Command 'Save/Send Archive'” on page 960*

1.4.1.4 Comparing projects

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

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

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

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

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

1.4.1.4.1 Creating a comparison view

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

1. Select the command *“Project ➔ Compare”*.

   ⇨ The *“Project Comparison”* dialog box opens.

2. Enter the path to the reference project, for example the path to the last-saved version of your current project.

3. Leave the activation of the comparison option *“Ignore Spaces”* as it is.

4. Click on *“OK”*.

   ⇨ The comparison view opens. Title: *“ Project Comparison – Differences”*. The Device trees of the current project and the reference project are displayed alongside each other and the changed objects are marked in color.

5. Select an object marked in blue in the tree of the reference project (right). The current project no longer contains this object.

   Click on *“Accept Single”*

   ⇨ The object is added to the tree of the current project (left). The line has a yellow background. appears in the middle column.

6. Select an object marked in green in the tree of the current project (left). The reference project does not contain this object.

   Click on *“Accept Single”*

   ⇨ The object is removed again from the tree of the current project (left). The line has a yellow background. appears in the middle column.

7. If changes are detected in the content of an object that is contained in both the current project and the reference project, this is indicated by red lettering. You can then switch to the detailed comparison view for the object by double-clicking on the object.

8. Close the comparison view and answer the query whether the changes made are to be saved with *“Yes”*.

   ⇨ The changes become effective in the project.

1.4.1.4.2 Opening the detailed compare view

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

1. Double-click the line of the aligned POU versions.

   ⇨ The compare view switches to the detailed compare view of the POU. The modified code lines are highlighted in gray and written in red.

2. Click .
   ⇨ Code lines with changes (red) are extended by two lines: an line with insert (left, green) and a line with delete (right, blue).
3. Click again.
   ⇨ The code line is marked again as modified.
4. Move the mouse pointer to the code line marked as modified and click *"Accept Single"*.
   ⇨ The code line from the reference project is activated for acceptance into the current project.
5. Click .
   ⇨ The project compare view opens for the entire project. It is write-protected (read-only) to prevent you from activating differences for acceptance. The link highlighted in yellow above the tree view also indicates this.
6. Click the link: *"Project compare view is read only because there are uncommitted changes in another view. Click here to switch to the modified view."*
   ⇨ The detailed compare view opens again. The unconfirmed changes are highlighted in yellow.
7. Click ✕ in the tab of the view and confirm that the changes should be saved.
   ⇨ The detail project view is closed and the POU is overwritten. Now it corresponds to the POU of the reference project. The project view is active again so that you can continue working with project compare.

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

See also

- *Chapter 1.6.6.1.1.6 "Comparing projects" on page 3640*
- *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*
- *Chapter 1.4.1.4.1 "Creating a comparison view" on page 196*

1.4.1.5 Protecting and Saving Projects

General information about write and access protection

You can protect a project against unintentional changes by means of access and write protection. You can also provide it with read protection (knowledge protection).

Write protection:

The following options are available for providing the entire project with simple write protection:

- Select the *"Open Read-Only"* option when opening the project.
- You set the *"Released"* status in the *"Project Information"*.
- You select the "read-only" option in the properties of the project file in the local file system.

In order to protect only certain objects in a project against changes, or to allow access only to certain users, you can use a user and access rights management (see below). Some target devices similarly support user and rights management. The access of CODESYS to objects and files of the target device can thus be restricted.

However, write protection and access protection do not serve as protection of expertise of the POUs. Both CODESYS itself, automation platform plug-ins and persons with knowledge of the project file format can view or modify function blocks created with CODESYS.

Knowledge protection:

Knowledge protection of a project is done by encrypting the project file. Either with a project password, the CODESYS Security Key (dongle), or a certificate. We recommend protection by means of the key or the certificate because in this case no secret needs to be shared between authorized users. The desired type of project encryption is enabled in the project settings.

You can attain knowledge protection of a library by providing it as a target-system-independent "protected library" (`*.compiled-library`, `*.compiled-library-v3`). The library file no longer contains source code in this format, but only encrypted precompile context. The compiler is still able to interpret these data. Whether access by other CODESYS components or additional plug-ins is possible depends on their functionality and is to be observed in individual cases. Signing can increase protection even more.

Knowledge protection and copy protection of a boot application can be done by means of a runtime system dongle (simple or licensed) or encryption with a certificate. One of these options is enabled in the object properties of the application.

See also

- *“User management and password manager” on page 199*
- *Chapter 1.4.1.5.3 “Protecting Projects Using a Dongle” on page 203*
- *Chapter 1.4.1.5.2 “Assigning Passwords” on page 202*
- *Chapter 1.4.1.5.5 “Protecting Objects in the Project by Access Rights” on page 204*
- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*
- *Chapter 1.4.1.8.17 “Encrypting an application” on page 294*

Encryption with certificates

In CODESYS, projects and applications can be encrypted with certificates and signed in order to protect them from unauthorized access.

To do this, you can configure specific security settings for each individual user profile. These settings are always used automatically when the user works with CODESYS projects. Therefore, they do not have to be redone for each project. The general configuration of the security features for a user profile is done in the *“Security Screen”* view of CODESYS. See the individual instructions below.

You can also encrypt a project file or an application for download or online change directly with a certificate:

- User-independent encryption for the current project is configured in the *“Security”* category of the *“Project Settings”*.
- User-independent encryption of the application is configured in the *“Properties”* dialog of the application object.

NOTICE!
When you encrypt a project, an application, or online code with a certificate, you will always require the certificate with a private key in order to open the object again.

If the CODESYS Security Agent add-on product is installed, then the “Security Screen” view provides an additional tab: “Devices”. This allows for the configuration of certificates for the encrypted communication with controllers.

Certificates, **Windows Certificate Store**

All available certificates are located in the Windows Certificate Store (*“certmgr”*) on your computer. There are two types of keys:

- Certificates with private keys
  - for file decryption
  - for digital signatures
- Certificates with public keys
  - for file encryption
  - for verifying digital signatures

The local Windows Certificate Store is usually filled with certificates by the IT administrator of the computer. Certificates are either created using special tools or the creation is requested by a trusted certification authority (CA).

If you receive a certificate file that you need to install yourself in the Windows Certificate Store, then double-click the file in the store directory. Depending on the type (certificate with private or public key only), the appropriate import wizard will appear.

See also

- *Chapter 1.4.1.18.1 “General Information” on page 453*
- *Chapter 1.4.1.5.7 “Encrypting Projects with Certificates” on page 207*

User management and password manager

User accounts with different rights can be managed in CODESYS. For each account you can define the actions with which the user can access a project object.

The user management is configured in the *“Project settings”* in the category *“Users and Groups”*.

Before the creation of users and groups, please note the following:

- Rights can only be assigned to user groups. Therefore, you must assign each user to a group.
- There is automatically always a group 'Everyone' and by default every user and every other group is initially a member of this group. Thus each user account is automatically equipped with at least the defined standard rights.

  You cannot delete the group 'Everyone', you can only rename it, and you cannot remove members from this group.

  Caution: by default "Everyone" does not have the right to change the current user, group and rights configuration!
- There is automatically always a group 'Owner' containing a user 'Owner'. From V3.5 only the 'Owner' initially has the right to change the current user, group and rights configuration in a new project! Hence, only 'Owner' can assign this right to another group.

  Initially the 'Owner' can log in with user name 'Owner' and an empty password. You can add further users to the group 'Owner' or remove users from it, but at least one member must be retained. Like 'Everyone', you cannot delete the group 'Owner' and it always possesses all access rights. This prevents a project from being rendered unusable by denying all access rights to all groups.

  You can rename both the group 'Owner' and the user 'Owner'.
- If the programming system or a project is restarted, no user is initially logged in to the project. However, the user can then log in via a certain user account with user name and password in order to obtain the access rights defined for the account.
- Each project has its own user management! Therefore, in order to obtain certain access rights to a library integrated into the project, for example, the user must explicitly log in to the library project.

  Users and groups defined in different projects are not the same, even if they have the same names.
- A user management in a project only makes sense if it is connected with corresponding rights assignment for the access to project and objects. The project rights are generally managed in the dialog box *“Rights”* of the *“User Management”*. You can also change the access rights to an individual project object on the *“Access control”* tab of the *“Properties”* of the object.
- There are standard menu commands under *“Project ➔ User Management”* for logging into and out of a project as a defined user. A password manager permits the management of the login data on your computer.

From V3.5 only the 'Owner' initially has the right to change the current user, group and rights configuration in a new project! Hence, only 'Owner' can assign this right to another group.

NOTICE!
CODESYS stores the user passwords inaccessibly. If you forget a password, the user account becomes unusable. If you forget the 'Owner' password, the entire project may become unusable!

Password manager

The password manager enables you to save login data records that you enter during the login procedures for projects. It is accessible via a button in the login dialog box and offers fast access to the login data currently required. This can be helpful, for example, if you are working in parallel on several library projects that are protected by different passwords.

The password manager itself is protected by an individual master password. If you wish to use the password manager for the first time, CODESYS requests you to define this password in the password manager configuration dialog box. CODESYS notes the master password until you terminate the current CODESYS session. You must always input the password when you wish to log in to the password manager for the first time during a new session, or after you have changed it.

See also

- *Chapter 1.4.1.5.5 “Protecting Objects in the Project by Access Rights” on page 204*
- *Chapter 1.4.1.5.6 “Logging in via User Account and Password Manager” on page 205*
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

Rights management

Rights management for access to a project and objects in a project is necessary in order to make a user management meaningful.

The rights for a project are generally managed in the *“Rights”* editor of the *“User Management”*. You can also change the access rights to an individual project object on the *“Access control”* tab of the *“Properties”* dialog box of the object.

Before assigning rights, please observe the following:

- In a new project CODESYS always sets all rights for the execution of actions on objects with the **default value** 'allowed' (standard right). The only exception to this is the right to change the current user, group and rights configuration. Initially only the 'Owner' group has this right.
- If you are member of a group that is permitted to change rights, you can do this at any time for each right when working further on a project. You change a right by switching between 'allowed' and 'forbidden' or by resetting to the default.

See also

- *Chapter 1.4.1.5.4 “Setting up a user management” on page 203*
- *Chapter 1.4.1.5.5 “Protecting Objects in the Project by Access Rights” on page 204*

Filing, saving

Provide the project file with the desired protection before saving it in the file system; see above. For a read-only project file you are given various options so that you can still save the file, depending on the type of write protection.

If the project is to be opened later in an older CODESYS version, it makes sense to save the project for precisely this version (file type), since CODESYS will also inform you immediately about possible losses of data in the course of saving it.

If you wish to save library projects, please observe the rules for the creation of libraries. Also consider the possibility of installing a library directly in a library repository.

If you wish to continue to use a project on another computer, it makes sense not only to save the project file, but also to create a project archive from all relevant auxiliary files.

You can make a setting so that a backup copy of this project is created each time the project is saved. In addition you can configure CODESYS so that projects are generally automatically saved at certain time intervals.

If you wish to keep projects in a source control system, observe the corresponding add-ons for CODESYS. For example, the link to SVN is supported.

See also

- *Chapter 1.4.1.20.4.13.16 “Dialog 'Options' – 'Load and Save'” on page 1196*
- *Chapter 1.4.1.3 “Exporting and Transferring Projects” on page 193*
- *Chapter 1.4.1.5.8 “Saving the Project” on page 209*
- *Chapter 1.4.1.5.9 “Saving/Sending the project archive” on page 210*
- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*
- *Chapter 1.4.1.5.10 “Linking a project to the source control system” on page 211*

1.4.1.5.1 Setting up write protection

A project can be protected against inadvertent changes by means of access and write protection. In addition, however, it can also be provided with read protection (know-how protection). You have the following options:

Open the project with write protection

Requirement: No project is opened.

1. Select *“File ➔ Open Project”*.
   ⇨ The dialog box *“Open project”* appears.
2. Select the project.
3. Click on the arrow button ▾ next to the *“Open”* button and select *“Open read-only”* from the menu.
   ⇨ CODESYS opens the project. At the top right in the main window a line appears *“'Project file cannot be saved…'”*. You must now select one of the offered options if you wish to save the project file.

See also

- *Chapter 1.4.1.2.1 “Opening a V3 Project” on page 186*

Providing projects with the attribute 'Released'

Requirement: project is opened.

1. Select *“Project ➔ Project Information”*, then the *“Summary”* tab.
2. Activate the option *“Released”*, confirm with *“OK”*.
3. Save the project, for example with *[Ctrl]+[S]*.
4. Open the project again with the command *“File ➔ Open Project”*.
   ⇨ CODESYS opens the project. At the top right in the main window a line appears *“'Project file cannot be saved…'”*. You can now directly remove the status *“Released”* again via the offered option if you wish to save the project file.

See also

- *Chapter 1.4.1.20.2.21 “Object 'Project Information'” on page 919*

Providing a project in the file system with the property 'Read-only'

▷ Provide the project file in its local file system with the property attribute 'Read-only'.

⇨ If you had already opened the project and you now attempt to save it under the same name, a dialog box appears informing you about the existent write protection. This dialog box provides you with the following options:

You can save the project under another name or another path using the button *“Save As…”*.

You can deliberately save the project under the same name and path and thus overwrite the existing version in the file system using the button *“Overwrite”*.

You can abort the saving procedure using the *“Cancel”* button, for example to remove the write protection on the disk.

If you re-open the project, a line appears at the top right in the main window `'The project cannot not be saved…'`. You must now select one of the offered options if you wish to save the project file.

See also

- ↳ *“General information about write and access protection” on page 197*

1.4.1.5.2 Assigning Passwords

Requirement: The project is open.

1. Click *“Project ➔ Project Settings”* and then select the *“Security”* category.
   ⇨ The dialog *“Project Settings / Security”* opens.
2. Select the *“Encryption”* option.
   ⇨ The option fields *“Password”*, *“Dongle”*, and *“Certificates”* are selectable.
3. Select the option *“Password”*.
   ⇨ The input fields for the encryption password appear.
4. Enter the encryption password in the input field *“New Password”*.
5. Enter the encryption password for confirmation in the input field *“Confirm new password”*.
6. Click *“OK”*.
   ⇨ CODESYS saves the encryption password for the project. You must enter this password in order to be able to open the project again, even if it is to be loaded as a library reference.

CAUTION!
If you no longer know the encryption password, you can no longer open or restore the project!

See also

- ↳ *Chapter 1.4.1.20.4.11.7 “Dialog 'Project Settings' - 'Security'” on page 1176*
- ↳ *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*

1.4.1.5.3 Protecting Projects Using a Dongle

Requirement: The project is opened and you have connected the CODESYS Security Key (dongle) to your computer.

1. Click *“Project ➔ Project Settings”* and then select the *“Security”* category.
   ⇨ The dialog *“Project Settings / Security”* opens.
2. Select the *“Encryption”* option.
   ⇨ The option fields *“Password”*, *“Dongle”*, and *“Certificates”* are selectable.
3. Select the option *“Dongle”*.
   ⇨ The dialog with the drop-down list *“Registered Dongles”* and the buttons *“Add”*, *“Remove”*, *“Comment”* and *“Flash”* opens.
4. Click *“Add”*.
   ⇨ The *“Add Registered Dongle”* dialog opens.
5. Select the CODESYS Security Key (dongle) from the *“Dongle”* drop-down list and optionally enter a comment.
6. Click *“OK”*.
   ⇨ The added dongle is listed in the list *“Registered Dongles”*.
7. Click *“OK”*.
   ⇨ The dongle is registered for the project. You must connect the dongle to your computer in order to be able to open the project again, even if it is to be loaded as a library reference.

NOTICE!
If the CODESYS Security Key registered for the project is lost, you can no longer open the project or restore it.

See also

- *Chapter 1.4.1.20.4.11.7 “Dialog 'Project Settings' - 'Security'” on page 1176*
- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*

1.4.1.5.4 Setting up a user management

This concerns a user management for a CODESYS project file. Visualizations and devices can have their own user management.

The following guide describes how you can adapt the user management for the first time in a project. It deals with the definition of a user and a group to which he belongs.

Requirement: the project for which the user management is to be set up is opened. There is no adapted user configuration yet.

1. Select *“Project Settings ➔ Users and Groups”* and then the *“Users”* tab. The user `Owner` is already created by default.
2. Click on *“Add”*.
   ⇨ The dialog box *“Add User”* appears.

3. Enter a login name, for example 'Dev1', and a password. Leave the option *"Activated"* activated. Click on *"OK"*.

   ⇨ On creating a group for the first time, CODESYS now requests you to authenticate yourself to perform this action.

   In this case, enter 'Owner' as the *"current user"*. Do not enter a *"password"*, just click on *"OK"*.

   The user `Dev1` appears in the list and is automatically a `member of the group 'Everyone'.`

4. Change to the tab *"Groups"*, in order to add the user to a new group.

   ⇨ The groups `Everyone` and `Owner` have already been created.

5. Click on *"Add"* in order to open the dialog box *"Add Group"*.

6. Specify at least one name for the new group, for example 'Developers'. Activate the checkbox next to the entry *"User 'Dev1'"* in the field *"Members"*. Click on *"OK"*.

   ⇨ The group *"Developers"* now appears with `has user member 'Dev1'.`

7. Switch to the *"Users"* tab.

   ⇨ The user *"Dev1"* now appears as a member of the groups 'Everyone' and 'Developers'.

You can take over the user management configuration from another project by using the "Export/Import" functions in the dialog box "Project Settings", category "Users and Groups".

See also

- *Chapter 1.4.1.5 "Protecting and Saving Projects" on page 197*
- *"User management and password manager" on page 199*
- *Chapter 1.4.1.20.4.11.2 "Dialog 'Project Settings' - 'Users and Groups'" on page 1172*
- *Chapter 1.4.1.10.3 "Handling of Device User Management" on page 385*

1.4.1.5.5 Protecting Objects in the Project by Access Rights

Protection of individual objects by setting access rights in the *"Rights"* editor

1. Select *"Project ➔ User Management ➔ Rights"*

   ⇨ The window of the *"Rights"* editor opens. On the left you can see the action categories, on the right the currently existing user groups.

2. Expand the relevant action category and below it the action for which you wish to change a right.

3. Select the goal of the action in the *"Actions"* window. In the *"Rights"* window, select the group for which you would like to change the right. Multiple selection is possible.

   ⇨ The buttons in the symbol bar are active.

4. Click on the appropriate button in order to change the right of the group for the action on the target object.

   ⇨ CODESYS updates the symbol in front of the group according to the new right. The right is immediately effective.

See also

- ↳ *Chapter 1.4.1.20.4.6 “Dialog 'Permissions'” on page 1152*

Protection of individual objects by setting access rights in the object properties

Here you can configure whether the members of a group have the right to view, edit or remove the object and to add/remove child objects to/from the object.

1. Select the object in the navigator tree.
2. In the context menu, select the command *“Properties”* and in the dialog box select the category *“Access Control”*.
3. In the table under *“Groups, Actions and Permissions”*, double-click on the symbol of the right that you wish to change.
   ⇨ A selection list of the possible rights appears: *“Grant”*, *“Deny”*, *“Clear”*.
4. Select the desired right and click on *“Accept”* or *“OK”*.
   ⇨ The right is immediately effective for the action and group. The symbol changes accordingly.

See also

- ↳ *Chapter 1.4.1.20.4.10.6 “Dialog 'Properties' - 'Access Control'” on page 1161*

1.4.1.5.6 Logging in via User Account and Password Manager

Logging in to a project without using the password manager functions

Requirement: A project is open. You wish to log in as a defined user for this project or for a library integrated in it in order to edit one or the other with certain rights. You have the required login data for the respective project or the library.

1. Select *“Project ➔ User Management ➔ User Logon”*.
   ⇨ The dialog box *“Logon”* opens.
2. Select the project file from *“Project/Library”* and enter the required access data *“User name”* and *“Password”*.
3. Log in with *“OK”*.
   ⇨ If another user is already logged in, this user will automatically be logged out by the new login.

Setting a master password for the password manager

Requirement: A project is open. The dialog box *“Login”* is open for you to log in as a defined user for a project or for a library integrated in the project. You wish to use the password manager in order to save login data in it.

1. Select *“Project ➔ User Management ➔ User Logon”*.
2. In the dialog box *“Logon”*, click on the button .
   ⇨ If you are working for the first time with the password manager, the dialog box *“Password Manager Configuration”* opens.

3. Enter a character string as the future master password. Confirm it in the second line and click on *“OK”*.

   ⇨ CODESYS notes the master password until you terminate the current CODESYS session. You must always input this password when you wish to log in to the password manager for the first time during a new session, or after you have changed it.

> **NOTICE!**
> If you have forgotten your master password, you no longer have any possibility to access the login data already saved! In this case you can only reset the password manager. After that you must start again to save passwords in the manager!

Saving login data in the password manager

Requirement: A project is open. You wish to log in as a defined user for this project or for a library integrated in it in order to edit one or the other with certain rights. You have the required login data for the respective project or the library. These login data have not yet been saved in the password manager.

1. Select *“Project ➔ User Management ➔ User Logon”*, in order to open the *“Logon”* dialog box.
2. Select the project file from *“Project/Library”*.
3. Enter the user name and password for the project or the library.
4. Click on the button .

   ⇨ If you are working for the first time with the password manager, you will be requested to define a master password. Refer to the above guide 'Setting a master password for the password manager' for this.

   When you call the password manager for the first time in this CODESYS session, you will be requested to enter the master password.
5. Enter the master password when requested to do so.

   ⇨ The password manager menu appears.
6. Select the option *“Save the credentials locally on this computer”*.

   ⇨ The login takes place. The data are saved in the password manager.

Getting the login data from the password manager

Requirement: A project is open. You wish to log in as a defined user for this project or for a library integrated in it in order to edit one or the other with certain rights. The login data required for this are already saved in the password manager.

1. Select *“Project ➔ User Management ➔ User Logon”* in order to open the *“Logon”* dialog box.
2. Click on the button .

   ⇨ If you are working for the first time with the password manager, you will be requested to define a master password. Refer to the above guide 'Setting a master password for the password manager' for this.

   When you call the password manager for the first time in this CODESYS session, you will be requested to enter the master password.
3. Enter the master password when requested to do so.

   ⇨ The password manager menu appears.
4. Select the appropriate entry *“Use the stored credentials for <user name>”*.

   ⇨ The login takes place automatically with the data read from the password manager.

Opening the password manager, changing the master password

Requirement: A project is open. You wish to open the password manager in order to view and/or edit the entries or to change the master password. You have already logged in once with the master password.

1. Select *"Project ➔ User Management ➔ User Logon"*, in order to open the *"Logon"* dialog box.
2. Click on the button .

   Select *"Open the Password Manager"*.

   ⇨ The password manager window opens.
3. Click on *"Change Master Password"* and make the change.

Logging out from the project

Requirement: A project is open. A user is logged in, which is recognizable by a name entry in the field *"Current User"* in the status bar.

▷ Select *"Project ➔ User Management ➔ User Logoff"*. Alternatively, double-click on the field *"Current User"* in the status bar.

⇨ If the user is logged in to only one project, he will now be logged out without further interaction. *"(nobody)"* appears again in the field *"Current User"* in the status bar

If the user is logged in to several projects, the dialog box *"Logoff"* opens. There, select the specific project or library project from which the user is to be logged out.

See also

- *"User management and password manager" on page 199*
- *Chapter 1.4.1.20.3.4.28 "Command 'User management' – 'Log in User'" on page 1016*
- *Chapter 1.4.1.5.4 "Setting up a user management" on page 203*

1.4.1.5.7 Encrypting Projects with Certificates

Configuring a certificate for project file encryption in a user profile

When a project is encrypted with a certificate, this certificate is needed for decryption to open the project. You can assign this certificate to specific user profiles. To do this, select the certificate from the Windows Certificate Store on the *"User"* tab of the *"Security Screen"*.

1. Double-click in the status bar or click *"View ➔ Security Screen"*.

   ⇨ The *"Security Screen"* view opens.
2. In the *"User"* tab, select the user profile for which the communication will be encrypted. By default, the specified user profile is the one you have used on your computer to sign into Windows. You can also create a new user profile with .
3. Click the button in the *"Project file decryption"* area.

   ⇨ The *"Certificate Selection"* dialog opens.
4. Select a certificate with a private key from the list *"Available certificates in the local Windows Certificate Store"*. Certificates with a private key are identified by the symbol.
5. Click .
6. The certificate is added to the upper part of the dialog.
7. Click *"OK"* to confirm your selection.

   ⇨ The selected certificate is displayed in the *"Security Screen"* in the *"Project file decryption"* area.

Encrypting a project with a certificate

A project encrypted with a certificate in connection with a user management allows you to restrict access to the project.

1. Click *"Project ➔ Project Settings"* and then select the *"Security"* category.
   ⇨ The *"Project Settings / Security"* dialog opens.
2. Select the *"Encryption"* option.
   ⇨ The option fields *"Password"*, *"Dongle"*, and *"Certificates"* are available.
3. Select the *"Encryption"* option.
   ⇨ The certificates available for project encryption are listed in the lower part of the dialog. If no certificate has been specified yet, then click to select a relevant certificate in the *"Certificate Selection"* dialog. Then return to the *"Project Settings"* dialog. Now the certificate is specified for encryption. Now the project can only be edited on computers of users who also have the certificate for file decryption.

Deleting a certificate in the user profile

You delete the certificate in the *"Security Screen"* view, either directly on the *"User"* tab or in the *"Certificate Selection"* dialog. The deletion will follow in the other dialog.

- Dialog *"Security Screen"*, tab *"User"*, *"Digital Signature"*, or *"Project Data Decryption"*: Select a certificate and click
- Dialog *"Certificate Selection"*: in the *"Security Screen"* dialog, click on the *"User"* tab. In the upper field of the *"Certificate Selection"* dialog, select the certificate to be deleted and click .

Configuring a certificate for the digital signature in a user profile

To ensure that the project is not only encrypted with a certificate, but also that its authorship and integrity can be verified, you can add a signature to the project:

1. Double-click in the status bar or click *"View ➔ Security Screen"*.
   ⇨ The *"Security Screen"* view opens.
2. In the *"User"* tab, select the user profile for which the digital signature will be created. By default, the specified user profile is the one you have used on your computer to sign into Windows. You can also create a new user profile with .
3. Click the button in the *"Digital signature"* area.
   ⇨ The *"Certificate Selection"* dialog opens.
4. Select a certificate with a private key from the list *"Available certificates in the local Windows Certificate Store"*. Certificates with a private key are identified by the symbol.
5. Click .
   ⇨ The certificate is added to the upper part of the dialog.
6. Click *"OK"* to confirm your selection.
   ⇨ The selected certificate is displayed in the *"Security Screen"* in the *"Digital signature"* area.

See also

- *"Encryption with certificates" on page 198*
- *Chapter 1.4.1.20.3.3.18 "Command 'Security Screen'" on page 995*
- *Chapter 1.4.1.20.4.18 "Dialog 'Certificate Selection'" on page 1215*
- *Chapter 1.4.1.20.4.10.3 "Dialog 'Properties' - 'Encryption'" on page 1158*
- *Chapter 1.4.1.20.4.11.7 "Dialog 'Project Settings' - 'Security'" on page 1176*
- *Chapter 1.4.1.8.17 "Encrypting an application" on page 294*

1.4.1.5.8 Saving the Project

Saving a project under the same name

Requirement: The project is open. The project file is not write-protected.

▷ Select *“File ➔ Save”*.

⇨ CODESYS saves the project file with the current project name, which appears in the title bar of the main window. If the project has been changed since it was last saved, then the project name is provided with an asterisk. If this is set in the CODESYS options in the category *“Load and Save”*, then a backup copy will also be made.

Saving a project under a different name or format

Requirement: The project is open.

1. Select *“File ➔ Save Project as”*.

   ⇨ The *“Save Project”* dialog box opens.

2. Select a storage location in the file system and the desired *“File Type”* (project file or library file) and the desired storage version. If you want to open the project later in an older version, then it makes sense to save for precisely this version, as you will then be informed immediately in the message window about possible data loss.

   ⇨ If the project file is not write protected, then CODESYS saves it in the selected path. Otherwise you will be informed how to proceed.

3. If the current project contains add-ons that are not available in the selected memory format, then the *“Extend Profile”* dialog box opens.
4. Select the add-ons to extend the memory profile in order for the add-on data to be saved.
5. To save the memory profile permanently, click *“Save Profile”* and specify a name in the *“Enter profile name”* dialog box.
6. In the *“Extend Profile”* dialog box, select the *“Use saved profile”* option and click *“Yes”*.

   ⇨ CODESYS saves the project with the saved profile.

Saving a read-only project

Requirement: A read-only project is open.

▷ Select *“File ➔ Save”*.

If the write protection was assigned in CODESYS, then it will be displayed by a line in the top right corner of the main window. Depending on the current situation you will be offered one or more of the following actions so that you can still save the project:

- *“Save project under a different file name on the disk”*: Always appears and continues to the *“Save File”* dialog box, as for the *“Save File as”* command
- *“Exit read-only mode”*: Appears if the *“Open read-only”* option is selected when opening the project.
- *“Remove read-only attribute from the project on the disk”*: Appears if the project file was provided with the 'Read-only' property in the local file system at the time of opening.
- *“Remove identification 'Released' in the project information”*: Appears only if this attribute is currently set.

If the write protection was assigned outside of CODESYS in the properties of the project file in the file system, you will be offered the following options when you attempt to save under the same name and path:

- *“Save as”*: You can save under a different name as with the *“Save Project as”* command.
- *“Overwrite”*: The write protection is removed from the project file and the file is saved under its existing name.

1. Click on the line in the top right corner of the main window that indicates the write protection.
   ⇨ The current options with which you can still save the project appear in a selection menu.
2. Select one of the options offered and perform any necessary actions.
3. Click *"File ➔ Save"* or *"File ➔ Save as"*.
   ⇨ The project can be saved.

Saving of the project automatically; creating a backup copy

Requirement: The project is open.

1. Click *"Tools ➔ Options"* (category *"Load and Save"*.
   ⇨ The *"Load and Save"* dialog box opens.
2. Activate the *"Create backup files"* option.
3. Activate the *"Automatically save every … minutes"* option and select a time interval.
4. Click *"OK"* to close the *"Options"* dialog box.
   ⇨ Each time the project is saved, CODESYS also creates a backup copy `<project name>.backup`.

   CODESYS saves the project automatically at the specified time interval to a file `<project name>.autosave` in the project directory. If you open the project again after the development system was closed irregularly, then this file will be offered to you as an alternative to the file last saved by the user.

See also

- *Chapter 1.4.1.5 "Protecting and Saving Projects" on page 197*
- *Chapter 1.4.1.20.4.13.16 "Dialog 'Options' – 'Load and Save'" on page 1196*

1.4.1.5.9 Saving/Sending the project archive

You can configure a project archive and then save it in the file system or send it directly in an e-mail.

To send, follow the guide below as far as point 9. There, click on the button *"Send"* instead of *"Save"* in order to directly open the standard e-mail program, in which a new mail will automatically be created with the project archive file as an attachment.

Requirement: A project is opened.

1. Select *"File ➔ Project Archive ➔ Save/Send Archive"*.
   ⇨ The dialog box *"Project Archive"* appears.
2. Activate the checkbox next to each object that is to be saved in the archive.

In order to guarantee know-how protection,CODESYS will not automatically add unprotected libraries, not available as "compiled-library", to a project archive. If you explicitly select such a library in the list of additional files, you will get an appropriate warning.

3. If you want to pack further files in the archive, click on *"Additional Files"*.
   ⇨ The dialog box *"Additional Files"* opens.
4. Click on *"Add"*.

5. Select the files and click *“Open”*.
   ⇨ The files are added to the list of additional files.
6. Click on *“OK”*.
7. Click on *“Comment”*.
   ⇨ The dialog box *“Comment”* opens.
8. Enter a comment and click on *“OK”*.
9. Click on the button *“Save”*.
10. Select a storage location and a file name and click on *“Save”*.
    ⇨ The project archive is saved in the file directory.

See also

- *Chapter 1.4.1.20.3.1.4 “Command 'Save project'” on page 957*
- *Chapter 1.4.1.20.3.1.5 “Command 'Save Project as'” on page 958*

1.4.1.5.10 Linking a project to the source control system

To link your CODESYS projects to a source control system, check the following option:

The Professional Version Control add-on provides the capability of directly linking to an SVN database. You can get the package at the CODESYS Store and install it with the help of the Package Manager.

Refer to the corresponding help when using Professional Version Control.

1.4.1.6 Localizing projects

You can display your project in different languages when you create and link localization files. The localization files correspond to those of the `GNU gettext` system. The format of the localization template files is *.pot (Portable Object Template), from which localization files `*.po` (Portable Object) are generated after translation.

The project can be localized in different languages. However, editing is possible only in the original version.

You configure which categories of text information are localized in the project. Then you export these texts into a translation template. This template is a file in `*.pot` format (example: `project_1.pot`). You produce localization files in the format `*.po` (example: `de.po`, `en.po`, or `es.po`), either automatically with a corresponding external translation tool or manually with a neutral text editor. You can import the `*.po` files back into CODESYS and use them for localization.

The commands for using project localization are located in the menu *“Project ➔ Project localization”*.

See also

- *Chapter 1.4.1.20.3.4.16 “Command 'Project Localization' - 'Create Localization Template'” on page 1007*

Generating localization templates

Requirement: A project is open.

1. Click *"Project ➔ Project Localization ➔ Create Localization Template"*.
   ⇨ The *"Create Localization Template"* dialog box opens.
2. Activate the categories of text information that should be included in the localization template.
3. *"Position information"* can also be included in the template. For each text to be translated, specify its location in the project. Select the positions to be displayed in the translation template: only the first position found, all positions found, or none.
4. Click the *"Generate"* button.
   ⇨ The dialog box opens for saving a `*.pot` file to the file system. Save the localization template. Then you can process the file in a translation tool and generate localization files `<language>.po` in the required languages.

Format of the localization template: file `*.pot`

In the first line, the text categories are specified that were selected for the translation when generating the template:

Example: `#: Content:Comments|Identifiers|Names|Strings`: All four categories were selected.

Then each text to be translated is segmented in the form as in the following example:

Example

```
#: D:\Projects\p1.project\Project_Settings:1
msgid "Project Settings"
msgstr ""
```

Line 1: Position information displayed as source code reference. Displayed only if this has been configured when generating the translation file.

Line 2: Untranslated text as entry `msgid` (example: `msgid "Project settings"`).

Line 3: Placeholder for the translation: `msgstr ""`. Between the single straight quotation marks, the translation in the `*.po` file must be inserted in the respective language.

Format of the localization file: `*-<language>.po`

You can generate a `*.po` file with a translation tool or create one using a neutral text editor based on the `*.pot` file. For this purpose, you could change the file extension from `*.pot` to `*.po` and edit the according to `*.po` standard format.

It is imperative to specify the language in the form of the usual culture abbreviation in the metadata of the file (example: `"Language: de"` for German. Then you insert the translations of the individual texts between the straight quotation marks for the `msgstr ""` entries.

Example

```
"Language: de\n"
#: Content:Names
#: D:\projects\p1.project\Project_Settings:1
msgid "Project Settings"
msgstr "Projekteinstellungen"
```

Importing localization files / localizing projects

Requirement: For your project, localization files (`<language>.po`) were generated based on the translation template `*.pot`. The project is open.

1. Click *"Project ➔ Project Localization ➔ Manage Localizations"*.
2. Click on the *"Add"* button.
   ⇨ The *"Open Localization File"* dialog box appears for selecting a `*.po` file from the file system.

3. Select one of the localization files (example: `<project name>-de.po`).
   ⇨ The dialog box closes and the affected texts appear in the project in the respective language. For example, if you specify the translation `msgstr "Main program"` for the POU name "PLC_PRG" in the English localization file, then the object name "Main program" appears in the device tree.
4. In the same way, you import the localization files for other language targets.

Switching localization, adding and removing localization files

Requirement: All required language are stored in the project by importing the corresponding `*.po` file. The project is open.

1. Click *"Project ➔ Project Localization ➔ Manage Localization"*.
   ⇨ The *"Manage Localization"* dialog box opens. All stored localization files `*-<language>.po` appear in *"Files"*, as well as the entry *"<original version>"*.
2. Select the desired language and click the *"Switch Localization"* button.
   ⇨ The project appears in the selected language. When you select *"<original version>"*, the project is displayed in the original, unlocalized version and it cannot be edited.

Optional: Defining a default localization, toggling localizations

▷ Select one of the available localizations and activate the *"Default Localization"* option.
   ⇨ Click *"Project ➔ Project Localization ➔ Toggle Localization"* to toggle the localization between the default localization and original version. By default, this command is also available with the button on the toolbar.

See also

- *Chapter 1.4.1.20.3.4.17 "Command 'Project Localization' - 'Manage Localizations'" on page 1008*
- *Chapter 1.4.1.20.3.4.18 "Command 'Project Localization' - 'Toggle Localization'" on page 1009*

1.4.1.7 Configuring I/O Links

With the help of device objects you can map hardware to be controlled in a tree structure in your CODESYS project. This makes the linking of hardware and application easy to handle.

In the configuration editors of the device objects, you can configure the settings for the communication between CODESYS and the controller, and above all for I/O mapping. The I/O mapping is the linking of the inputs and outputs of the controller with the variables of your application.

Access to control objects at runtime can be controlled, depending on the device, via an 'online user management', which you can edit – likewise depending on the device – in the CODESYS Development System. Moreover, communication with the controller depends on the current security settings.

See also

- *Chapter 1.4.1.7.1 "Configuring Devices and I/O Mapping" on page 213*
- *Chapter 1.4.1.10.2 "Encrypting Communication, Changing Security Settings" on page 381*
- *Chapter 1.4.1.10.3 "Handling of Device User Management" on page 385*

1.4.1.7.1 Configuring Devices and I/O Mapping

Configuring devices

You can configure the device objects inserted into the device tree in the associated device editor. The possibilities depend on the device description. The 'generic device editor' provides tabs that are supplemented as necessary by device-specific tabs.

Requirement: You have opened a standard project in whose device tree a standard PLC and below that a fieldbus device object are inserted.

1. Double-click the device object of the standard PLC in the device tree of your project.
   ⇨ The *“<device name>”* editor opens in the CODESYS main window. The *“Communication Settings”* tab is in the foreground. Change to the other tabs in order to make configuration settings for the controller. See the help pages for the generic device editor.
2. Double-click the fieldbus device object in the device tree of your project.
   ⇨ The *“<fieldbus device name>”* editor opens in the CODESYS main window. Specific tabs are available depending on the device. For the configuration options, see the help pages for the respective device editor. If the *“Show generic device configuration views”* option is selected in *“Tools ➔ Options”*, in the *“Device Editor”* category, then see also the tabs contributed by the generic device editor.

See also

- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*

General information about I/O mapping

Whether or not you can configure an I/O mapping to project variables or even to the entire function blocks depends on the type of device. Configuring an I/O map means linking input and output channels of the device with variables of the project. We also use the term **'mapping'** for this.

Pay attention in general to the following for the mapping of inputs and outputs of a device to variables in CODESYS:

- You do not have write access to variables that are mapped to an input.
- You can map an existing variable to one input only.
- You can directly generate new global implicit variables in the I/O map and map them to a device channel.
- The memory layout of structures is specified by the device.
- You can change addresses and fix values in the I/O map.
- For each variable that is assigned to an I/O channel in the *“I/O Mapping”* dialog, you can cause 'force variables' to be generated during the compilation of the application (see further below). Using these variables you can, for example during the commissioning of a plant, force a value on the input or output via a visualization/HMI.
- Changes in the I/O map can be transferred to the controller with an online change.
- If a pointer to a device input is used, the access is considered to be a write access, for example `pTest := ADR(input);`. This leads to a compiler warning when the code is generated: "`...invalid assignment target`". If you require a construct of this kind, you have to first copy the input value `input` to a variable with write access.
- An I/O address can also be linked with a variable via the 'AT declaration' in the IEC code. Since a device configuration often changes again, however, we recommend that you make the assignments only in the device editor.

  If you use the AT declaration, note the following:
  – An AT declaration is permissible only with local or global variables, not with input or output variables of function blocks.
  – Implicit 'force variables' for I/Os (see below) cannot be generated for AT declarations.
  – If you use an AT declaration with structure variables or function block variables, all instances will access the same memory location. This then corresponds to the use of 'static variables' in classic programming languages such as 'C'.

NOTICE!

If a pointer to a device input is used, then the access (for example, `pTest := ADR(input);`) applies as write access. This leads to a compiler warning when the code is generated: "`...invalid assignment target`".

If you require a construct of this kind, you have to first copy the input value (`input`) to a variable with write access.

As an alternative, you can assign a variable to an address in the programming code using the AT declaration. In view of possible changes of the device configuration, however, we recommend that you make the assignments only in the device editor.

You can export the I/O mapping configuration of a device to a `csv` file or import it from such a file.

See also

- *Chapter 1.4.1.20.2.8.11 "Tab '<device name> I/O Mapping'" on page 854*
- *Chapter 1.4.1.20.3.4.37 "Command 'Export Mappings to CSV'" on page 1019*
- *"Generating implicit variables for the forcing of I/Os" on page 221*

Linking a device input with an existing project variable ("mapping")

Requirement: A device that supports an I/O mapping configuration in CODESYS is inserted in the device tree of your project. On the *"I/O Mapping"* tab in the device editor you thus get a tabular display of the input and output channels of the device with specification of the addresses and data types.

NOTICE!

Mapping 'too large' data types

If a variable of a data type that is larger than a byte is mapped to a byte address, the value of the variable will be truncated to byte size there. For monitoring the variable value in the *"I/O Mapping"* dialog, this means that, in the root element of the address, the value is displayed which the variable currently has in the project. The current individual bit values of the byte are displayed in succession in the bit elements below that, but this may not be sufficient for the entire variable value.

If a UNION is represented by I/O channels in the mapping dialog, it depends on the device whether mapping to the root element is also possible.

1. In a POU, declare, for example, a variable `xBool4` of the type `BOOL` with which you want to access an input of the target device from the application.
2. To open the device editor, double-click the device object in the device tree, and then the *"<device name> I/O Mapping"* tab.
3. Observe the *"Variable"* column with the display of the device input channels and device output channels, which can still be sorted by organizational nodes, depending on the device. We assume that there is a device input of the type `BYTE`. It is displayed with its individual bit addresses (bit channels) below the `BYTE` node.

4. Note: When mapping structured variables, the editor prevents you from entering both the structure variable (example: `%QB0`) and individual structure elements (example: `%QB0.1` and `QB0.2`). Therefore, if there is a main entry with a subtree of bit channel entries in the mapping table, then the following applies: Then you can specify a variable either into the line of the main entry, or into the lines of the subelements (bit channels), but not into both.

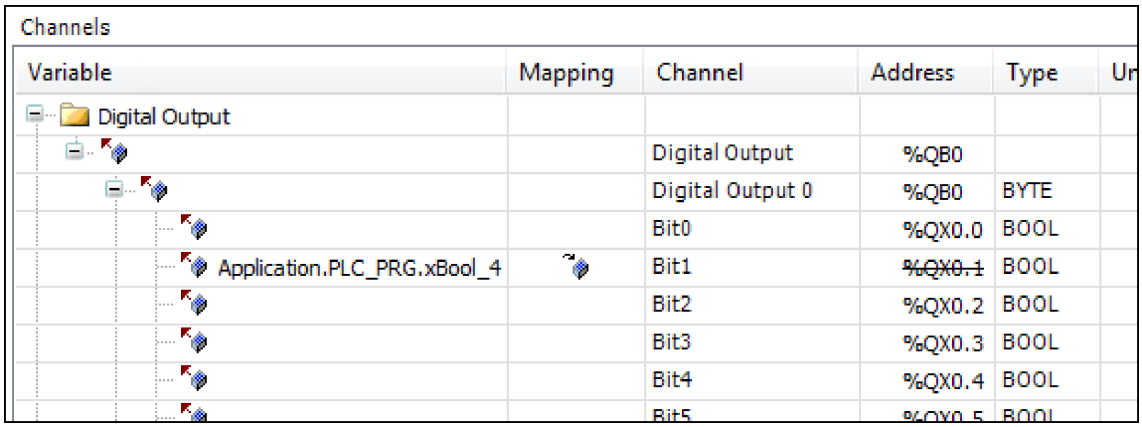
   You can now occupy either the entire channel with a variable of a suitable type OR its individual bit-channel addresses with suitable variables of the type `BOOL` or `BIT`. First of all, double-click a bit input channel in the *"Variables"* column.

   ⇨ An input field opens.

5. In order to place an existing variable on the channel, you have to enter the desired project variable with the complete path. Press ... to open the Input Assistant. Select, for example, the variable `Application.PLC_PRG.xBool4` declared in `PLC_PRG`.

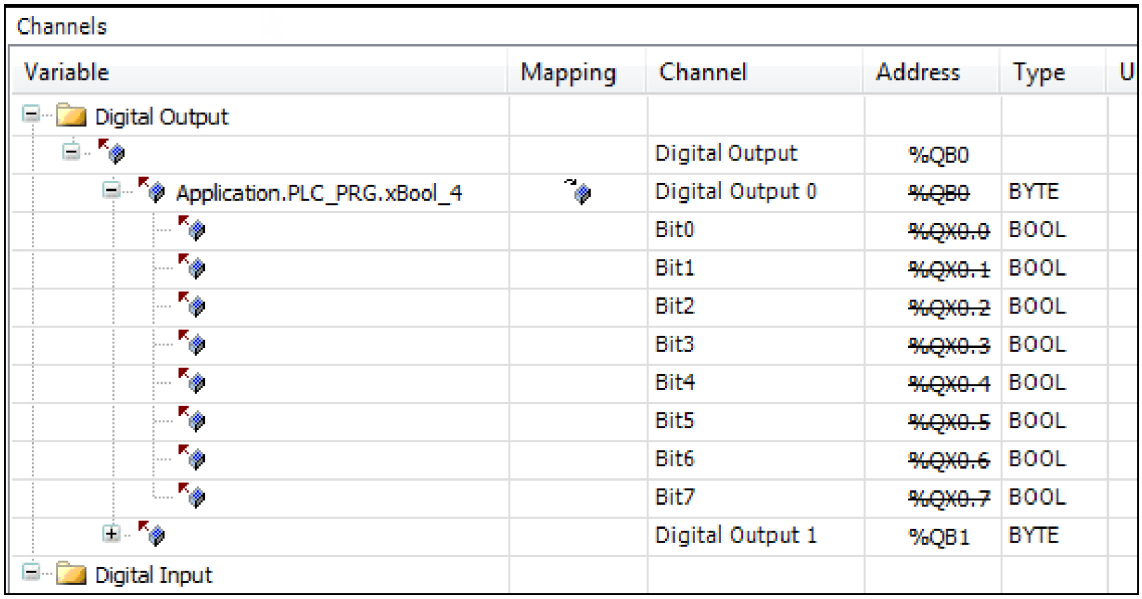
   ⇨ The variable is inserted. The HMI symbol ( ) is displayed in the *" Mapping"* column. The address is now struck through. That does not mean that the address is no longer available, because values of existing variables are managed at another memory space. But: in order to avoid ambiguities when writing the values, you should nevertheless not occupy the address with a further variable, especially in the case of outputs.

   Note: For compiler version V3.5 SP11 and higher, the initialization value of the variables is used automatically as the default value when mapping to an existing variable. You can edit the *"Default value"* field only if you map to a new created variable or if no mapping is specified. In older versions, users had to specify explicitly that the default value and initialization value were identical.



6. Delete the variable assignment again. Click the root of the channel, the `BYTE` node. Use the Input Assistant again to select the variable `Application.PLC_PRG.byte_gotodevice`.

   ⇨ The variable is inserted, all bit addresses of the main channel are struck through and you should not additionally occupy them.



See also

- *Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*

Mapping a device input to a recently created project variable

In the following you will map a device output to a global implicit variable, which you recently create for this purpose directly in the *“I/O Mapping”* dialog.

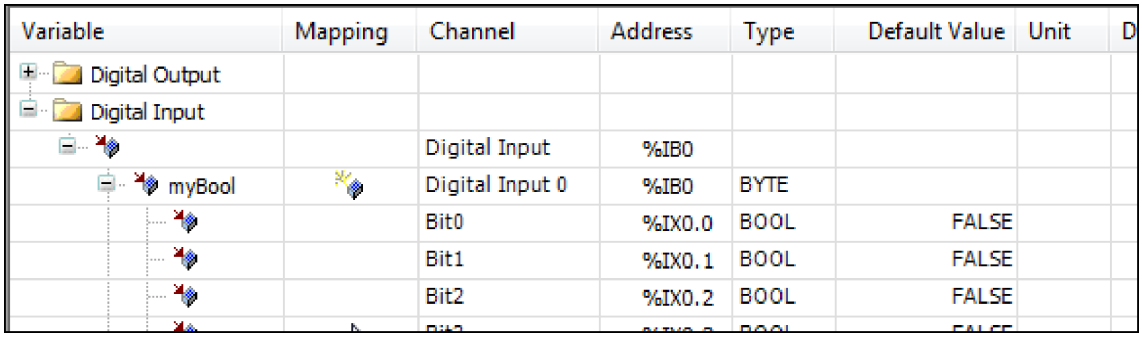
> *The “I/O Mapping” dialog is thus a further place for declaring a global variable.*

Requirement: A device that supports an I/O mapping configuration in CODESYS is inserted in the device tree of your project. On the *“I/O Mapping”* tab in the device editor you will thus see a tabular display of the input and output channels of the device with specification of the addresses and data types.

1. To open the device editor, double-click the device object in the device tree, and then the *“<device name> I/O Mapping”* tab.
2. Click in the mapping table on a channel entry in the *“Variable”* column in order to open an input field.

3. Enter a simple name (without '.') for a new variable (for example, `myBool`).

   ⇨ CODESYS creates the variable as an implicit global variable in the project and assigns it directly to the channel address. Therefore in this case the address does not appear struck through as in the case of mappings to existing variables.



Linking a device with a function block instance

If supported by the device, you can map entire function blocks to an input or output channel. This allows you to count the frequency of signal changes or scale a channel value for maintenance purposes, for example.

Here you will map a device output channel to a function block. In this example, the block scales the channel output value.

Requirement: A device with a digital output that supports FB mapping is linked in the project. There is a function block *"Scale_Output_Int"* with the following implementation. The attributes of the function block itself and before the output parameter with which the channel output is processed are important.

```
{attribute 'io_function_block'}
FUNCTION_BLOCK Scale_Output_Int
VAR_INPUT
    iInput : INT;
    iNumerator : INT;
    iDenominator : INT :=1;
    iOffset : INT := 0;
END_VAR
VAR_OUTPUT
 {attribute 'io_function_block_mapping'}
    iOutput : INT;
END_VAR
VAR
END_VAR
IF iDenominator <> 0 THEN
    iOutput := TO_INT(TO_DINT(iInput) * TO_DINT(iNumerator) /
TO_DINT(iDenominator)) + iOffset;
```

1. Open the *"<device name> I/O Mapping"* tab of the device editor. Double-click the output that should be connected to the function block. Click the button ✚ *"Add FB for IO channel"*.

   ⇨ The *"Select Function Block"* dialog opens. On the left side, you see at least the function block *"Scale_Output_int"* below the *"Application"* node. Libraries linked in the project that contain corresponding function blocks are also displayed for selection.

2. Select the POU `myScaleOutputInt`.

   ⇨ After clicking *"OK"*, the path of the function block parameter `iOutput` in the *"Variable"* is entered in the mapping dialog. The path comprises the application name, the device channel name, and the selected FB output (example: `App1.Out_4_Int_myScale_Output_Int_1.iOutput`).

3. Select the channel and click *"Go to Instance"*.

   ⇨ The focus switches to the *"<device name> IEC Objects"* tab and the created entry for the new IEC object `Out_4_Int_myScale_Output_Int_1`. In this view in online mode, you see the current value of the parameter `iOutput` for the channel `Out_4_Int` scaled by the FB. You can also write and force the value as in other monitoring views.

See also

- *Chapter 1.4.1.20.4.3 "Dialog 'Select Function Block'" on page 1150*
- *Chapter 1.4.1.20.2.8.11 "Tab '<device name> I/O Mapping'" on page 854*
- *Chapter 1.4.1.20.2.8.12 "Tab '<device name> IEC Objects'" on page 859*
- *Chapter 1.4.1.19.6.2.22 "Attribute 'io_function_block', 'io_function_block_mapping'" on page 707*

Changing and fixing an address value in the I/O map

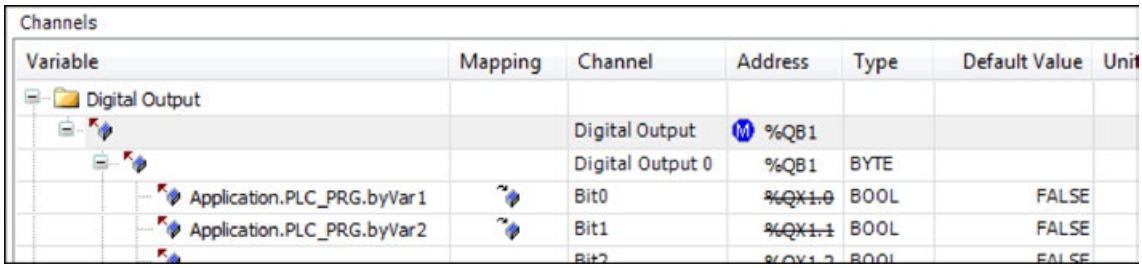
You can change the address value of an entire channel (but not that of an individual subelement of the channel!) in the mapping table of the *"<device name> I/O Mapping"* tab. This allows you to adapt the addressing to a specified machine configuration and to retain the address value even if the layout of the modules changes. By default, a change of the layout leads to an automatic adaptation of the address values.

Requirement: Your project has I/O mapping. See the corresponding sections of the help page above.

1. To open the device editor, double-click the device object in the device tree, and then the *"<device name> I/O Mapping"* tab.
2. Click in the mapping table on a channel entry in the *"Address"* column in order to open an input field. This is only possible for the 'root' address of a channel, not for a particular one of its subelements.

   Therefore, change the top address entry of a channel in the table, for example from `QB0` to `QB1`. Exit the input field.

   ⇨ The address value is changed. The symbol Ⓜ is displayed before the address. It indicates that the address is fixed. The addresses of the subelements of the channel are also changed accordingly. If you now change the position of the device object inside other device objects with input/output channels in the device tree, CODESYS does not adapt these addresses to the new order as would be the case without fixing.



3. In order to undo the manual change or fixing, open the input field of the address value again, delete the address entry and press the Enter key.

   ⇨ CODESYS resets the address and the subsequent addresses concerned to the values they had before the change and removes the symbol Ⓜ.

See also

- *Chapter 1.4.1.20.2.8.11 "Tab '<device name> I/O Mapping'" on page 854*
- *Chapter 1.4.1.19.4.10 "Addresses" on page 643*

Configuration of the I/O variable update

Depending on the device that you link in the project, CODESYS updates the variables applied to its inputs and outputs in different ways. You can explicitly change the settings for this in the *"I/O Mapping"* dialog.

See also

- *Chapter 1.4.1.20.2.8.11 "Tab '<device name> I/O Mapping'" on page 854*

Monitoring of variables in the I/O map in online mode

Requirement: You have compiled an application with a device configuration containing I/O maps without error. The associated hardware and the bus system are running. You have connected to the controller by means of the *"Online ➔ Login"* command and have loaded and started the application.

1. Open the *"I/O Mapping"* tab of the PLC in the device editor. To open the editor, double-click the device object in the device tree.

   ⇨ The mapping table now additionally contains the *"Current Value"* and *"New Value"* columns.

   If a structure variable is mapped to the 'root' element of an address[1], CODESYS does not display a value in this line in online mode. If, for example, a `DWORD` variable is mapped to the address, however, the respective values are monitored both in the 'root' line and in the indented bit-channel lines below it.

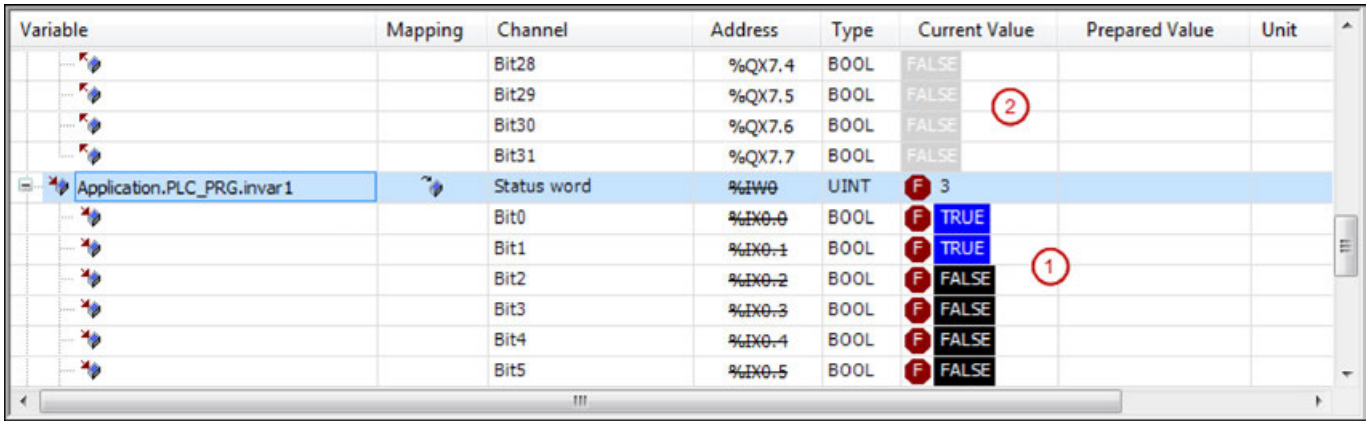
   As a matter of principle, the field in the 'root' line always remains empty if the value would be composed of several subelements.

   [1] 'root' = top element of this address in the Mapping dialog

2. Enter a certain variable value for an entry in the column *"New value"* and press *[F7]* to force or *[Ctrl]+[F7]* to write the value.

   ⇨ As in the case of monitoring in the declaration editor or in watch lists, the forced variable value is displayed in the column *"Current Value"* with a prefixed red F-symbol or the written value.

> **NOTICE!**
> Inputs and outputs that the PLC code does NOT use are not read by the PLC in online mode, as a result of which the displayed value could be incorrect. The *"Current Value"* of the variables concerned is displayed with a gray background.



(1) Forced values on the controller
(2) Values not used on the controller, value shown in gray

Generating implicit variables for the forcing of I/Os

During the commissioning of a plant or machine it may be necessary to 'force' the values applied at the inputs and outputs. If a device supports this you can cause special 'force variables' to be generated for this purpose and use them, for example, in an HMI visualization.

Requirement: The device supports the functionality. You have a project in which an I/O map is configured for the device and which contains a program object `PLC_PRG`.

1. Open the device editor, *"PLC Settings"* tab, by double-clicking the device object in the device tree.
2. Activate the option *"Generate force variables for IO mapping"*.
3. Press *[F11]* to compile the application.
   ⇨ Two variables are created for each I/O channel in accordance with the following syntax, in the process of which spaces in the channel name are replaced by underscores:

   `<device name>_<channel name>_<IEC address>_Force` of type `BOOL` for the activation and deactivation of forces.

   `<device name>_<channel name>_<IEC address>_Value` of the data type of the channel for defining the value that you want to force on the channel.

   These variables are available in the Input Assistant in the category *"Variables"* / *"IoConfig_Globals_Force_Variables."* You can use them in CODESYS in programming objects, in visualizations, in the symbol configuration, etc.
4. Open the function block *"PLC_PRG"*, set the focus in the implementation part and press *F2*.
   ⇨ The Input Assistant opens. The variables are available in the category *"Variables"* / *"IoConfig_Globals_Force_Variables"* as described above.

   A rising edge at the 'Force variable' input activates the forcing of the respective input or output with the value given by the 'Value variable'. A falling edge deactivates the forcing. Deactivation by resetting the 'Force' variable to `FALSE` is the requirement for being able to force a new value.

Take note of the following restrictions.

- Forcing via the implicit force variables is only possible for channels that are mapped in the *"I/O Mapping"* of the device to an existing or recently created variable.
- Forcing via the implicit force variables is not possible for unused inputs and outputs or those that are mapped to a variable via an AT declaration in an application program.
- I/O channels that you want to force via the mechanism have to be used by CODESYS in at least one task.
- CODESYS identifies forced inputs in the monitoring by the red Force symbol, but not forced input/outputs. The forced value is used only implicitly by the I/O driver for writing to the device.

See also

- *Chapter 1.4.1.20.2.8.9 "Tab 'PLC Settings'" on page 850*
- *Chapter 1.4.1.11.4 "Forcing and Writing of Variables" on page 401*

I/O mapping in one dialog for multiple devices

There is a table that displays the I/O map of a device plus the I/O maps of all subelements inserted below it in the device tree. There you can edit the I/O maps in exactly the same way as in the individual mapping tables of the respective device editors.

Requirement: In the device tree of your project there are several PLCs inserted that each enable an I/O mapping configuration.

1. Select the root node of the device tree and click *"Edit I/O Mapping"* in the context menu .
   ⇨ The *"Edit I/O Mapping"* dialog opens, in which the I/O mapping configurations of all devices inserted in the project are displayed in a table. You can edit the entries in the same way as in the *"I/O Mapping"* dialog of the associated device editor.

2. Now select one of the control objects in the device tree and select the *“Edit I/O Mapping”* command once again in the context menu.

   ⇨ The *“Edit I/O Mapping”* dialog now shows only the I/O table for the I/O mapping configurations found in and under the selected object.

3. Set a desired *“Filter”* in the bar above the table or enter a variable name in the *“Search for variable”* field in order to see the use of this variable in the mapping.

   ⇨ The method of working in this window is the same as that described for the *“<device name> I/O Mapping”* tab.

See also

- ↳ *Chapter 1.4.1.20.3.4.35 “Command 'Edit I/O Mapping'” on page 1018*

1.4.1.8 Programming of Applications

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

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

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

1.4.1.8.1 Designating identifiers

Identifiers are names of variables and programming objects (for example programs, function blocks, and methods) and names of other objects of the application and project. There are rules that you must follow when assigning identifiers. Furthermore, there are also recommendations to help you designate uniform and expressive identifiers.

You designate variables identifiers in the variables declaration. These identifiers can e changed in the declaration section of the programming object. You designate identifiers for programming objects and other objects in the dialog box when adding the object. You can change the identifier of an existing object of the application or of the project in the properties dialog of the object. However, you cannot change the identifiers of objects that can be available only one time per application or project (for example, the *“Library Manager”* and *“ImagePool”* identifiers).

See also

- ↳ *Chapter 1.4.1.19.7 “Identifiers” on page 740*

1.4.1.8.2 Declaration of Variables

Variable declaration: Where and how?

You can declare variables at the following locations:

- Declaration part of a POU
  The *“Declare Variable”* dialog helps you with this.
  Hint: If you define a variable in the tabular declaration editor, the correct syntax is automatically produced.
- Declaration part of the GVL or NVL editor
- I/O mapping configuration of an I/O device object

Syntax

```
( <pragma> )*
<scope> ( <type qualifier> )?
    <identifier> (AT <address> )? : <data type> ( := <initial
value> )? ;
END_VAR
```

	Declaration	See also ● *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226* ● *Chapter 1.4.1.8.2.2 “Using the 'Declare variable' dialog box” on page 227* ● *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*
<pragma>	Pragma (none, one, or multiple) Note: By adding a pragma, you can affect the behavior and the properties of one or more variables.	See also ● *Chapter 1.4.1.8.6 “Using Pragmas” on page 263* ● *Chapter 1.4.1.19.6 “Pragmas” on page 683*
<scope>	Scope ● VAR ● VAR_CONFIG Note: If variables with incomplete address information are declared in function blocks (for example, AT %I*), then the variables in the variable declaration VAR_CONFIG have to be completely declared. You can access these variables in a local instance only when this is done. ● VAR_EXTERNAL ● VAR_GLOBAL ● VAR_INPUT ● VAR_INST ● VAR_IN_OUT ● VAR_OUTPUT ● VAR_STAT ● VAR_TEMP	See also ● *Chapter 1.4.1.19.2.1 “Local variables - VAR” on page 526* ● *Chapter 1.4.1.19.2.10 “Configuration variables - VAR_CONFIG” on page 534* ● *Chapter 1.4.1.19.2.8 “External variables - VAR_EXTERNAL” on page 533* ● *Chapter 1.4.1.19.2.5 “Global variables - VAR_GLOBAL” on page 531* *Chapter 1.4.1.20.2.10 “Object 'GVL' - Global Variable List” on page 871* ● *Chapter 1.4.1.19.2.2 “Input variables - VAR_INPUT” on page 526* ● *Chapter 1.4.1.19.2.9 “Instance variables - VAR_INST” on page 533* ● *Chapter 1.4.1.19.2.4 “Input/Output Variable (VAR_IN_OUT)” on page 527* ● *Chapter 1.4.1.19.2.3 “Output variables - VAR_OUTPUT” on page 527* ● *Chapter 1.4.1.19.2.7 “Static variables - VAR_STAT” on page 532* ● *Chapter 1.4.1.19.2.6 “Temporary variable - VAR_TEMP” on page 532*

`<type qualifier>`	Type qualifier • `CONST` • `RETAIN` • `PERSISTENT`	See also • *Chapter 1.4.1.19.2.11 “Constant Variables - 'CONSTANT'” on page 534* • *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537* • *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*
`<identifier>`	Identifier, variable name Note: The rules listed in the chapter "Identifiers" must be followed without exception when assigning an identifier. In addition, you will find recommendations for uniform naming.	See also • *Chapter 1.4.1.19.7 “Identifiers” on page 740*
`AT <address>`	Assignment of an address in the input, output, or flag memory range (I, Q, or M) `AT % <memory area prefix> ( <size prefix> )? <memory position>` Example • `AT %I* // Incomplete address` • `AT %I7.5` • `AT %IW0` • `AT %QX7.5` • `AT %MD48`	See also • *Chapter 1.4.1.7.1 “Configuring Devices and I/O Mapping” on page 213* • *Chapter 1.4.1.8.11.2 “AT declaration” on page 281* • *Chapter 1.4.1.19.4.10 “Addresses” on page 643*
`<data type>`	Data type • `<elementary data type>` • `<user defined data type>` • `<function block>`	See also • *Chapter 1.4.1.19.5 “Data Types” on page 646* • *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835* *Chapter 1.4.1.20.2.18.2 “Object 'Function Block'” on page 883*
`<initial value>`	Initial value `<literal value> | <identifier> | <expression>`	See also • *Chapter 1.4.1.19.7 “Identifiers” on page 740* • *“Constants and literals” on page 632* • *Chapter 1.4.1.19.1.3.3 “ST expressions” on page 464*
`( ... )?`	Optional	
`( ... )*`	Optional repetition	

Example

GVL

```
{attribute 'qualified_only'}
{attribute 'linkalways'}
VAR_GLOBAL CONSTANT
    g_ciMAX_A : INT := 100;
    g_ciSPECIAL : INT := g_ciMAX_A - 10;
END_VAR
```

GVL_CONFIG

```
{attribute 'qualified_only'}
VAR_CONFIG
    // Generated instance path of variable at incomplete address
    PLC_PRG.fbDoItNow.XLOCINPUT AT %I*: BOOL := TRUE;
END_VAR
```

FB_DoIt (FB)

```
METHOD METH_Last : INT
VAR_INPUT
  iVar : INT;
END_VAR
VAR_INST
  iLast : INT := 0;
END_VAR

METH_Last := iLast;
iLast := iVar;

FUNCTION_BLOCK FB_DoIt
VAR_INPUT
    wInput AT %IW0 : WORD; (* Input variable *)
END_VAR
VAR_OUTPUT
    wOutput AT %QW0 : WORD; (* Output variable *)
END_VAR
VAR_IN_OUT
    aData_A : ARRAY[0..1] OF DATA_A; // Formal variable
END_VAR
VAR_EXTERNAL
    GVL.g_ciMAX_A : INT; // Declared in object GVL
END_VAR
VAR_STAT
    iNumberFBCalls : INT;
END_VAR
VAR
    iCounter: INT;
    xLocInput AT %I* : BOOL := TRUE; // VAR_CONFIG
END_VAR

iNumberFBCalls := iNumberFBCalls + 1;
```

PLC_PRG (PRG)

```
PROGRAM PLC_PRG
VAR
    iLoop: INT;
    iTest: INT;
    fbDoItNow : FB_DoIt;
    iTest_200: INT;
    aData_Now : ARRAY[0..1] OF DATA_A := [(iA_1 := 1, iA_2 := 10,
dwA_3 := 16#00FF),(iA_1 := 2, iA_2 := 20, dwA_3 := 16#FF00)];
END_VAR

iTest := GVL.g_ciMAX_A;
iTest_200 :=  2 * GVL.g_ciMAX_A;
fbDoItNow(aData_A := aData_Now);
FOR iLoop := 0 TO GVL.g_ciSPECIAL DO
    ;
END_FOR
```

Variable initialization

The standard initialization value for all declarations is `0`. In the declaration part you can also specify user-defined initialization values for each variable and each data type.

The user-defined initialization starts with the assignment operator `:=` and consists of any valid expression of the programming language ST (structured text). You thus define the initialization value with the help of constants, other variables or functions. If you use a variable, you must also initialize it.

Examples

```
VAR

                var1:INT := 12;                 // initialization
value 12

                x : INT := 13 + 8;              // initalization
value defined by an expression of constants

                y : INT := x + fun(4);          // initialization
value defined by an expression,
                                                // that contains a
function call; notice the order!


                z : POINTER TO INT := ADR(y);   // not described in
the standard IEC61131-3:
                                                // initialization
value defined by an adress function;
                                                // Notice: In this
case the pointer will not be initialized
                                                // during an Online
Change *)
                END_VAR
```

Notes on the order of initialization

From compiler version 3.5.3.40, variables in a function block are initialized in the following order: firstly, all constants in accordance with the order of their declarations, then all other variables in accordance with the order of their declarations.

NOTICE!

From compiler version 3.3.2.0, variables from global variable lists are always initialized before the local variables of a POU.

See also

- *Chapter 1.4.1.19.5.14 “Data Type 'ARRAY'” on page 660*
- *“Declaration and initialization of structure variables” on page 675*
- *Chapter 1.4.1.19.5 “Data Types” on page 646*
- *Chapter 1.4.1.19.6.2.15 “Attribute 'global_init_slot'” on page 699*

Using the declaration editor

The declaration editor is used for declaring variables in the variable lists and POUs.

The declaration editor offers two possible views: textual and tabular.

in the dialog in *"Tools ➔ Options ➔ Declaration Editor"*, you define whether only the textual view or only the tabular view is available, or whether you can switch between both views by means of the buttons on the right side of the editor view.

If the declaration editor is used in conjunction with a programming language editor, it appears as the declaration part at the top of the window of a POU.

Declaring in the textual declaration editor

The behavior and the appearance of the textual editor are configured with the settings in the dialog *"Tools ➔ Options ➔ Text Editor"*. The settings concern colors, line numbers, tab widths, indentations etc. The usual Windows functions are available, plus the IntelliMouse functions if necessary.

Requirement: You have opened a programming object (POU, GVL or NVL) of a project. The textual declaration editor has the focus.

▷ Enter the variable declarations in correct syntax. With *[F2]* you can open the dialog *"Input Assistant"* for the selection of the data type or a keyword.

Declaring in the tabular declaration editor

In the tabular declaration editor, you add variable declarations to a table with the following columns: *"Scope"*, *"Name"*, *"Address"*, *"Data type"*, *"Initialization"*, *"Comment"*, and *"Attributes"* (pragmas).

Requirement: A programming object (POU or GVL) of a project is open. The tabular declaration editor has the focus.

1. Click the button in the declaration header or select the command *"Insert"* in the context menu.
   ⇨ CODESYS inserts a new row for a variable declaration and the input field for the variable name opens.
2. Specify a valid variable identifier.
3. Open the other fields of the declaration line as required with a double-click and select the desired specifications from the drop-down lists or with the help of the dialogs which appear.

See also

- *Chapter 1.4.1.19.1.1 "Declaration Editor" on page 461*
- *Chapter 1.4.1.20.3.2.32 "Command 'Auto Declare'" on page 975*
- *Chapter 1.4.1.8.2.2 "Using the 'Declare variable' dialog box" on page 227*
- *Chapter 1.4.1.20.3.16.2 "Command 'Edit Declaration Header'" on page 1121*
- *"Dialog 'Input Assistant' - Tab 'Categories'" on page 978*

Using the 'Declare variable' dialog box

Requirement: A programming object (POU or GVL) of a project is open.

1. Select the command *"Edit ➔ Auto Declare"*.
   ⇨ The dialog box *"Auto Declare"* opens.
2. Select the desired scope for the variable from the selection list *"Scope"*.
3. Enter a variable name in the input field *"Name"*.
4. Select the desired data type from the selection list *"Type"*.
5. If the initialization value deviates from the standard initialization value, enter an initialization value for the variable.
6. Complete your entries with a click on *"OK"*.
   ⇨ CODESYS lists the newly declared variable in the declaration part of your programming object.

With the help of pragmas in the declaration part you can affect the processing of the declaration by the compiler.

See also

- *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*
- *Chapter 1.4.1.19.5 “Data Types” on page 646*
- *Chapter 1.4.1.8.6 “Using Pragmas” on page 263*

Declaring arrays

Requirement: A programming object (POU or GVL) of a project is open.

1. Click *“Edit ➔ Declare Variable”*.
   ⇨ The *“Declare Variable”* dialog opens.
2. Select the desired scope for the array from the drop-down list *“Scope”*.
3. Enter an identifier for the array in the *“Name”* input field.
4. Click the arrow button (>) next to the *“Data type”* input field and select the *“Array Assistant”* entry from the selection menu.
5. In the input fields *“Dimension 1”*, type in the lower and upper limit of the first dimension of the array (example: `1` and `3`).
   ⇨ The field *“Result”* displays the 1st dimension of the array (example: `ARRAY [1..3] OF ?`).
6. In the input field *“Basic type”*, type in the data type of the array or use the *“Input Assistant”* (>) or the *“Array Assistant”* (example: `DINT`).
   ⇨ The field *“Result”* displays the data type of the array now (example: `ARRAY [1..3] OF DINT`).
7. Define the second and third dimensions of the array according to steps 5 and 6 (example: Dimension 2: `1` and `4`, Dimension 3: `1` and `2`).
   ⇨ The *“Result”* field displays the array with the defined dimensions: `ARRAY [1..3, 1..4, 1..2] OF DINT`. The array consists of 3 * 4 * 2 = 24 elements.

   In an array of variable length, declare the dimension limits with an asterisk placeholder (`*`). Arrays of variable length are permitted to be used only in VAR_IN_OUT declarations of function blocks, methods, or functions.

   Example of a 2-dimensional array of variable length:
   `aiUnknownLengthData : ARRAY [*,*] OF INT;`
8. Click *“OK”*.
   ⇨ In the dialog *“Declare Variable”* the field *“Data type”* displays the array.
9. To modify the initialization values of the array, click the arrow button (...) next to the *“Initialization value”* input field.
   ⇨ The *“Initialization Value”* dialog opens.
10. Select the line of the array element whose initialization value you wish to modify. Example: Select array component [1, 1, 1].

11. Enter the desired initialization value in the input field below the list and click button *“Use value on selected lines”* (example: value `4`).

    ⇨ CODESYS displays the changed initialization value of the selected line.

12. Click *“OK”*.

    ⇨ In the *“Initialization value”* field of the *“Declare Variable”* dialog, CODESYS displays the initialization values of the array (example: `[4, 23(0)]`).

13. You can optionally enter a *“Comment”* in the input field.

14. Click *“OK”* in order to conclude the declaration of the array.

    ⇨ CODESYS adds the declaration of the array to the declaration part of the programming object.

See also

- *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*
- *Chapter 1.4.1.19.5.14 “Data Type 'ARRAY'” on page 660*

Declaring global variables

Declaring global variables that are available within the application.

Requirement: A project is open.

1. In the Device tree of your project, select the application in which the global variables are to be valid.
2. Select the context menu command *“Add Object ➔ Global Variable List”*.

    ⇨ CODESYS inserts the *“GVL”* in the Device tree under the application and opens it in the editor.

3. Select the menu command *“Edit ➔ Auto Declare”*.

    ⇨ The dialog box *“Auto Declare”* opens.

4. In the selection list *“Scope”*, select the entry *“VAR_GLOBAL”*.
5. In the field *“Name”*, enter a name for the global variable.
6. Select a data type from the selection list *“Type”*.
7. If your variable is to have an initialization value other than the standard initialization value, click on  next to the field *“Initialization”*.

    ⇨ The dialog box *“Initialization Value”* opens.

8. Double-click on the cell *“Init value”* of your variable and enter the desired valid value.
9. Click on *“OK”*.

    ⇨ The initialization value is displayed in the dialog box *“Auto Declare”*.

10. Activate one of the *“Flags”* if necessary.
11. Confirm your entries by clicking on the button *“OK”*.

    ⇨ CODESYS inserts the declared variable in the GVL.

    The global variable is available in the total application of your project.

Declaring global variables that are available in the entire project.

1. Select the menu command *"View ➔ POUs"*.
   ⇨ The *"POUs"* view opens.
2. In the *"POUs"* view, select the uppermost node with the project name and select the context menu command *"Add Object ➔ Global Variable List"*.
   ⇨ CODESYS inserts the *"GVL"* in the *"POUs"* view and opens it in the editor.
3. Select the menu command *"Edit ➔ Auto Declare"*.
   ⇨ The dialog box *"Auto Declare"* opens.
4. In the selection list *"Scope"*, select the entry *"VAR_GLOBAL"*.
5. In the field *"Name"*, enter a name for the global variable.
6. Select a data type from the selection list *"Type"*.
7. If your variable is to have an initialization value other than the standard initialization value, enter it in the column *"Initialization"*.
8. Activate one of the *"Flags"* if necessary.
9. Confirm your entries by clicking on the button *"OK"*.
   ⇨ CODESYS inserts the declared variable in the GVL.

   The global variable is now available in the entire project.

See also

- *Chapter 1.4.1.20.3.2.32 "Command 'Auto Declare'" on page 975*

Using Task-Local Variables

Task-local variables are cycle-consistent. In a task cycle, they are written only by a defined task, while all other tasks have read-only access. It is taken into account that tasks can be interrupted by other tasks or can run simultaneously. The cycle consistency also applies above all if the application is running on a system with a multicore processor.

Therefore, using task local global variable lists is one way to automatically achieve a synchronization (by the compiler) when multiple tasks are processing the same variables. This is not the case when using ordinary GVLs. Multiple tasks can write simultaneously to ordinary GVL variables during a cycle.

However, it is imperative to note: The synchronization of task-local variables requires a relatively large amount of time and memory and is not always the best solution for every application. For this reason, see below for more detailed technical information and best practice guidance to help you make the right decision.

In the CODESYS project, the *"Variable List (Task-Local)"* object is available for defining task-local variables. Syntactically, it corresponds to a normal GVL, but also contains the information of the task that has write access to the variables. Then all variables in such a GVL are not changed by another task during a cycle of a task.

The next section contains a simple example that demonstrates the principle and functionality of task-local variables. It includes a writing program and a reading program. The programs run in different tasks, but they access the same data that is stored in a task-local global variable list so that they are processed cycle-consistently.

Showing functionality in an example

See below for Instructions on reprogramming this sample application.

Sample application

```
(* task-local GVL, object name: "Tasklocals" *)
VAR_GLOBAL
    g_diaData : ARRAY [0..99] OF DINT;
END_VAR

PROGRAM ReadData
VAR
    diIndex : DINT;
    bTest : BOOL;
    diValue : DINT;
END_VAR
bTest := TRUE;
diValue := TaskLocals.g_diaData[0];
FOR diIndex := 0 TO 99 DO
    bTest := bTest AND (diValue = Tasklocals.g_diaData[diIndex]);
END_FOR

PROGRAM WriteData
VAR
    diIndex : DINT;
    diCounter : DINT;
END_VAR
diCounter := diCounter + 1;
FOR diCounter := 0 TO 99 DO
    Tasklocals.g_diaData[diIndex] := diCounter;
END_FOR
```

The programs *"WriteData"* and *"ReadData"* are called by different tasks.

In the program `WriteData`, the array `g_diaData` is populated with values. The program `ReadData` tests whether or not the values of the array are as expected. If so, then the variable `bTest` yields the result `TRUE`.

The array data that is tested is declared via the variable `g_diaData` in the object `Tasklocals` of type `Global Variable List (Task-Local)`. This synchronizes the data access in the compiler and guarantees cycle consistency, even when the accessing programs are called from different tasks. In the sample program, this means that the variable `test` is always `TRUE` in the program `ReadData`.

If the variable `g_diaData` were declared only as a global variable list in this example, then the test (the variable `test` in the program `ReadData`) would yield `FALSE` more often. In this case, this is because one of the two tasks in the `FOR` loop could be interrupted by the other task, or both tasks could run simultaneously (multicore controllers). And therefore the values could be changed by the writer while the reader reads the list.

Constraints in the declaration

NOTICE!
An online change of the application is not possible after changes in declarations in the list of task-local variables.

Note the following when declaring a global task-local variable list:

- Do not assign direct addresses by means of an AT declaration.
- Do not map to task-local variables in the controller configuration.
- Do not declare any pointers.
- Do not declare any references.

- Do not instantiate any function blocks.
- Do not declare any task-local variables as `PERSISTENT` and `RETAIN` at the same time.

The compiler reports write access in a task without write access as an error. However, not all write-access violations can be detected. The compiler can only assign static calls to a task. However, the call of a function block by means of a pointer or an interface is not assigned to a task, for example. As a result, any write access is not recorded there either. Moreover, pointers can point to task-local variables. Therefore, data can be manipulated in a read task. In this case, a runtime error is not issued. However, values that are modified by means of pointer access are not copied back in the shared reference of variables.

Properties of task-local global variables and possible behavior

The variables are located at a different address in the list for each task. For read access, this means: `ADR(variable name)` yields a different address in each task.

The synchronization mechanism guarantees the following:

- Cycle consistency
- Freedom from locked states: A task never waits for an action from another task at any time.

With this method, however, no time can be determined when a reading task securely receives a copy of the writing task. Fundamentally, the copies can deviate. In the example above, it cannot be concluded that each written copy is processed one time by the reader. For example, the reading task can edit the same array over multiple cycles, or the contents of the array can skip one or more values between two cycles. Both can occur and have to be considered.

The writing task can be paused for one cycle between two accesses to the shared reference by each reading task. This means that when `n` reading tasks exist, the writing task can have `n` cycles of delay until the next update of the shared reference.

In each task, the writing task can prevent a reading task from getting a reading copy. As a result, no maximum number of cycles can be specified after which a reading task will definitely receive a copy.

In particular, this can become problematic if very slow running tasks are involved. Assuming a task runs only every hour and cannot access the task-local variables during this time, then the task works with a very old copy of the list. Therefore, it can be useful to insert a time stamp in the task-local variables so that the reading tasks can at least determine whether or not the list is up-to-date. You can set a time stamp as follows: Add a variable of type `LTIME` to the list of task-local variables and add the following code to the writing task, for example: `tasklocal.g_timestamp := LTIME();`.

Best practice

Task-local variables are designed for the use case "Single writer - multiple readers". When you implement a code that is called by different tasks, using task-local variables is a significant advantage. For example, this is the case for the sample application `appTasklocal` as described above when it is extended by multiple reading tasks that all access the same array and use the same functions.

Task-local variables are especially useful on multicore systems. On these systems, you cannot synchronize tasks by priority. Then other synchronization mechanisms become necessary.

Do not use task-local variables when a reading task always has to work on the newest copy of the variable. Task-local variables are not suitable for this purpose.

A similar issue is the "Producer - Consumer" dilemma. This happens when a task produces data and another task processes the data. Choose another type of synchronization for this configuration. For example, the producer could use a flag to notify that a new date exists. Then the consumer can use a second flag to notify that it has processed its data and is waiting for new input. In this way, both can work on the same data. This removes the overhead for cyclic copying of data, and the consumer does not lose any data generated by the producer.

Monitoring

At runtime, multiple different copies of the task-local variable list may exist in memory. When monitoring a position, not all values can be displayed. Therefore, the values from the shared reference are displayed for inline monitoring, in the watch list, and in the visualization for a task-local variable.

When you set a breakpoint, the data of the task is displayed that ran to the breakpoint and was halted as a result. Meanwhile, the other tasks continue running. Under certain circumstances, the shared copy can be changed. In the context of the halted task, however, the values remain unchanged and are displayed as they are. You need to be aware of this.

Background: Technical implementation

For a list of task-local variables, the compiler creates a copy for each task, as well as a shared reference copy for all tasks. This creates a structure that contains the same variables as the list of task-local variables. Moreover, an array with this structure is created in which an array dimension is created for each task. As a result, an array element is indexed for each task. If a variable in the list is accessed now in the code, then the task-local copy of the list is actually accessed. Furthermore, it is determined in which task the block is currently running and the access is indexed accordingly.

For example, the line of code `diValue := TaskLocals.g_diaData[0];` from the above example is replaced by:

```
diValue := __TaskLocalVarsArray[__CURRENTTASK.TaskIndex].__g_diarr[0];
```

`__CURRENTTASK` is an operator that is available in CODESYS V3.5 SP13 and later in order to determine the current task index quickly.

At runtime, at the end of the writing task, the contents of the task-local list are written to the global list. For a reading task at the beginning, the contents of the shared reference are copied to the task-local copy. Therefore, for n tasks, there are n+1 copies of the list: One list serves as a shared reference and every task also has its own copy of the list.

A scheduler controls the time-based execution of multiple tasks and therefore also task switching. The strategy, which is tracked by the scheduler in order to control the allocation of the execution time, has the goal of preventing a task from being blocked. The synchronization mechanism is therefore optimized to the properties of task-local variables to prevent blocking states (lock states) and at no time does a task wait for the action of another task.

Synchronization strategy:

- As long as the writing task writes a copy back to the shared reference, none of the reading tasks gets a copy.
- As long as a reading task gets a copy of the common reference, the writing task does not write back a copy.

Instructions for creating the sample application as described above

Aim: With a program `ReadData`, you want to access the same data that is written by a program `WriteData`. Both programs should run in different tasks. You make the data available in a task-local variable list so that it is processed automatically in a cycle-consistent manner.

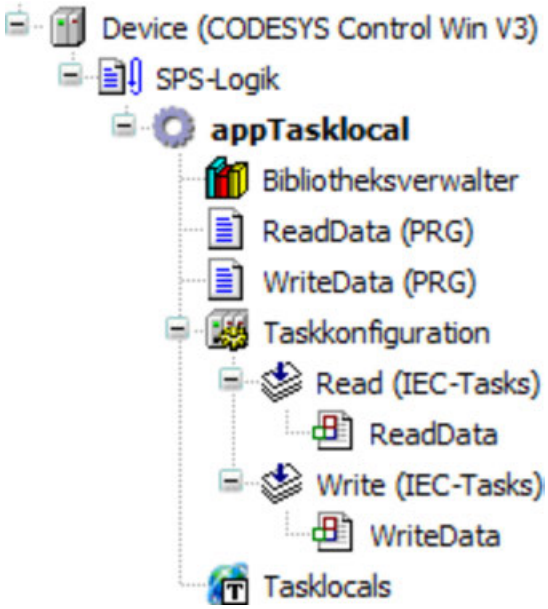
☑ Requirement: A brand new standard project is created and open in the editor.

1. Rename the application from `Application` to `appTasklocal`.
2. Below `appTasklocal`, add a program in ST named `ReadData`.
3. Below `appTasklocal`, add another program in ST named `WriteData`.
4. Below the object `Task Configuration`, rename the default task from `MainTask` to `Read`.
5. In the *“Configuration”* dialog of the task `Read`, click the *“Add Call”* button to call the program `ReadData`.
6. Below the *“Task Configuration”* object, add another task named `Write`, and add the call of the program `Write` to this task.

   ⇨ Now there are two tasks `Write` and `Read` in the task configuration which call the programs `WriteData` and `ReadData`, respectively.
7. Select the application `appTasklocal` and add an object of type *“Global Variable List (Task-Local)”*.

   ⇨ The *“Add Global Variable List (Task-Local)”* dialog opens.

8. Specify the name `Tasklocals`.
9. Select the `Write` task from the *“Task with write access”* list box.



⇨ The object structure for using task-local variables within an application is complete. Now you can code the objects as described in the example above.

See also

- *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*
- *Chapter 1.4.1.8.2.4 “Declaring global variables” on page 229*
- *Chapter 1.4.1.20.2.11 “Object 'GVL' - Global Variable List (task-local)” on page 872*
- *Chapter 1.4.1.8.11.2 “AT declaration” on page 281*
- *Chapter 1.4.1.12.1.1 “Calling of monitoring in programming objects ” on page 410*

1.4.1.8.3 Creating Source Code in IEC

Source code:

"Source code" is a term used for the implementation code, which you insert in the programming modules by using the appropriate programming language editors. The following programming module types are available for this purpose: POU (Program, Function, Function Block), Action, Method, Property, Interface.

Programming Language:

When creating a POU, you define, in which programming language the implementation should be inserted. Besides the IEC languages also CFC is available.

Programming Language:Editors:

You get a programming module editable in the corresponding programming language editor on a double-click on the programming module object. So, the module will appear either in the textual ST editor or in one of the graphical editors for FBD/LD/IL or CFC. Each editor consists of two windows: In the upper window you insert the declarations, in textual or tabular form, depending on the setting. In the lower window you insert the implementation code. The display and behaviour of each editor can be configured in the corresponding tab of the CODESYS *“Options”* dialog.

Regard the possibility to open a programming module for offline-editing even while the application is in online mode.

See also

- *Chapter 1.4.1.19.1 “Programming Languages and Editors” on page 460*
- *Chapter 1.4.1.20.3.4.11 “Command 'Edit Object (Offline)'” on page 1006*

FBD/LD/IL

A combined editor enables programming in the languages FBD (function block diagram), LD (ladder diagram) and IL (instruction list).

The basic unit of the FBD and LD programming is a network. Each network contains a structure that can represent the following: a logical or arithmetic expression, the call of a POU (function, function block, program etc.), a jump or a return instruction. IL actually requires no networks. In CODESYS, however , an IL program also consists of at least one network in order to support conversion to FBD or LD. In view of this you should also divide an IL program meaningfully into networks.

See also

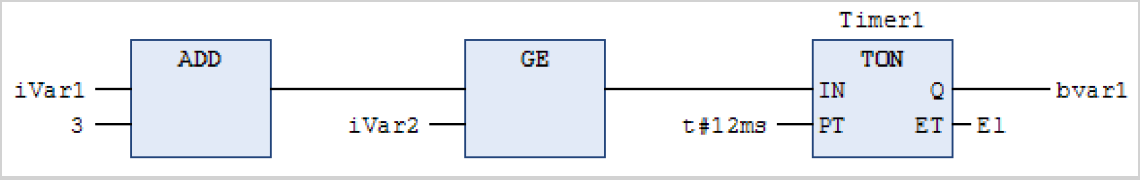
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Function block diagram (FBD)

The function block diagram is a graphically oriented IEC 61131 programming language. It works with a list of networks, where each network contains a structure that can contain logical and arithmetic expressions, calls of function blocks, a jump or a return instruction.

Boxes familiar from boolean algebra are used here. Boxes and variables are connected by connecting lines. The signal flow in the network runs from left to right. The signal flow in the editor runs from top to bottom, starting with network 1.

Example



CFC is also a programming language based on the same principle as FBD, but with the following differences:

- *The CFC editor is not network-oriented.*
- *You can freely place the elements in the CFC editor.*
- *Direct insertion of feedbacks is possible.*
- *The order of execution is determined by a list of currently inserted elements, which you can change.*

See also

- *Chapter 1.4.1.8.3.1.1 “Programming function block diagrams (FBD)” on page 237*
- *Chapter 1.4.1.20.3.13 “Menu 'FBD/LD/IL'” on page 1104*(commands)
- *Chapter 1.4.1.8.3.2 “Continuous Function Chart (CFC)” on page 241*

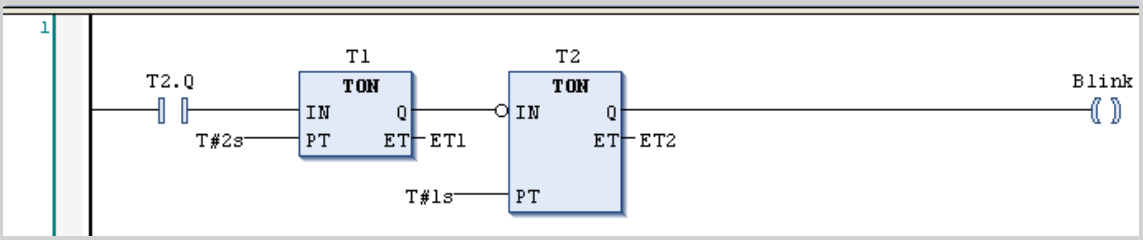
Ladder diagram (LD)

The ladder diagram (LD) is a graphically oriented programming language that approximates an electrical circuit diagram. On the one hand the ladder diagram is suitable for designing logical switching units, but on the other you can also create networks just as in FBD. Therefore you can use LD very well for controlling calls of other program blocks.

The ladder diagram consists of a series of networks. A network is bounded on the left side by a vertical line (bus bar). A network contains a circuit diagram of contacts, coils, optional boxes (POUs) and connecting lines. On the left side of a network there is a contact or a series of contacts that relay the ON or OFF state, which corresponds to the boolean values `TRUE` and `FALSE`, from left to right. A boolean variable is associated with each contact. If this variable is TRUE, the status is relayed from left to right via the connection line. Otherwise OFF is relayed. Thus the coil(s) in the right part of the network receive(s) the value ON and OFF coming from the left and the value TRUE or FALSE is written accordingly into the boolean variable assigned to them.

If the elements are connected in series, this means an AND operation. If they are connected in parallel, this means an OR operation. A line through an element means a negation of the element. The negation of an input or an output is indicated by a circle symbol.

Example



IEC 61131-3 defines a complete LD command set, consisting of different types of contacts and coils. Contacts conduct the current (according to their type) from left to right. Coils store the incoming value. Contacts and coils are assigned to boolean variables. You can supplement an LD network by jumps, returns, labels and comments.

See also

- *Chapter 1.4.1.8.3.1.2 “Programming ladder diagrams (LD)” on page 239*
- *Chapter 1.4.1.20.3.13 “Menu 'FBD/LD/IL'” on page 1104*(Befehle)

Instruction list (IL)

The instruction list is an assembler-like IEC 61131-compliant programming language. It supports accumulator-based programming.

An instruction list (IL) consists of a series of instructions. Each instruction starts in a new line and contains an operator and, depending on the type of operation, one or more operands separated by commas. A label, followed by a colon, can be placed in front of an instruction. It serves the identification of the instruction and you can use the label as a jump destination. A comment must be the last element in a line. Empty lines can be inserted between instructions.

All IEC 61131-3 operators are supported, as are multiple inputs, multiple outputs, negations, comments, set/reset of outputs and conditional/unconditional jumps.

Each instruction is based primarily on the loading of values into the accumulator (`LD` instruction). After that the corresponding operation is executed with the parameter from the accumulator. The result of the operation is written again into the accumulator, from where you should store it purposefully with the help of an `ST` instruction.

The instruction list supports comparison operators (EQ, GT, LT, GE, LE, NE) and jumps for programming of conditional executions or loops. Jumps can be unconditional (JMP) or conditional (JMPC / JMPCN). In the case of conditional jumps, a check is performed as to whether the value in the accumulator is `TRUE` or `FALSE`.

Example

```
1
LD      BVar1
ST      inst.IN     starts timer with rising e…
JMPC    mark1
CAL     inst(
        PT:=t1,                                 is True
        ET:=>t2)
LD      inst.Q      gets TRUE delay time after…
ST      inst2.IN    starts timer with rising e…
2


mark1:
LD      iVar2
ADD     230
```

See also

- *Chapter 1.4.1.8.3.1.3 “Programming in instruction list (IL)” on page 240*
- *Chapter 1.4.1.19.1.5.3 “Modifiers and operators in IL” on page 500*

Programming function block diagrams (FBD)

Creating a POU in the function block diagram (FBD) implementation language

1. Select an application in the device tree.
2. Select the command *“Project ➔ Add Object ➔ POU”*.
   ⇨ The dialog box *“Add POU”* opens.
3. Enter a name and select the implementation language *“Function Block Diagram (FBD)”*.
   Click on *“Add”*.
   ⇨ The POU is added to the device tree and opened in the editor. It consists of the declaration editor in the top part and the implementation part with an empty network in the lower part. The view *“ToolsBox”* is also automatically opened , in which the suitable elements, operators and function blocks for FBD programming are available.

See also

- *Chapter 1.4.1.20.2.18 “Object 'POU'” on page 881*
- *Chapter 1.4.1.8.3.1 “FBD/LD/IL” on page 235*

Programming a network

1. Click inside the automatically inserted empty network in the implementation part.
   ⇨ The network is given a yellow background and the area at the left-hand side with the network number is given a red background.

2. Open the context menu with the right mouse button.
   ⇨ You obtain amongst other things the insert commands for the elements that can be inserted at this point.
3. Insert the elements required for your programming using the menu commands or by dragging in the elements from the toolbox.
4. For example, select the command *“Insert Assignment”*.
   ⇨ An assignment line is inserted. In each case three question marks stand for assignment source and assignment target.
5. Select the question marks and replace them with the desired variable. Input assistance is available for this purpose.
6. Move the cursor over the assignment line.
   ⇨ The possible insertion positions for further elements are displayed as grey diamonds. A click on a diamond selects that position and the suitable insert commands are once again available.
7. Alternatively, you can drag an element with the mouse from the toolbox into the network. For example, click in the tool box on the box element, keep the mouse button pressed and move the cursor over the network.
   ⇨ Each possible insertion position lights up green.
8. Release the mouse button in order to insert the box.
   ⇨ The box is displayed in the network. The type of box on the inside and the instance name above the box, which is required in the case of a function block, are still kept free with three question marks.
9. Select the string `???` inside the box and replace it with the name of the box. Input assistance is available for this purpose.
   ⇨ The inputs and outputs of the selected box are displayed. They are still kept free with question marks, as is the instance name in the case of a function block.

See also

- *Chapter 1.4.1.20.3.13 “Menu 'FBD/LD/IL'” on page 1104*
- *Chapter 1.4.1.8.3.1 “FBD/LD/IL” on page 235*

Programming line branches (subnetworks)

1. In the implementation part of your POU, insert a new network using the command *“FBD/LD/IL ➔ Insert network”* or drag it in from the tool box.
2. For example, drag an *“ADD”* operator into the empty network and replace the characters `???` with two variables of the type `INT`.
3. Drag the element *“Branch”* from the tool box into your implementation and release the mouse button at the green insertion position directly at the output of the operator.
   ⇨ The line branch splits the processing line at the output of the operator box into 2 subnetworks.
4. Further FBD elements and also further line branches can now be added to each of the two subnetworks.

See also

- *Chapter 1.4.1.19.1.5.4.9 “FBD/LD/IL element 'Branch'” on page 506*
- *Chapter 1.4.1.20.3.13.33 “Command 'Insert Branch'” on page 1113*

See also

- ↳ *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Programming ladder diagrams (LD)

Creating a POU in the ladder diagram (LD) implementation language

1. Select the application in the Device tree.
2. Select the command *“Project ➔ Add Object ➔ POU”*.
   ⇨ The dialog box *“Add POU”* opens.
3. Enter a name and select the implementation language *“Ladder Diagram (LD)”*.
   Click on *“Add”*.
   ⇨ CODESYS adds the POU to the Device tree and opens it in the editor. An empty network is inserted in the implementation part. The empty network is bounded on the left by a vertical line, which represents a bus bar. The view *“ToolBox”* is also automatically opened, in which the suitable elements, operators and function blocks for LD programming are available.

Adding a contact and a function block (TON)

☑ Requirement: a POU with the implementation language LD is opened in the editor and an empty network is inserted.

1. Click on the category *“Ladder Elements”* in the view *“ToolBox”*
2. Click on the *“Contact”* element, drag it into your network and release the mouse button at the insertion position *“Start here”*.
   ⇨ The contact is added on the left in the network directly against the vertical line.
3. Click on `???` and enter the identifier of a boolean variable. The input assistant is also available to you for this.
4. Click on the category *“Function Blocks”* in the view *“ToolBox”* and drag the function block *“TON”* onto an insertion position on the connecting line to the right of the inserted contact.
   ⇨ CODESYS inserts the box *“TON”* to the right of the contact. The contact is connected with the input `IN` of the TON box.
5. Enter a time constant at the input `PT`, for example `T#3s`.
   ⇨ If the variable of your contact goes `TRUE`, then the input `IN` of the TON box also goes `TRUE`. The TON box forwards the value `TRUE` to the output `Q` with a switch-on delay of `T#3s`, for example.

Inserting a closed line branch

Requirement: a POU with the implementation language LD is opened in the editor and an empty network is inserted.

1. Click inside the empty network and select the command *“FBD/LD/IL ➔ Insert Contact”*.
2. Select the connecting line to the left of the contact and select the command *“FBD/LD/IL ➔ Set Branch Start Point”*.
   ⇨ The starting point on the connecting line is marked by a red rectangle. CODESYS marks all possible end points of the branch with a blue rectangle.

3. Click on a blue rectangle in order to set the end point of your closed line branch.
   ⇨ CODESYS inserts the line branch between the starting and end points. The program flow will go through both branches up to the end point.

   If you insert the line branch at a box instead of at a contact, the box will only be called if none of the other branches is `TRUE`.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.14 “Closed branch” on page 509*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Programming in instruction list (IL)

If necessary, IL can be activated in the CODESYS options.

Creating POUs in the instruction list (IL) implementation language

1. Select the application in the device tree.
2. Click *“Project ➔ Add Object ➔ POU”*.
   ⇨ The *“Add POU”* dialog opens.
3. Enter a name and select the implementation language *“Instruction List (IL)”*.

   Click *“Add”*.
   ⇨ CODESYS adds the POU to the device tree and opens it in the editor. A network is already inserted in the implementation part.

Programming networks (example: ADD operation)

Requirement: A POU (IL) is opened in the editor and possesses an empty network.

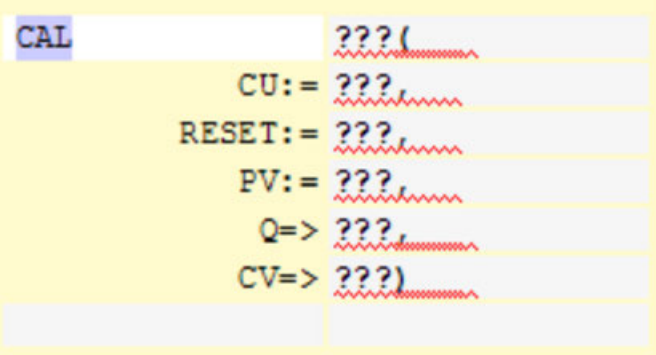
1. Click the line marked in color in the 1st column and enter the operator `LD`.
2. Press the *[Tab]* key.
   ⇨ The cursor jumps to the 2nd column
3. Enter the first summand of your ADD operation, for example `6`.
4. Press *[Ctrl]+[Enter]* or select the command *“FBD/LD/IL ➔ Insert IL Line After”*.
   ⇨ CODESYS inserts a new instruction line. The first column of this line has the focus.
5. Enter `ADD` and press *[Tab]*.
6. Enter the second summand of your ADD operation, for example `12`.
7. Press *[Ctrl]+[Enter]*
8. Enter the operator `ST` and press *[Tab]*.

9. Specify a variable of the data type `INT`, for example `iVar`.
   ⇨ The result – `16` in the example – is stored in the `iVar`.

Calling function blocks

Requirement: A POU (IL) is opened in the editor and possesses an empty network. A variable of the data type <function block> is declared in the declaration part (example: `C1:CTU;`).

1. Click the line marked in color in the 1st column and select the command *"FBD/LD/IL ➔ Insert Box"*.
   ⇨ The input assistant opens.
2. Select the desired function block in the category *"Function Blocks"* or *"Boxes"*, for example the *"CTU"* counter from the *"Standard"* library, and click *"OK"*.
   ⇨ CODESYS inserts the selected function block *"CTU"* as follows:



3. Replace the strings `???` with the variable name and the values or variables for the inputs/outputs of the function block.
4. As an alternative to inserting the function block via the input assistant, you can directly enter the call in the editor as shown in the picture at step 4.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*
- *Chapter 1.4.1.19.1.5.3 "Modifiers and operators in IL" on page 500*

Continuous Function Chart (CFC)

The *"Continuous Function Chart (CFC)"* implementation language is a graphical programming language which extends the standard languages of IEC 61131-3.

You can graphically program a system by means of a POU in CFC. You insert elements and position them freely. You insert connections and wire the elements to a network so that a well-structured function block diagram is created. You can also insert feedback. You can read function block diagrams like an circuit diagram or a block diagram.

The execution order of a function block diagram is based on data flow. Moreover, a POU can process multiple data flows. Then the data flows do not have any common data. In the editor, multiple networks do not have any connections to each other.

On the other hand, POUs in FBD, LD, or IL have a network-based execution order.

The *"Continuous Function Chart (CFC) - page-oriented"* implementation language is also a graphical programming language which extends the standard languages of IEC 61131-3.

In this language, you can graphically program large, complex function block diagrams. The same elements and commands are available as for *“Continuous Function Chart (CFC)”*. In addition, you can arrange the code on as many pages as you like. This allows you to create extensive function block diagrams that are still easy to print. Furthermore, each page has border areas. You can arrange inputs and sink connection marks on the left, and outputs and source connection marks on the right. This helps you to insert connecting lines and provides a better overview.

Unfortunately, it is not possible to switch a POU between the “Continuous Function Chart (CFC) - page-oriented” and “Continuous Function Chart (CFC)” implementation languages.

See also

- *Chapter 1.4.1.19.1.6.1 “CFC Editor” on page 511*
- *Chapter 1.4.1.19.1.6.2 “CFC editor, page-oriented” on page 514*
- *Chapter 1.4.1.20.3.12 “Menu 'CFC'” on page 1089*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*

Automatic Execution Order by Data Flow

The execution order in POUs is uniquely determined in text-based and network-based editors. In the CFC editor, however, you can position the elements freely, so the execution order is initially not unique. For this reason, CODESYS determines the execution order by data flow and, in the case of multiple networks, by the topological position of the elements. The elements are sorted from top to bottom and left to right. Now the execution order is unique and makes sure that the POU is processed while optimized by time and by cycle.

You can get information about the chronological order of the elements in the chart and temporarily display the execution order. When you program networks with feedback you can define an element as the starting point in the feedback loop.

You can also explicitly edit the processing order in a CFC object if necessary. To do this, switch the *“Auto Data Flow Mode”* property of the CFC object to *“Explicit Execution Order Mode”*. In this mode, you have the option of editing the execution order by means of menu commands.

Before CODESYS Development System V3.5 SP15, you had to define the execution order explicitly for each POU. The was no mode switching.

Data flow

In general, data flow is described as the chronological order in which data is read or written when and how in which programming objects. A POU can process any number of data flows, which can also be executed independently of each other.

Displaying the execution order

By default, the execution order of a CFC object is determined automatically. The *“Auto Data Flow Mode”* property is selected for this. You can temporarily display the automatically determined execution order in the CFC editor.

1. Create a new project using the *"Standard project"* template and specify the name `Minimal` for example.
2. Extend the application with the function block `FB_DOIt` in the *"ST"* implementation language with inputs and outputs.

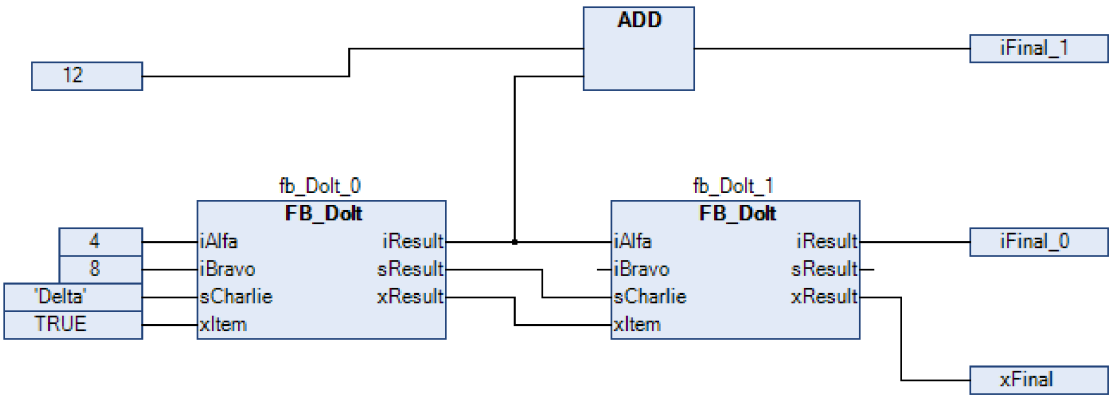
   ⇨

```
FUNCTION_BLOCK FB_DoIt
VAR_INPUT
    iAlfa : INT;
    iBravo: INT;
    sCharlie : STRING := 'Charlie';
    xItem : BOOL;
END_VAR
VAR_OUTPUT
        iResult : INT;
        sResult : STRING;
        xResult : BOOL;
END_VAR
VAR
END_VAR

iResult := iAlfa + iBravo;

IF xItem = TRUE THEN
    xResult := TRUE;
END_IF
```

3. Create the function block `ExecuteCFC` in the *"CFC"* implementation language.

   ⇨

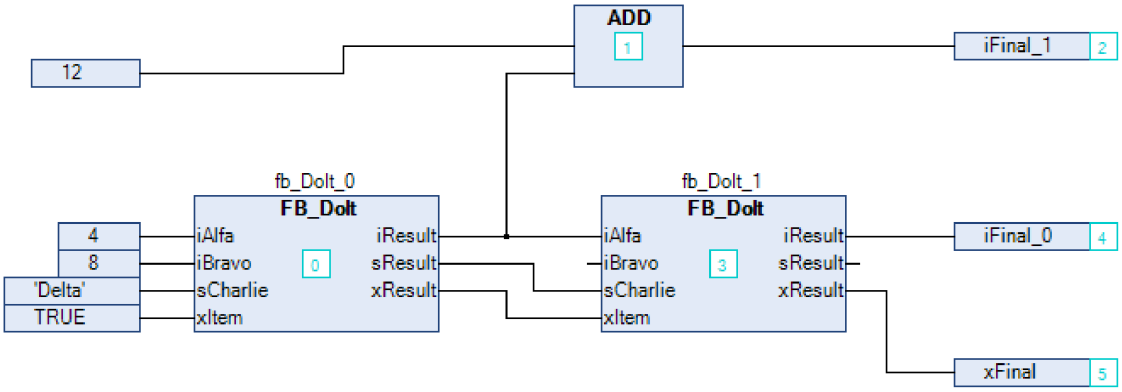
```
PROGRAM ExecuteCFC
VAR
    fb_DoIt_0: FB_DoIt;
    fb_DoIt_1: FB_DoIt;
    iFinal_1: INT;
    iFinal_0: INT;
    xFinal: BOOL;
END_VAR
```



Recently created programming objects in CFC have the Auto Data Flow Mode selected. The execution order of the programming object is optimally defined internally.

4. Click *“CFC ➔ Execution Order ➔ Display Execution Order”*.

   ⇨ The execution order of the object is shown. The boxes and inputs are numbered accordingly and reflect the chronological processing order. The numbering is hidden as soon as you click again in the CFC editor.

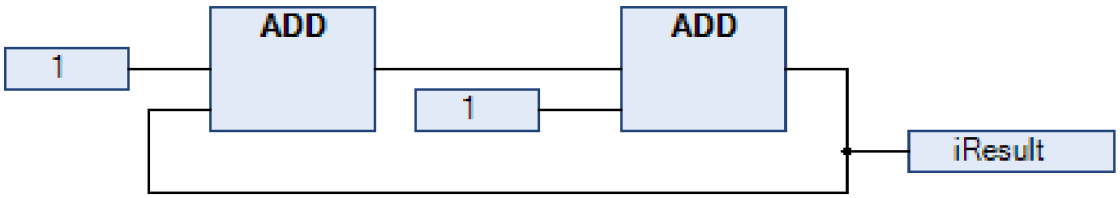


Determining the execution order in feedback networks

1. Create a CFC program with feedback.

   ⇨ The POU `PrgPositiveFeedback` counts.

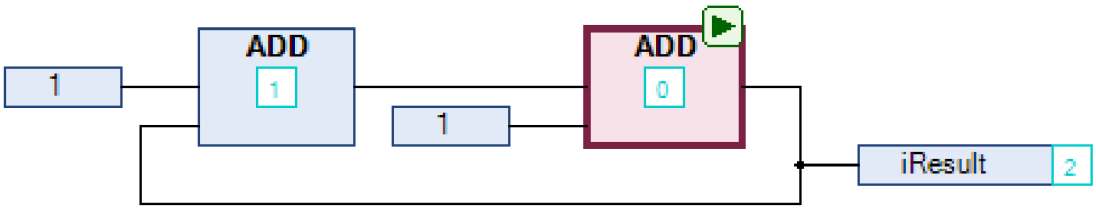
```
PROGRAM PrgPositiveFeedback
VAR
    iResult: INT;
END_VAR
```



2. Select an element within the feedback.

   ⇨ The selected element is highlighted in red.

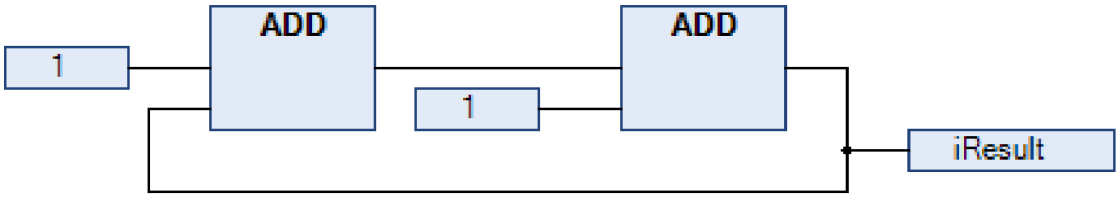
3. Click *“CFC ➔ Execution Order ➔ Set Start of Feedback”*.



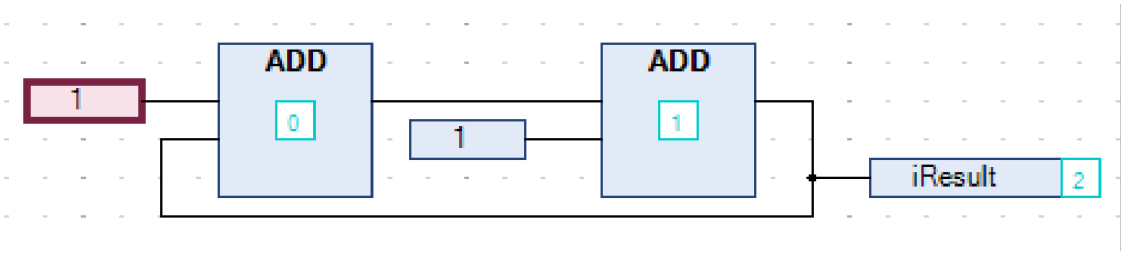
   ⇨ At run time, this POU is processed first. The start POU of the feedback is defined and decorated with the ▶ symbol. The execution order is resorted and the selected element gets the number 0. (This is the lowest number of the feedback.)

4. Select the start POU again.

5. Click *"CFC ➔ Execution Order ➔ Set Start of Feedback"*.
   ⇨ The POU is not selected as the start POU.



   The execution order is defined internally.
6. Click *"CFC ➔ Execution Order ➔ Display Execution Order"*.
   ⇨ The execution order by data flow is displayed.



Defining the execution order explicitly

> *The automatically defined execution order by data flow results in time- and cycle-optimized execution of the POU. You do not need any information about the internally managed execution order during the development process.*
>
> *In "Explicit Execution Order Mode", it is your responsibility to adapt the execution order and to assess the consequences and impacts. This is another reason why the execution order is always displayed.*

You can change the automatically defined execution order of a CFC object explicitly when you select the *"Explicit Execution Order Mode"* option for the object.

1. In the *"Devices"* or *"POUs"* view, select a CFC object.
2. In he context menu, click *"Properties"*.
3. Click the *"CFC Execution Order"* tab.
   ⇨ The *"Execution order"* list box displays the currently selected mode.
4. In the *"Execution order"* list box, select *"Explicit Execution Order Mode"*.
5. Click *"OK"* to confirm the dialog.
   ⇨ The Explicit Execution Order Mode property is selected. The networks are numbered in the CFC editor, and the following commands are provided in the *"CFC ➔ Execution Order"* menu for editing the execution order.
6. Open a CFC object.
7. Select a numbered element and click *"CFC ➔ Execution Order ➔ Send to Front"*.
   ⇨ The execution order is resorted and the selected element has the number 0.

See also

- *Chapter 1.4.1.19.1.6.1 "CFC Editor" on page 511*
- *Chapter 1.4.1.19.1.6.2 "CFC editor, page-oriented" on page 514*
- *Chapter 1.4.1.20.3.12 "Menu 'CFC'" on page 1089*
- *Chapter 1.4.1.20.4.10.13 "Dialog 'Properties' - 'CFC Execution Order'" on page 1165*

Programming in the CFC editor

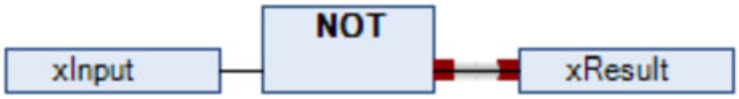
In the CFC editor, you can wire POUs to each other and create well-structured block diagrams.

The editor supports you in the following ways:

- Programming with elements and connecting lines
- Dragging instances and variables to the editing area
- Auto-routing the connecting lines
- Automatic linking
- Fixing of connecting lines by control points
- Collision detection
- Input assistance for connection marks
- Forcing and writing of values in online mode
- Movement of selection using arrow keys
- Reduced display of a POU without disconnected pins

Inserting elements and wiring with connecting lines

1. Drag a *“Box”* element and an *“Output”* element into the editor.
2. Click the output of the *“Box”* element.
   ⇨ The output is marked with a red box.
3. Drag a connecting line from the box output of the *“Box”* element to the box input of the *“Output”* element.
   ⇨ The cursor symbol changes when it reaches the box input.
4. Release the left mouse button.
   ⇨ The output pin of the box is wired to the input pin of the output.



You can also hold down the *[Ctrl]* key, select each pin, and then right-click *“Connected Selected Pins”*.

Calling of instances

1. Create a new project using the standard template and specify the name `First` for example.
   ⇨ The project `First.project` is created.

2. Extend the application with the function block `FB_DOIt` in the *"ST"* implementation language with inputs and outputs.

   ⇨
   ```
   FUNCTION_BLOCK FB_DoIt
   VAR_INPUT
       iAlfa : INT;
       iBravo: INT;
       sCharlie : STRING := 'Charlie';
       xItem : BOOL;
   END_VAR
   VAR_OUTPUT
           iResult : INT;
           sResult : STRING;
           xResult : BOOL;
   END_VAR
   VAR
   END_VAR

   iResult := iAlfa + iBravo;

   IF xItem = TRUE THEN
       xResult := TRUE;
   END_IF
   ```

3. Select the application and click *"Add Object ➔ POU"* in the context menu. Select the *"Continuous Function Chart (CFC)"* implementation language and the type `Program`. Specify the name `PrgFirst` for example.

   Click *"OK"* to confirm the dialog.

   ⇨ The program `PrgFirst` is created and it opens in the editor. It is still empty.

4. Instantiate function blocks and declare variables.

   ⇨
   ```
   PROGRAM PrgFirst
   VAR
       iCounter: INT;

       fbDoIt_1 : FB_DoIt;
       fbDoIt_2 : FB_DoIt;

       iOut : INT;
       sOut: STRING;
       xOut: BOOL;

   END_VAR
   ```

5. Drag a *"Box"* element from the *"ToolBox"* view into the editor.

6. Click the `???` field and type in `ADD`.

   ⇨ The box type is `ADD`. The box acts as an adder.

7. Click line 3 in the declaration editor.

   ⇨ The declaration line of `iCounter` is selected.

8. Click in the selection and drag the selected variable into the implementation. Focus there on an input of the `ADD` box.

   ⇨ An input has been created, declared, and connected to the box.

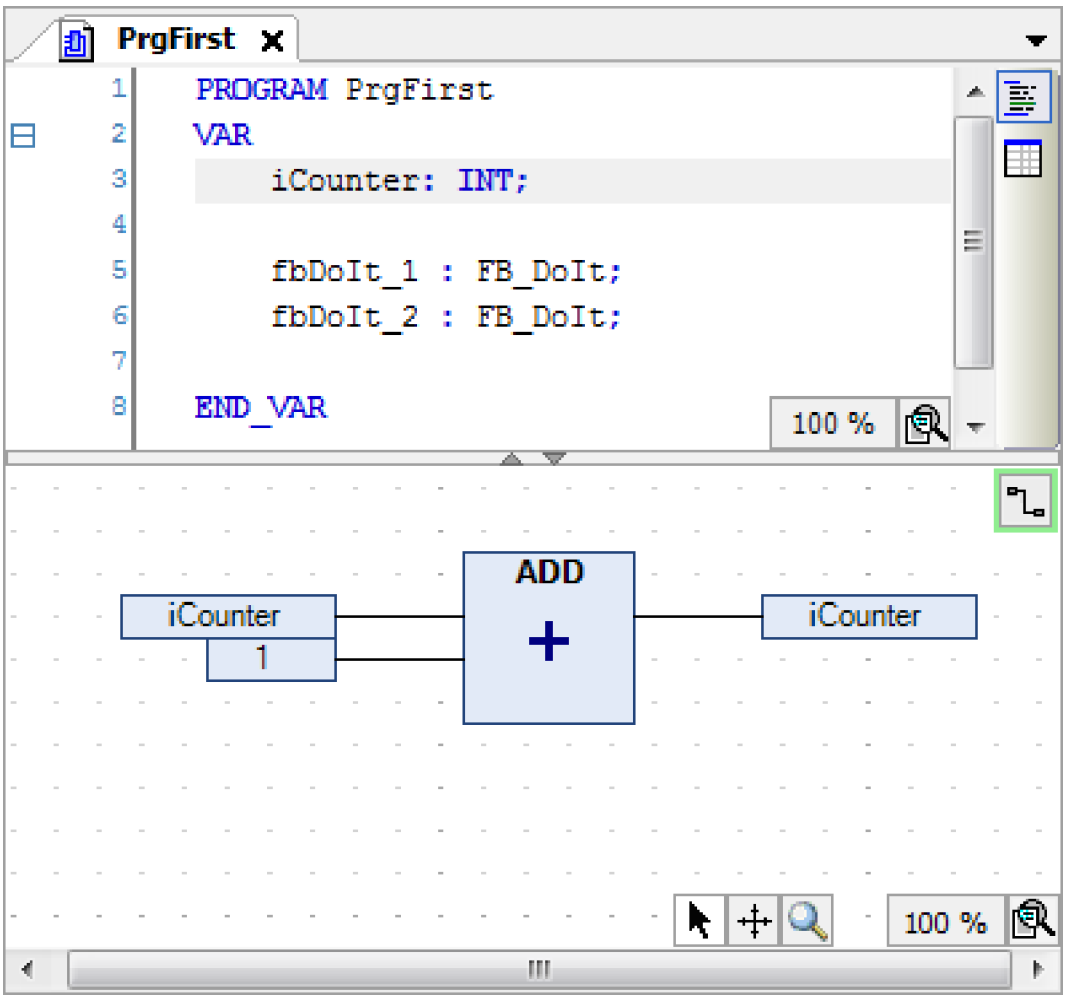
9. Click again in the selection and drag the variable to the output of the `ADD` box.

   ⇨ An output has been created, declared, and connected to the box.

10. Drag an *"Input"* element from the *"ToolBox"* view to the implementation. Click its `???` field and type in `1`.

11. Connect the `1` input to an input of the `ADD` box.

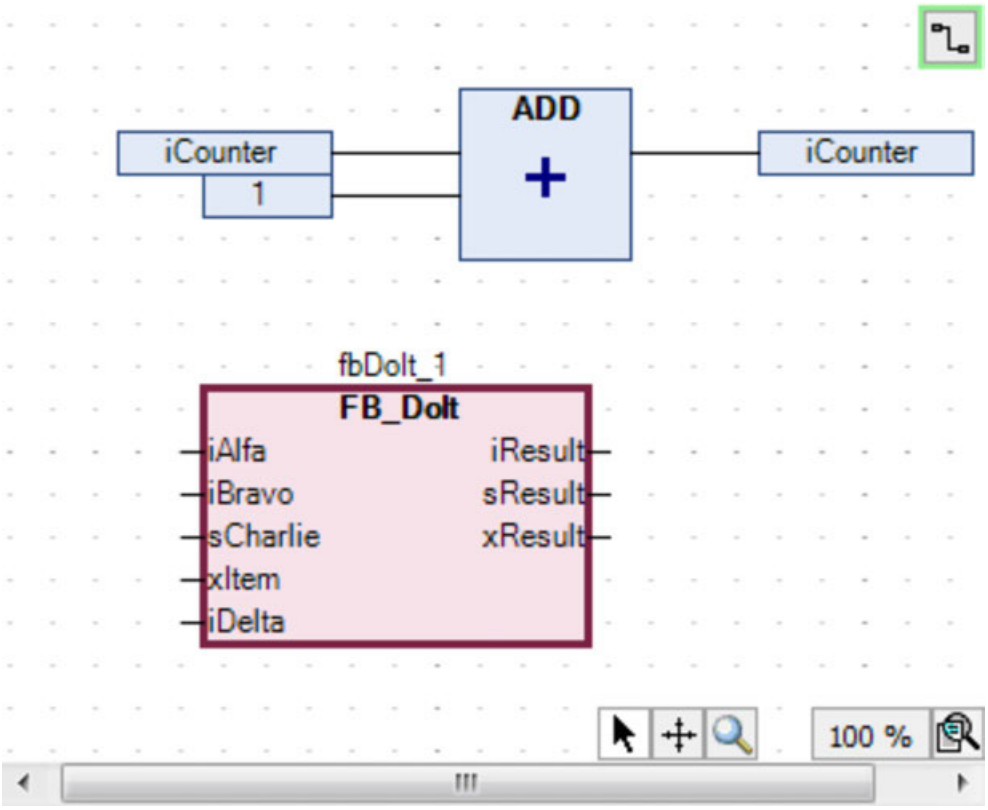
    ⇨ A network is programmed. At runtime, the network counts the bus cycles and stores the result in `iCounter`.



12. Click line 5 in the declaration editor.

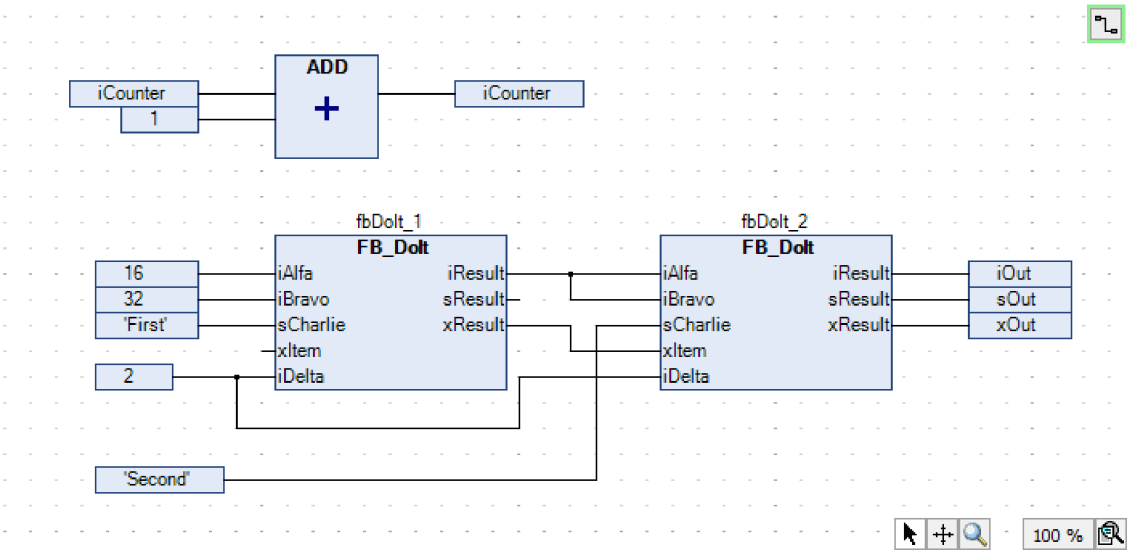
    ⇨ The line is selected.

13. Click in the selection and drag the selected instance into the implementation.
    ⇨ The instance appears as a POU in the editor. The type, name, and POU pins are displayed accordingly.



14. Drag the `fbDoIt_2` instance to the editor. Interconnect the instances to each other and to inputs and outputs.

    ⇨ Example:



A program in ST with the same functionality might look like this:

```
PROGRAM PrgFirstInSt
VAR
    iCounter: INT;

    fbDoIt_1 : FB_DoIt;
    fbDoIt_2 : FB_DoIt;

    iOut : INT;
    sOut: STRING;
    xOut: BOOL;

END_VAR
iCounter := iCounter + 1;
fbDoIt_1(iAlfa := 16, iBravo := 32, sCharlie := 'First',
xItem := TRUE, iDelta := 2, iResult => fbDoIt_2.iAlfa, xResult
=> fbDoIt_2.xItem);
fbDoIt_2(iBravo := fbDoIt_1.iResult, sCharlie := 'Second',
iDelta := 2, iResult => iOut , sResult=> sOut, xResult =>
xOut);
```

Creating connection marks

☑ Requirement: A CFC POU has connected elements.

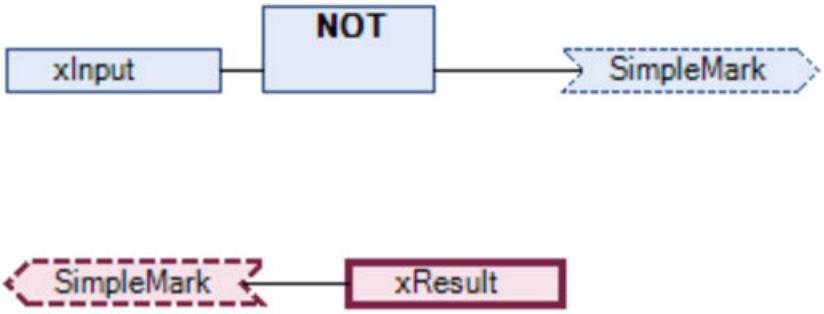
1. Select a connecting line between two elements.

   ⇨ The connecting line is displayed as selected. The ends of the connecting line are marked with red boxes (■).

2. Click *“CFC ➔ Connection Mark”*.

   ⇨ The connection is separated into a *“Connection Mark - Source”* and a *“Connection Mark - Sink”*. The name of the mark is generated automatically.

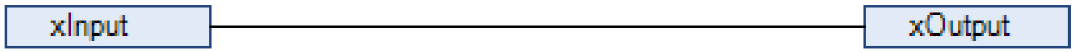
3. Click in the source connection marks.
   ⇨ You can edit the name.
4. Specify a name `SimpleMark` for the source connection mark.
   ⇨ The source connection mark and sink connection mark have the same name.



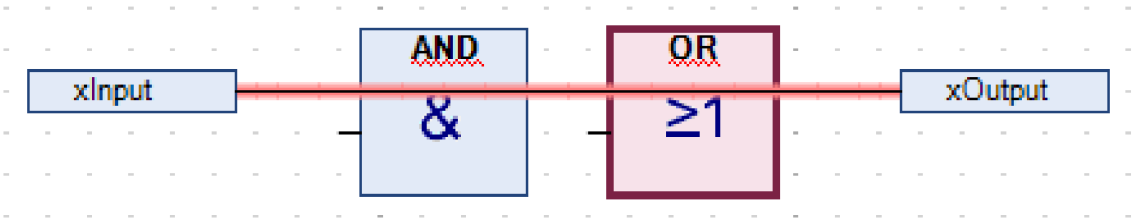
Resolving collisions and fixing connecting lines by means of control points

The following example shows how to use the *"Route All Connections"* command with control points.

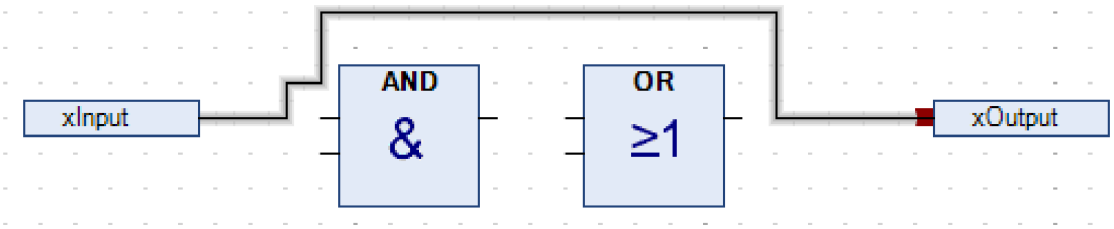
1. Position the *"Input"* and *"Output"* elements. Connect the elements.



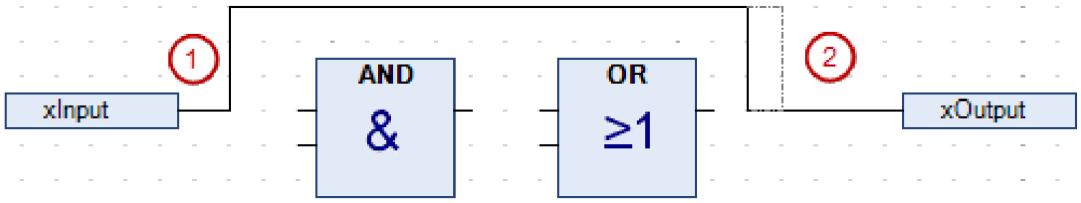
2. Position two *"Box"* elements on the line.
   ⇨ The connecting line and the boxes are marked red because of the collision.



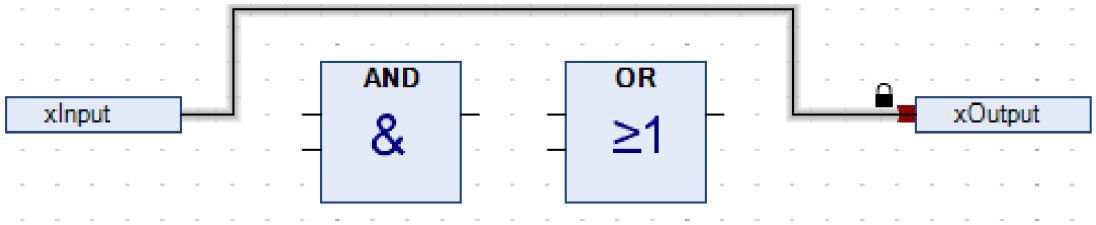
3. Click *"CFC ➔ Routing ➔ Route All Connections"*.
   ⇨ The collision is resolved.



4. Change the connecting lines gradually.

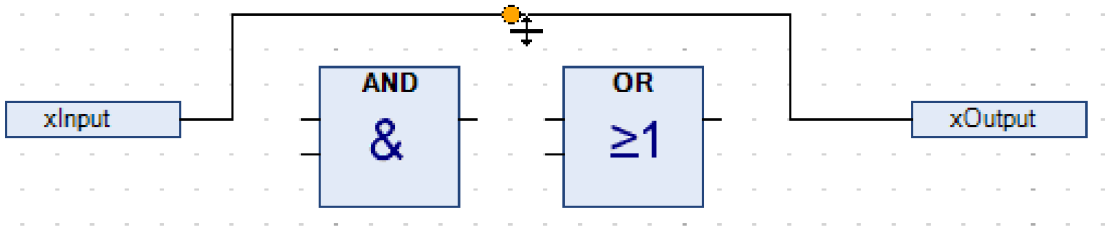


⇨ The connecting line has been changed manually and is now blocked for auto-routing. This is shown by a lock symbol at the end of the connection.



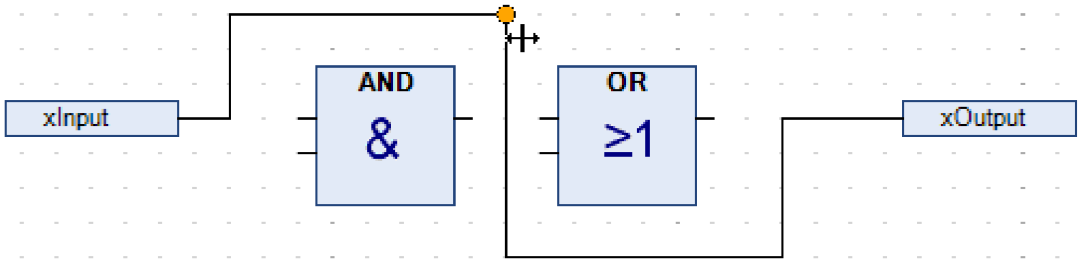
5. Select the connecting line and click *"CFC ➔ Routing ➔ Create Control Point"*.

⇨ A control point is created on the connecting line. The connecting line is fixed to the control point.



You can also drag a control point from the *"ToolBox"* view to a line.

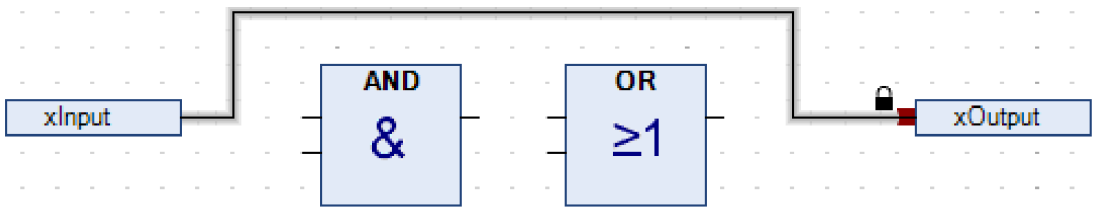
6. Change the connecting line as seen in the following example.



⇨ Use the control point for changing the connecting line according to your needs. You can set any number of control points.

7. In the context menu, click *"CFC ➔ Routing ➔ Remove Control Point"* to remove the control point.
8. Unlock the connection by clicking *"Unlock Connection"* or by clicking the lock symbol.
9. Select the connecting line and click *"Route All Connections"*.

⇨ The connecting line is routed automatically as seen in Step 3.

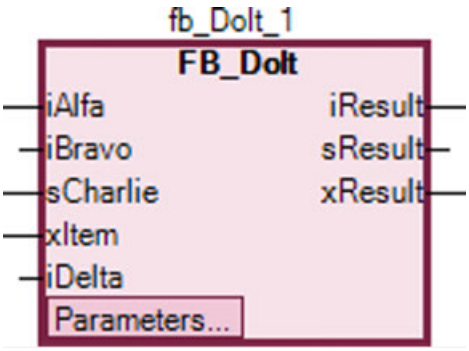


NOTICE!
Connections in a group are not auto-routed.

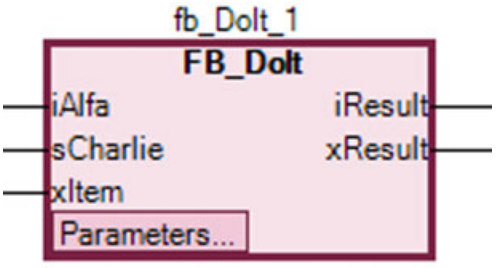
Reducing the display of a POU

☑ Requirement: A CFC POU is open. In the editor, its POUs with all declared pins are displayed.

1. Select a POU whose pins are partially disconnected.
   ⇨ Example: `fb_DoIt_1`



   The POU needs space for all of the pins.
2. Click *"CFC ➔ Pins ➔ Remove Unused Pins"*.
   ⇨ Now the POU needs less space and is displayed only with the functionally relevant pins.



See also

- *Chapter 1.4.1.19.1.2 "Common functions in graphical editors" on page 462*
- *Chapter 1.4.1.19.1.6.1 "CFC Editor" on page 511*
- *Chapter 1.4.1.19.1.6.2 "CFC editor, page-oriented" on page 514*
- *Chapter 1.4.1.19.1.6.5 "Elements" on page 522*

Structured Text (ST), Extended Structured Text (ExST)

The ST editor is used for the programming of POUs in the IEC-61131-3 programming language Structured Text (ST) and Extended Structured Text. The Extended Structured Text offers some additional functions with regard to the IEC 61131-3 standard.

Structured Text is a programming language, comparable with other high-level languages such as C or PASCAL, which permits the development of complex algorithms. The program code consists of a combination of **expressions** and **instructions**, which can also be executed conditionally (IF… THEN… ELSE) or in loops (WHILE… DO).

An expression is a construct that returns a value following its evaluation. Expressions are also operators and operands together. You can also use assignments as expressions. An operand can be a constant, a variable, a function call or a further expression.

Instructions control how the expressions are to be processed.

For this text editor you can make various settings with regard to behavior, appearance and menus in the dialog boxes *"Options"* and *"Adapt"* in the *"Tools"* menu. The familiar Windows functions (for example IntelliMouse) are also available for this editor.

See also

- ↳ *Chapter 1.4.1.8.3.3.1 "Programming structured text (ST)" on page 254*
- ↳ *Chapter 1.4.1.20.4.13.25 "Dialog 'Options' - 'Text Editor'" on page 1203*
- ↳ *Chapter 1.4.1.20.4.14.1 "Dialog 'Customize' - 'Menu'" on page 1206*

ExST - Extended structured text

Extended Structured Text (ExST) is a CODESYS-specific extension of the IEC 61131-3 standard for Structured Text (ST).

See also

- ↳ *Chapter 1.4.1.19.1.3.4.4 "ExST assignment 'R='" on page 466*
- ↳ *Chapter 1.4.1.19.1.3.4.3 "ExST assignment 'S='" on page 465*
- ↳ *Chapter 1.4.1.19.1.3.4.5 "ExST – Assignment as expression" on page 467*

Programming structured text (ST)

Principle

The programming languages 'Structured Text' and 'Extended Structured Text' are programmed in the ST editor. The program code consists of a combination of expressions and instructions, which can also be executed conditionally or in loops. You must conclude each instruction with a semicolon `;`.

The variables are declared in the declaration editor.

Creating a POU in the structured text (ST) implementation language

1. Select an application in the device tree.
2. Select the command *"Project ➔ Add Object ➔ POU"*.
   ⇨ The dialog box *"Add POU"* opens.
3. Enter a name and select the *"Implementation language""Structured Text (ST)"*. Click on *"Add"*.
   ⇨ The POU is added to the device tree and opened in the editor.

   Now insert the variable declarations in the upper part of the POU and enter the ST program code in the lower part of the POU.

See also

- ↳ *Chapter 1.4.1.19.1.3.3 "ST expressions" on page 464*
- ↳ *Chapter 1.4.1.19.1.3 "Structured Text and Extended Structured Text (ExST)" on page 463*
- ↳ *Chapter 1.4.1.19.1.3.5.10 "ST function block call" on page 474*

- *Chapter 1.4.1.19.1.3.5.11 “ST – Comments” on page 475*
- *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226*

Sequential Function Chart (SFC)

Use the SFC editor for programming POUs in the IEC 61131-3 compliant SFC implementation language. SFC is a graphical programming language for describing the chronological sequence of individual actions in a program. For this purpose, actions (discrete programming objects) are assigned to step elements. Transition elements control the processing order of steps.

See also

- *Chapter 1.4.1.19.1.4.1 “SFC editor” on page 476*

Programming in SFC

Creating a POU in SFC

1. Select an application in the device tree.
2. Click *“Project ➔ Add Object ➔ POU”*.
   ⇨ The *“Add POU”* dialog opens.
3. Specify a name and select the *“Sequential Function Chart (SFC)”* implementation language.
   Click *“Add”*.
   ⇨ CODESYS adds the POU to the device tree and opens it in the editor.

Adding a step-transition

1. Select the transition after the initial step.
   ⇨ The transition is marked in red.
2. Click *“SFC ➔ Insert Step-Transition After”*.
   ⇨ CODESYS inserts the *“Step0”* step and the *“Trans0”* transition.
3. Select the *“Trans0”* transition and click *“SFC ➔ Insert Step-Transition”*.
   ⇨ CODESYS inserts the *“Trans1”* transition and the *“Step1”* step before the *“Trans0”* transition.

You can also drag the *“Step”* and *“Transition”* elements into the diagram from the *“Toolbox”* view.

See also

- *Chapter 1.4.1.19.1.4.8.1 “SFC elements 'Step' and 'Transition'” on page 486*
- *Chapter 1.4.1.20.3.11.6 “Command 'Insert Step-Transition'” on page 1081*
- *Chapter 1.4.1.20.3.11.7 “Command 'Insert Step-Transition After'” on page 1081*

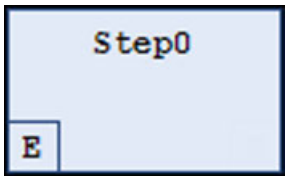
Adding an entry action

1. Select the *“Step0”* step.
2. Click *“SFC ➔ Add Entry Action”*.
   ⇨ By default, you are prompted to select the duplication mode for the step actions. You decide whether the reference information about the existing step action objects is copied when the step is copied, or the objects are embedded. Embedding results in new step action objects being created when the step is copied. The duplication mode is defined in the *“Duplicate when copying”* step property. When this property is deactivated, the copied steps call the same actions as the current step.

   You can deactivate the prompt completely in the SFC properties.

   The display of embedded objects in the *“Devices”* and *“POUs”* views can be deactivated by means of a menu command.
3. For this example, accept the *“Copy reference”* default setting and click *“OK”* to confirm.
   ⇨ The *“Add Entry Action”* dialog opens.
4. Enter the name "Step0_entry" and select the *“Structured Text (ST)”* implementation language. Click *“Add”*.
   ⇨ CODESYS inserts the *“Step0_entry”* action below the POU in the device tree and opens the action in the editor.

   In the `Step0_entry` entry action, you program statements to be executed one time when the *“Step0”* step becomes active.
5. Close the editor of `Step0_entry`.
   ⇨ The *“Step0”* step is now marked with an *“E”* in the lower left corner. Double-click this marker to open the editor.

   The entry action `Step0_entry` is now available in the properties of the step in *“Entry action”*. Other actions can also be selected there as needed.
6. Select the *“Step0”* step. Press *[Ctrl]+[V]* to copy the step.
   ⇨ The same entry actions inserted above are available In the inserted copy of the step. The new step then calls the same exact action.

See also

- *Chapter 1.4.1.20.3.11.8 “Command 'Add Entry Action'” on page 1082*
- *“2. Step actions” on page 489*
- *Chapter 1.4.1.20.4.13.22 “ Dialog 'Options' - 'SFC Editor'” on page 1200*

Adding an exit action

1. Select the *“Step0”* step.
2. Click *“SFC ➔ Insert Exit Action”*.
   ⇨ By default, you are prompted to select the duplication mode for the step actions of the step. See above for adding an entry action. Then the *“Insert Exit Action”* dialog opens.

3. Enter the name "Step0_exit" and select the *"Structured Text (ST)"* implementation language. Click *"Add"*.
   ⇨ CODESYS inserts the *"Step0_exit"* action below the POU in the device tree and opens the action in the editor.
   In the `Step0_exit` exit action, you program statements to be executed one time before the *"Step0"* step becomes inactive.
4. Close the editor of `Step0_exit`.
   ⇨ The *"Step0"* step is now marked with an *"X"* in the lower right corner. Double-click this marker to open the editor.

You can define the exit action in the properties of the step in *"Exit action"*. Other actions can also be selected there.

See also

- *Chapter 1.4.1.20.3.11.9 "Command 'Add Exit Action'" on page 1082*
- *"2. Step actions" on page 489*

Adding an action

1. Double-click the *"Step0"* step.
   ⇨ By default, you are prompted to select the duplication mode for the step actions of the step. See above for adding an entry action. The *"Add Action"* dialog opens.
2. Type in the name "Step0_active" and select the *"Structured Text (ST)"* implementation language. Click *"Add"*.
   ⇨ CODESYS inserts the *"Step0_active"* action below the POU in the device tree and opens the action in the editor.
   In the `Step0_active` step action, you program statements to be executed as long as the step is active.
3. Close the editor of `Step0_active`.
   ⇨ The *"Step0"* step is now marked with a black triangle in the upper right corner.

You can define the action in the properties of the step in *"Step action"*. Other actions can also be selected there.

See also

- *"2. Step actions" on page 489*

Adding an alternative branch

1. Select the *“Step1”* step.
2. Click *“SFC ➔ Insert Branch Right”*.
   ⇨ CODESYS inserts the *“Step2”* step to the right of *“Step1”*. The steps are connected as a parallel branch signified by two pairs of double lines.
3. Select one of the double lines.
   ⇨ The double line is marked red.
4. Click *“SFC ➔ Alternative”*
   ⇨ CODESYS converts the branch into an alternative branch. The double lines change into a single line.

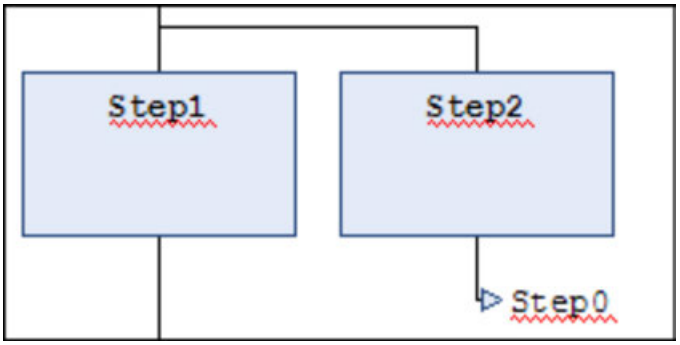
You can click *“SFC ➔ Parallel”* to convert an alternative branch into a parallel branch.

See also

- *Chapter 1.4.1.19.1.4.8.3 “SFC element 'Branch'” on page 491*
- *Chapter 1.4.1.20.3.11.10 “Command 'Parallel'” on page 1082*
- *Chapter 1.4.1.20.3.11.12 “Command 'Insert Branch'” on page 1083*
- *Chapter 1.4.1.20.3.11.13 “Command 'Insert Branch Right'” on page 1083*

Adding a jump

1. Select the *“Step2”* step.
2. Click *“SFC ➔ Insert Jump After”*.
   ⇨ CODESYS inserts the *“Step”* jump after the *“Step2”* step.
3. Select the *“Step”* jump destination.
   ⇨ You can type the jump destination manually or select it by using the Input Assistant [...]. Select `Step0`.



See also

- *Chapter 1.4.1.19.1.4.8.4 “SFC element 'Jump'” on page 492*
- *Chapter 1.4.1.20.3.11.16 “Command 'Insert Jump'” on page 1085*
- *Chapter 1.4.1.20.3.11.17 “Command 'Insert Jump After'” on page 1085*

Adding a macro

1. Select the *“Step1”* step.
2. Click *“SFC ➔ Insert Macro After”*.
   ⇨ CODESYS inserts the *“Macro0”* macro after the *“Step1”* step.

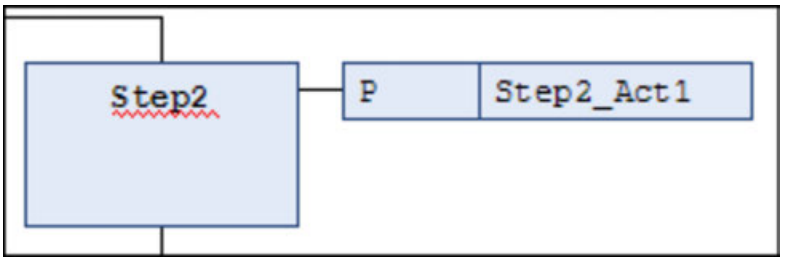
3. Double-click the *“Macro0”* element.
   ⇨ The macro opens in the implementation section of the editor. The name "Macro0" is displayed in the caption.
4. Click *“SFC ➔ Insert Step-Transition”*.
   ⇨ CODESYS inserts a step-transition combination.
5. Click *“SFC ➔ Zoom out of Macro”*.
   ⇨ The implementation section returns to the main diagram.

See also

- *Chapter 1.4.1.19.1.4.8.5 “SFC element 'Macro'” on page 492*
- *Chapter 1.4.1.20.3.11.18 “Command 'Insert Macro'” on page 1086*
- *Chapter 1.4.1.20.3.11.19 “Command 'Insert Macro After'” on page 1086*

Adding an association

1. Select the *“Step2”* step.
2. Click *“SFC ➔ Insert Action Association”*.
   ⇨ CODESYS inserts an association to the right of the *“Step2”* step.
3. Click in the left field of the association to select the qualifier.
   ⇨ You can enter the qualifier manually or use the Input Assistant ... . Select "P".
4. Click in the right field of the association to select the action.
   ⇨ You can type the action or select it by using the Input Assistant ... .



See also

- *“1. IEC actions” on page 488*
- *Chapter 1.4.1.19.1.4.4 “Qualifiers for Actions in SFC” on page 479*
- *Chapter 1.4.1.20.3.11.14 “Command 'Insert Action Association'” on page 1084*
- *Chapter 1.4.1.20.3.11.15 “Command 'Insert Action Association After'” on page 1085*

Using the library 'analyzation.library' for the analyzation of expressions

The library `analyzation.library` allows for the analyzation of expressions. It can be used, for example, in the SFC diagram to examine the result of the flag `SFCError`. This flag is used to monitor timeouts in the SFC diagram.

See also

- *Chapter 1.4.1.19.1.4.7 “Library "Analyzation"” on page 485*

See also

- *Chapter 1.4.1.19.1.4.1 “SFC editor” on page 476*

1.4.1.8.4 Function block — Calling functions or methods with external implementation

A runtime system can include the implementation of a function block, function, or method (for example, from a library). If you create a POU in your application with the same name by using the *"External implementation"* without an implementation, then you can execute the existing implementation. Please make sure that you declare local variables only in an external function block. External functions and methods must not contain local variables.

When the application is downloading, CODESYS searches for and links the associated implementation in the runtime system for each external POU.

Objects with the property "External implementation" are postfixed with `(EXT)` *after the object name in the "Devices" or "POUs" view.*

See also

- ↳ *Chapter 1.4.1.20.4.10.4 "Dialog 'Properties' - 'Build'" on page 1159*

Creating POUs with external implementation

1. Click *"Project ➔ Add Object ➔ POU"*.
2. Activate *"Function block"*, Method, or *"Function"* and specify the name of the associated implementation of the runtime system. Close the dialog box by clicking *"Add"*.
   ⇨ The runtime system POU is created in the *"POUs"* view. The name is postfixed with `(EXT)`.
3. Right-click the POU and select *"Properties"*.
   ⇨ The dialog box opens.
4. Click the *"Build"* tab.
5. Select the *"External implementation (Late link in the runtime system)"* check box.
   ⇨ The POU is declared and you can implement a POU call.

Creating methods with external implementation

1. Select a function block in the device tree or in the POUs view.
2. Select *"Add Object ➔ Method"* and type the name of the associated implementation of the runtime system. Click *"Add"* to close the dialog box.
   ⇨ The method is created.
3. Right-click the method and select *"Properties"*.
   ⇨ The dialog box opens.
4. Click the *"Build"* tab.
5. Select the *"External implementation (Late link in the runtime system)"* check box.
   ⇨ The method is declared and you can implement a method call. The method name is postfixed with `(EXT)` in the *"Devices"* or *"POUs"* view.

1.4.1.8.5 Using input assistance

CODESYS provides tools and features to help you code when creating programs.

Input assistant

The input assistant provides all program elements that you can insert at the current cursor position. Open the *“Input Assistant”* dialog by clicking *“Edit ➔ Input Assistant”* or by pressing *[F2]*.

See also

- *“Dialog 'Input Assistant' - Tab 'Categories'” on page 978*

Dialog 'Auto Declare'

This dialog supports the declaration of variables.

See also

- *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*

"List components"

The "List components" function is an input tool in textual editors to help you input valid identifiers. Activate this function by clicking *“Tools ➔ Options”* and then the *“SmartCoding”* category. Select the *“List components after typing a dot (.)”* check box.

- If you type a dot (`.`) instead of a global variable, then a drop-down list opens with all available global variables. You insert the selected variable after the dot by double-clicking a variable in the drop-down list or by pressing *[Enter]*.
- If you type a dot (.) instead of a global variable after a function block instance variable or a structure variable, then CODESYS opens a drop-down list with all global variables, all input and output variables for the function block, or all structure members.

  You insert the selected variable after the dot by double-clicking a variable in the drop-down list or by pressing *[Enter]*.

  Note: When you also want to choose from the local variables of function block instances, select the *“Show all instance variables in input assistant”* option in the CODESYS options (SmartCoding category).
- If a component access (with a dot) for a drop-down list has already happened, then the last selected entry is preselected at the next component access.
- When you type any sequence of characters and then press *[Ctrl]+[Space]*, a drop-down list opens with all available POUs and global variables. The first element in this list that starts with the sequence of characters is selected by default and you can insert it by double-clicking it or by pressing *[Enter]*.

  Matches with the entered character string are highlighted in yellow in the drop-down list.

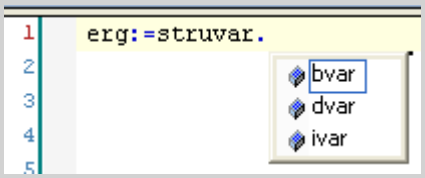
  If the entered character string is changed, then the displayed drop-down list is refreshed.
- In the ST editor, you can filter the displayed drop-down list by scopes:

  Depending on the displayed drop-down list, you can use the *[Arrow right]* and *[Arrow left]* keys to toggle between the following drop-down lists:
  - *“All items”*
  - *“Keywords”*
  - *“Global declarations”*
  - *“Local declarations”*
- CODESYS displays a tooltip if you type an opening parenthesis for a POU parameter when calling a function block, a method, or a function. This tooltip includes information about the parameters as they are declared in the POU. The tooltip remains open until you click to close it or you change the focus away from the current view. If you accidentally close the tooltip, then you can reopen it by pressing *[Ctrl]+[Shift]+[Space]*.

You can use the pragma attribute `'hide'` for excluding variables from the "List components" feature.

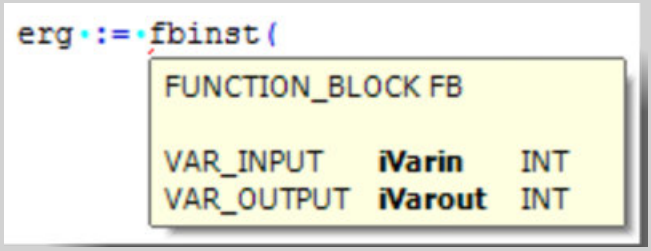
Examples

Typing structure variables:



list

Calling a function block:



See also

- *Chapter 1.4.1.20.4.13.23 “Dialog 'Options' - 'SmartCoding'” on page 1201*
- *Chapter 1.4.1.19.6.2.16 “Attribute 'hide'” on page 700*

Short form feature

The short form feature allows you to type abbreviated forms for variable declarations in the declaration editor and in textual editors where variables declarations are possible. Use this feature by pressing *[Ctrl]+[Enter]* to end a declaration line.

CODESYS supports the following short forms:

- All identifiers become variable identifiers except the last identifier of a line.
- The data type of the declaration is determined by the last identifier of the line. The following applies:
  - `B` or `BOOL` yields `BOOL`
  - `I` or `INT` yields `INT`
  - `R` or `BOOL` yields `BOOL`
  - `S` or `STRING` yields `STRING`
- If a data type is not defined using this rule, then the data type is automatically `BOOL`, and the last identifier is not used as the data type (see Example 1).
- Depending on the type of declaration, every defined constant becomes an initialization or string length definition (see Example 2 and 3).
- An address, such as `%MD12`, is automatically extended with the AT attribute (see Example 4).
- Any text after a semicolon (`;`) is converted into a comment (see Example 3).
- All other characters in the line are ignored (see exclamation mark in Example 5).

Examples

Example	Short Form	Resulting declaration
1	`A`	`A: BOOL`
2	`A B I 2`	`A, B: INT := 2;`
3	`ST S 2; A string`	`ST:STRING(2); (* A string *)`
4	`X %MD12 R 5 Real Number`	`X AT %MD12: REAL := 5.0;(* Real Number *)`
5	`B !`	`B: BOOL`

See also

- *Chapter 1.4.1.8.2 “Declaration of Variables ” on page 222*

Smart tag functions

Smart tags make it easier to write program code by suggest appropriate commands directly at the programming element. When you place the cursor over a programming element that has a smart tag function, the symbol appears. When you click the symbol, the commands that you can choose from are shown. Available smart tags:

- The smart tag function provides the *“Declare Variable”* command for undeclared variables in the implementation part of the ST editor.

See also

- *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*

1.4.1.8.6 Using Pragmas

Pragma in CODESYS

A pragma is a text in the source code of the application that is enclosed in curly brackets. Pragmas are used to insert special statements in the code, which the compiler can evaluate. This allows a pragma to influence the properties of one or more variables with respect to precompilation or compilation (code generation). Pragmas that the compiler does not recognize are passed over as a comment.

The statement string of a pragma can also extend over multiple lines. For more details about the syntax, see the descriptions of the individual CODESYS pragmas.

There are different pragmas for different purposes (example: initialization of a variable, monitoring of a variable, adding a variable to the symbol configuration, forcing the display of messages during the compilation process, and behavior of a variable under certain conditions).

NOTICE!
Uppercase and lowercase characters have to be respected.

Examples

```
{warning 'This is not allowed'}

{attribute 'obsolete' := 'datatype fb1 not valid!'}

{attribute 'Test':='TestValue1;
                    TestValue2;
                    TestValue3'}
```

Possible insertion positions

NOTICE!
Pragmas in CODESYS are not one-to-one implementations of C preprocessor directives. You have to position a pragma like an ordinary statement. You must not use a pragma within an expression.

A pragma that the CODESYS compiler should evaluate can be inserted at the following positions:

- In the declaration part of a POU:
  - In the textual declaration editor, specify pragmas directly as line(s), either at the beginning of the POU or before a variable declaration.
  - In the tabular editor, you specify pragmas that should be located before the first declaration line in the *“Edit Declaration Part”* / *“Attributes”* dialog.
- In a global variable list
- In the implementation part of a POU:
  - The pragma has to be at a "statement position", meaning at the beginning of a POU on a separate line, or after a "`;`" or `END_IF`, `END_WHILE`, etc.
  - FBD/LD/IL editor: In networks of the FBD/LD/IL editor, you insert pragmas like a label by means of the *“FBD/LD/IL ➔ Insert Label”* command. Then, in the text field of the label with the corresponding pragma statement, replace the default text *“Label:”*. To use a pragma in addition to a label, you specify the pragma first and then the label.

Incorrect and correct positions for a conditional pragma

INCORRECT:

```
{IF defined(abc)}
IF x = abc THEN
{ELSE}
IF x = 12 THEN
{END_IF}
y := {IF defined(cde)} 12; {ELSE} 13; {END_IF}
END_IF
```

CORRECT:

```
{IF defined(abc)}
IF x = abc THEN
{IF defined(cde)}
    y := 12;
{ELSE}
   y := 13;
{END_IF}
END_IF
{ELSE}
IF x = 12 THEN
{IF defined(cde)}
    y := 12;
{ELSE}
   y := 13;
{END_IF}
END_IF
{END_IF}
```

In the “Properties” dialog (“Compile” category), you can specify "defines" that can be queried in pragmas.

Scope:

Depending on the type and contents of a pragma, it may influence the following:

- Subsequent declarations
- Exactly the next statement
- All subsequent statements until it is canceled by a corresponding pragma
- All subsequent statements until the same pragma is executed with other parameters or the end of the code is reached. In this context, "code" means the declaration part, implementation part, global variable list, and type declaration. Therefore, a pragma influences the entire object when the pragma is alone on the first line of the declaration part and is not superseded or canceled by another pragma.

Pragma categories in CODESYS

The CODESYS pragmas are divided into the following categories:

- Attribute pragmas (influence compiling and precompiling)
- Message pragmas (print user-defined messages when compiling)
- Conditional pragmas (influence code generation)
- User-defined pragmas

See also

- ↳ *Chapter 1.4.1.8.2.2 “Using the 'Declare variable' dialog box” on page 227*
- ↳ *Chapter 1.4.1.19.6.2 “Attribute Pragmas” on page 685*
- ↳ *Chapter 1.4.1.19.6.1 “Message Pragmas” on page 683*
- ↳ *Chapter 1.4.1.19.6.3 “Conditional Pragmas” on page 732*

1.4.1.8.7 Using Library POUs

Libraries are collections of objects that you can link to your application. You can use the objects contained in libraries in exactly the same way as objects that you have defined in the project.

Libraries can contain the following objects:

- POUs (for example function blocks, or functions)
- Interfaces and their methods and attributes
- Data types (for example enumerations, structures, aliases, and unions)
- Global variables, constants, and parameter lists
- Text lists, image pools, visualizations, and visual elements
- External files (for example, documentation)
- Cam plate tables

Libraries in a project are managed in the Library Manager. You use the dialog of the library repository to perform the previous installation of the library on the system.

For "visibility" of library POUs and namespaces of libraries, see also the help page for the library properties.

See also

- ↳ *Chapter 1.4.1.20.3.14.3 “Command 'Properties'” on page 1118*
- ↳ *Chapter 1.4.1.16 “Using Libraries” on page 448*
- ↳ *Chapter 1.4.1.20.3.14.3 “Command 'Properties'” on page 1118*

Using library POUs

The following instructions describe the example of how to insert the counter POU `CTUD` from the library `Standard` into your program.

1. Open a POU in the editor and place the cursor in the declaration part.
2. Specify the name for the function block instance, followed by a colon (example: `iCounter1:`).
3. Press *[F2]* to open the Input Assistant.
4. In the category *“Structured Types”*, select the `CTUD` function block from the `Standard` library (subfolder "Counter").

   Select the *“Insert with namespace prefix”* option.
5. Click *“OK”* to exit the dialog.

   ⇨ The function block is inserted with a namespace prefix into the declaration part: `iCounter1:Standard.CTUD`.

See also

- *Chapter 1.4.1.16 “Using Libraries” on page 448*

1.4.1.8.8 Managing text in text lists

Text lists are used for preparing visualization texts is multiple languages. You can specify the texts in Unicode format so that all languages and characters are possible. You can export text lists and then translate the texts outside of the current project.

CODESYS differentiates between static text (managed in the *“GlobalTextList”* object) and dynamic text (managed in objects of type *“TextList”*. Static texts exist in the visualization and can change only the displayed language while in runtime mode. The text ID stays the same. Dynamic texts can be controlled by means of an IEC variable that contains the text ID. In this way, you can display varying text in a visualization element in runtime mode. For example, you can configure a text field so that is shows an error text for an error number.

Both text list types include a table with text entries. An entry consists of an ID for identification, the output text, and its translation. In a text list or global text list, you can translate an output text in any number of languages. The translations are the basis for the language selection and the language switch in visualizations.

Adding a language and translating text

Requirement: A project is open with a text list or global text.

1. Double-click an object of type *“TextList”* or *“GlobalTextList”* in the device tree or POUs view.
   ⇨ The *“Textlist”* menu is shown in the menu bar and the text list opens in the editor.
2. Click *“Textlist ➔ Add Language”*.
3. Specify a name for the language (example: `en-US`). Click *“OK”* to close the dialog.
   ⇨ A column is displayed with the heading `en-US`.
4. Type in the translation of the source text into the column.

You can correct the name of a language in the table by means of the command “Rename Language” in the context menu of the text list.

Exporting a text list

☑ Requirement: A project is open with a text list or global text.

1. Double-click the object *“GlobalTextList”* or an object of type *“TextList”*.
   ⇨ The object opens.
2. Click *“Textlist ➔ Import/Export Text Lists”*.
   ⇨ The *“Import/Export”* dialog opens.
3. At *“Choose export file”*, click for more (...) and select the directory and file name (example: `Text_lists_exported`).
4. Select the *“Export”* option.

5. Click *"OK"* to close the *"Import/Export"* dialog.

   ⇨ CODESYS exports to a file the text list entries of all text lists of the project. The table contains a column with the text list names.

Example

Contents of the file `Text_lists_exported`

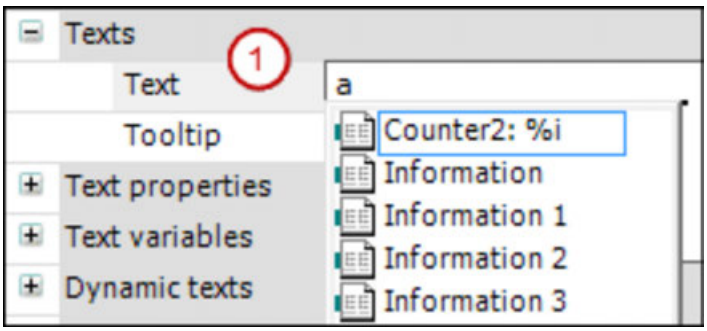
```
TextList    Id    Default    en_US
Text_list_A    A    Information A    Infrrmation A_en
Text_list_A    B    Information B: OK    Information B_en: OK
Text_list_A    C    Information C    Information C_en
Text_list_A    D    Information D    Information D_en
Text_list_A    E    Information E    Information E_en
Text_list_A    F    Information F    Information F_en
AlarmGroup    2    Warning 2
AlarmGroup    1    Warning 1
GlobalTextList        Information B    Information B_en
GlobalTextList        Information A    Information A_en
GlobalTextList        Switch    Switch
GlobalTextList        Counter: %i    Counter : %i
```

Preparing the exported file for input assistance

☑ Requirement: A file is created (example : `Text_lists_exported`) by means of the command *"Import/Export Text Lists"*. It contains the texts of the text lists of the project.

1. Click *"Tools* ➔ *Options"*, *"Visualization"* category, *"File Options"* tab.
2. Click … in *"Text file for textual "List components" "* and select a file (example: `Text_lists_exported`). Click *"OK"* to close the dialog.

   ⇨ When you specify a static text in the *"Texts"* property for an element in a visualization, CODESYS offers the source text of the file as input assistance when typing in the first letter.



(1): *"Texts"*, *"Text"*

See also

- ↳ *""List components"" on page 261*

Importing files with text list entries

A file to be imported has the `.csv` format. The first line is a header (example: `TextList Id Default en_US`). The other lines contain text list entries. You get this kind of file by exporting the text lists of the project to a file. There you can edit the text list entries and then import the file outside of CODESYS. When importing, CODESYS handles the text list entries differently for the `GlobalTextList` and for dynamic text lists.

`GlobalTextList`

- CODESYS does not create new text list entries for an unknown ID.
- CODESYS ignores changes that affect the ID or the source text.
- CODESYS accepts changes in the translations.

TextList

- For a new ID, CODESYS supplements the corresponding text list with a text list entry.
- For an existing ID that does not agree in the source text, the source text of the text list is overwritten with the source text of the file.
- CODESYS accepts changes in the translations.

Importing a file

☑ Requirement: A project is open with a text list or global text.

1. Double-click the object *"GlobalTextList"* or an object of type *"TextList"*.
   ⇨ The object opens.
2. Click *"Textlist ➔ Import/Export Text Lists"*.
   ⇨ The *"Import/Export"* dialog opens.
3. In the *"Choose file to compare or to import"* input field, click for more (...) and select the directory and file (example: `Text_lists_corrected.csv`).
4. Select the *"Import"* option.
5. Click *"OK"* to close the dialog.
   ⇨ CODESYS imports the text list entries of the file into the respective text lists.

Example

Contents of the file `Text_lists_corrected.csv`

```
TextList    Id    Default    en_US
Text_list_A    A    Information A    Information A2_en
Text_list_A    B    Information B: OK    Information B_en: OK
Text_list_A    C    Information C    Information C_en
Text_list_A    D    Information D    Information D_en
Text_list_A    E    Information E    Information E_en
Text_list_A    F    Information F    Information F_en
Text_list_A    G    Information G    Information G_en
AlarmGroup    2    Warning 2
AlarmGroup    1    Warning 1
GlobalTextList        Information B    Information B_en
GlobalTextList        Information A    Information A_en
GlobalTextList        Switch    Switch
GlobalTextList        Counter: %i    Counter : %i
```

These contents are applied to the text lists with the same name in the project.

See also

- *Chapter 1.4.1.20.3.20.6 "Command 'Import/Export Text Lists'" on page 1133*

Comparing text lists with a file and exporting differences

☑ Requirement: A project is open with a text list or global text.

1. Double-click the object *"GlobalTextList"* or an object of type *"TextList"*.
   ⇨ The object opens.

2. Click *"Textlist ➔ Import/Export Text Lists"* in the context menu.
   ⇨ The *"Import/Export"* dialog opens.
3. In the *"Choose file to compare or to import"* input field, click for more (...) and select the directory and file name of the comparison file (example: `Text_lists_corrected.csv`).
4. For *"Choose export file"*, click ... and select the directory and file that contains the comparison result.
5. Select the *"Export only text differences"* option.
6. Click *"OK"* to close the dialog.
   ⇨ CODESYS reads the import file and compares the text list entries that have the same ID. If they do not agree, then CODESYS writes the text list entries of the text list to the export file.

   For the global text list, CODESYS compares the translations of the same source texts. If they do not agree, then CODESYS writes the text list entries to the export file.

See also

- *Chapter 1.4.1.20.3.20.1 "Command 'Add Language'" on page 1132*
- *Chapter 1.4.1.20.3.20.6 "Command 'Import/Export Text Lists'" on page 1133*
- *Chapter 1.4.1.20.3.20.7 "Command 'Remove Language'" on page 1134*
- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Managing static text in global text lists

The global text list is the central location for texts that are displayed in the visualization.

When you configure a text for the first time in visualization element, CODESYS creates the global text list. CODESYS fills in the table as you create more texts. Therefore, the table includes all texts automatically that you create in the project visualizations. CODESYS assigns incremental IDs as integers, beginning at 0.

You can check, update, and compare the global text list with the static texts of the visualization. You cannot edit the source text or the ID directly in the table. However, you can replace a source text with another source text by creating and importing a replacement file. Menu commands are provided for this purpose.

Configuring visualization elements with static text

A text in a *"GlobalTextList"* can contain a format definition.

☑ Requirement: A project is open with a visualization. The *"GlobalTextList"* object contains the texts that are defined in the project visualizations.

1. Double-click the visualization.
   ⇨ The editor opens.
2. Select an element with the *"Text"* property (example: *"Text field"*).
3. Type in some text in the *"Text"* property (example: `Static Information A`).
   ⇨ CODESYS adds the text to the global text list in the POU view.

Checking the global text list

☑ Requirement: A project is open with a visualization. The *"GlobalTextList"* object contains the texts that are defined in the project visualizations.

1. Double-click the *"GlobalTextList"* object in the POUs tree.
   ⇨ The table opens with the static texts.
2. Click *"Text List ➔ Check Visualization Text IDs"*.
   ⇨ CODESYS reports when a source text of the text list does not match the static text that is identified by the ID. The source text in the global text list and the text in the visualization with the same ID do not match.

Updating IDs of the global text list

☑ Requirement: A project is open with a visualization. The *"GlobalTextList"* object contains the texts that are defined in the project visualizations.

1. Double-click the *"GlobalTextList"* object in the POUs tree.
   ⇨ The list opens with the text list entries.
2. Click *"Text List ➔ Update Visualization Text IDs"*.
   ⇨ CODESYS adds text to the global text list when a text in the *"Static Text"* property does not match the source text in the project visualizations.

Removing the global text list and creating current IDs again

☑ Requirement: A project is open with a visualization. The *"GlobalTextList"* object contains the texts that are defined in the project visualizations.

1. Right-click the *"GlobalTextList"* object in the POUs tree and select the *"Delete"* command.
   ⇨ The object is removed.
2. Open a visualization.
3. Click *"Visualization ➔ Create Global Text List"*.
   ⇨ In the POU view, a new *"GlobalTextList"* object is created. The global text list contains the static text from the existing project visualizations.

Removing IDs from the global text list

☑ Requirement: A project is open with a visualization. The *"GlobalTextList"* object contains the texts that were defined in the project visualizations.

1. Double-click the *"GlobalTextList"* object in the POUs tree.
   ⇨ The table opens with the texts.
2. Click *"Text List ➔ Remove Unused Text List Entries"*.
   ⇨ CODESYS removes the text list entries with IDs not referenced in the project visualizations.

Updating the global text list with a replacement file

A replacement file has the `CSV` format. The first row is a header: `defaultold defaultnew REPLACE`. The following rows contain the old source texts, the new source texts, and then the `REPLACE` command. Tabs, commas, and semicolons are permitted separators. A combination of separator characters in a file is not permitted.

Example (tab as separator character)

```
defaultold      defaultnew       REPLACE
Information A   Information A1   REPLACE
```

When you import a replacement file, CODESYS processes the replacement file row by row and performs the specified replacements in the *"GlobalTextList"*. In addition, CODESYS replaces the previous text with the replacement text in the visualizations. If the replacement text already exists as static text, then CODESYS recognizes this and harmonizes the static text and leaves only one text list entry.

☑ Requirement: A project is open with a text list or global text.

1. Double-click the *"GlobalTextList"* object.
   ⇨ The object opens.
2. Click *"Text List ➔ Import/Export Text Lists"*.
   ⇨ The *"Import/Export"* dialog opens.
3. At the *"Choose file to compare or to import"* input field, click for more (...) and select the directory and file (example: `ReplaceGlobalTextList.csv`).
4. Select the *"Import replacement file"* check box.
5. Click *"OK"* to close the dialog.
   ⇨ The texts in the text lists and the visualizations are replaced.

Example

The global text list contains the following source texts:

```
GlobalTextList        Counter: %i
GlobalTextList        Counter: %i
GlobalTextList        Information A
GlobalTextList        Information a
GlobalTextList        Information Aa
GlobalTextList        Switch
```

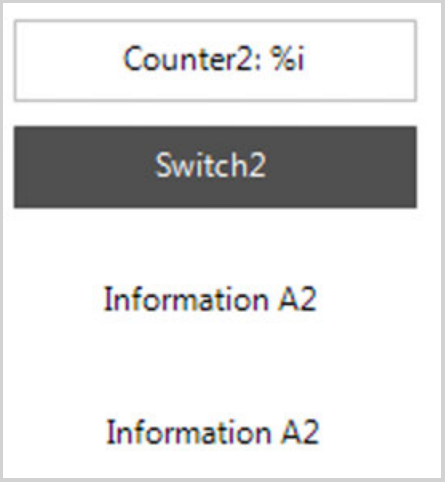
The replacement file contains the following replacements:

```
defaultold        defaultnew        REPLACE
Counter: %i        Counter2: %i    REPLACE
Counter:  %i        Counter2: %i    REPLACE
Information  A    Information A2    REPLACE
Information  a    Information A2    REPLACE
Information Aa    Information A2    REPLACE
Switch        Switch2            REPLACE
```

CODESYS detects duplicate text list entries and removes them. Afterwards, the global text list contains the following source texts:

5	Switch2
4	Counter2: %i
3	Information A2

The texts in the visualization have been replaced.



See also

- *Chapter 1.4.1.20.3.20.2 “Command 'Create Global Text List'” on page 1132*
- *Chapter 1.4.1.20.3.20.10 “Command 'Check Visualization Text IDs'” on page 1135*
- *Chapter 1.4.1.20.3.20.11 “Command 'Update Visualization Text IDs'” on page 1135*
- *Chapter 1.4.1.20.2.9 “Object 'GlobalTextList'” on page 871*

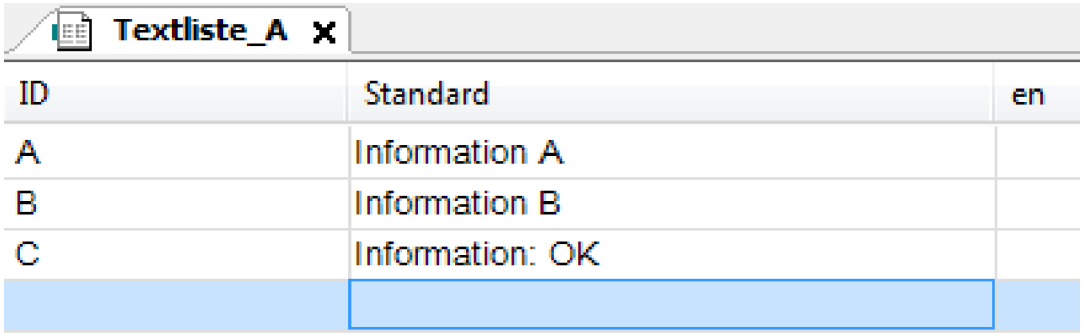
Managing dynamic text in text lists

You can create and translate texts in a text list for dynamic texts in order to show them dynamically in a visualization or in the alarm management. The object of type *"Text list"* can be located globally in the POUs view or below an application in the device tree. It contains a table with text list entries that you can edit and extend. A text list entry consists of an ID for identification, the output text, and its translation. You can add new text list entries to a text list. Menu commands are provided for this purpose.

Creating text lists for dynamic text output

Requirement: A project is open with a visualization.

1. Select an application in the POUs view or device tree and click *"Project ➔ Add Object"*.
2. Select *"Text list"*.
3. Type a name (example: `Textliste_A`). Click *"Add"* to close the dialog.
   ⇨ An object of type *"Text list"* is created.
4. Click below the *"Default"* column and open the input field. Type a text (example: `Information`).
   ⇨ The source text is created. It is used as a key in the table and as a source text for translations.
5. Type any string in the *"ID"* column (example: `A`).
   ⇨ A text list entry is defined with source text and ID. If you configure the *"Dynamic texts"* property of an element in a visualization, then you can select the text list `Textliste_A` and assign the ID `A`.
6. Double-click in the blank line at the end of the table below *"Default"* and type in more text list entries.



Displaying text dynamically

In a visualization, you can configure the dynamic output of texts that were created in a text list by configuring the *"Dynamic texts"* property of an element. You can directly assign a text list and an ID, as well as IEC variables, where you set the values programmatically.

☑ Requirement: A project with visualization is open and a text list is in the device tree.

1. Open the text list (example: `Text_list_A`).
2. Double-click the visualization.
   ⇨ The editor opens.
3. Drag an element to the visualization (example: a *"Text field"*).

4. Configure its *“Dynamic texts”* property by selecting one in the *“Text list”* property (example: `'Text_list_A'`) and add an ID from the text list into the *“Text index”* (example: `'A'`). **Pay attention to the single straight quotation marks**. You can also assign an IEC variable of type `STRING` for the text list name and ID.

   ⇨ The IEC variables allow for programmatic access to the texts of the text lists.

5. Build the application, download it to the controller, and start it.

   ⇨ The visualization shows the text from the text list in the text field : `Information A`.

See also

- ↳ *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

1.4.1.8.9 Using image pools

An image pool is a table of image files. CODESYS references image files for use in the project (for example, in a visualization) uniquely by the ID and name of the image pool. A project can include several image pools. You can create Image pools in the device tree below the application or in the POU pool. In a library project, you can use the object properties of an image pool to turn it into a symbol library for the visualization.

We recommend that you reduce the size of image files as much as possible integrating them. This will optimize the loading time of the visualization in every visualization type: TargetVisu, WebVisu and development system.

If you insert an image element into a visualization and enter an ID (*“Static ID”*) in the element properties, then CODESYS automatically creates a global image pool. CODESYS uses the default name *“GlobalImagePool”* for this.

Please note the following when the ID of an image file appears in several image pools.

- Search order: If you selected an image managed in *“GlobalImagePool”*, then you do not have to enter the name of the image pool. The search order for image files is as follows:
  - 1. GlobalImagePool
  - 2. Image pools assigned to the currently active application
  - 3. Image pools next to the GlobalImagePool in the POU window
  - 4. Image pools in libraries
- Unique access: You can reference a selected image directly and uniquely by appending the image ID to the name of the image pool in the following syntax "`<pool name>.<image ID>`.

See also

- ↳ *Chapter 1.4.1.20.2.13 “Object 'Image Pool'” on page 873*
- ↳ *Chapter 1.4.1.20.4.10.17 “Dialog 'Properties' - 'Image Pool'” on page 1168*

Creating image pools

1. Select the *“Application”* object in the device tree.

   Click *“Project ➔ Add Object ➔ Image Pool”*.

   ⇨ The *“Add Image Pool”* dialog box opens.

2. Type a name for the image pool (for example, "Images1") and click *“Add”*.

   ⇨ The image pool is added to the device tree.

3. Select the image pool object and open by choosing the command *“Project ➔ Edit Object”*.

4. Double-click the field in the *“ID”* column and assign an appropriate ID (for example, "Icon1").

   You can also add new images to the list by clicking *“Imagepool ➔ Add Image File”*.

5. Double-click the field in the *“File name”* column. Click for more settings (...).

   ⇨ The *“Select Image”* dialog box opens.

6. Click for more settings (...) and select the image file.

   ⇨ A thumbnail of the image file is displayed in the field of the column *“Image”*. The name of the file is displayed in the field of the column *“File name”*.

The image file can be references only by the name `Images1.Icon1`.

See also

- *Chapter 1.4.1.20.3.15.1 “Command 'Insert Image'” on page 1121*

Using image files in the 'Image' visualization element

When you insert an image element into a visualization, you can define the image type.

- Static image: Enter the image ID of the image file or the name of the image pool plus the image ID into the element configuration (property *“Static ID”*). Please note the comments for the search order and access.
- Dynamic image: Type the variable for defining the image file ID (for example, `PLC_PRG.imagevar`) in the element configuration (*“Bitmap ID variable”* property). You can exchange a dynamic element in online mode depending on a variable.

See also

- *Chapter 1.4.5.18.1.5 “Visualization Element 'Image'” on page 1418*

Using image files for the visualization background

You can set an image in the background definition of a visualization. You can define the image by the name of the image pool plus the filename, as described above for a visualization element.

See also

- *Chapter 1.4.5.19.2.10 “Command 'Background'” on page 1728*

1.4.1.8.10 Integrating C Modules

With the C code integration plugin, externally implemented C code files can be included in CODESYS projects and C stubs can be generated from IEC objects.

In CODESYS, the *“C Code Module”* object type is available for this purpose. The C code files and the used IEC objects are located below a *“C Code Module”*. A file directory on the hard disk with C code files is assigned to each C code module.

In the project, you can generate IEC objects from a C code file in the format *.h or *.hpp (header file) in order to use them in other POUs.

The generation of C-stubs is intended for the following use cases:

- A C code file accesses an IEC object: A C code file cannot access an IEC object directly. It can access only the C stub that was generated from the IEC object.
- Generation of precompiled modules that you can merge into a library project.

After being imported, the imported source code files are part of the CODESYS project and they are therefore decoupled from the original files on the disk.

During compilation, a dynamic module is generated from a C code module and saved as part of the project. Information, warnings, and errors are displayed in the message view in the *“C Code Module”* category.

All dynamic modules of an application are transferred and loaded to the runtime system during the download. The runtime system must support dynamic linking for this.

> ***License for the runtime system***
> *The runtime system requires a license that permits C modules to be loaded. Without this license, dynamic modules cannot be linked during the download, and therefore the download will be aborted.*

The dynamic modules are part of the boot application and they are reloaded and activated when the controller is restarted. The *"Reset Origin"* command unloads all C code modules in the application. The *"Reset Cold"* and *"Reset Warm"* commands do not lead to a repeated initialization of the C code modules.

> **NOTICE!**
> **No C code for simulation mode**
> In simulation mode, C code is not generated and loaded to the runtime system. To simulate the code contained in the C modules anyway, you can implement it for this purpose in the respective IEC objects of the C code module.

CODESYS does not support the monitoring of variables in C code files or the setting of breakpoints in C source code.

Precompiled module in a library:

C code integration provides the capability of assigning a precompiled runtime module (example: *.dll) in the library to a device and then to save it in the library. Then, these modules can be loaded dynamically.

See also

- *Chapter 1.4.1.10.3 "Handling of Device User Management" on page 385*

Configuring C code modules

Requirement: A project is open that already includes a C code module.

1. Click the object *"C Code Module"* in the device tree.
2. Select the command *"Properties"* in the context menu.
3. Open the *"Build"* tab in the *"Properties"* dialog.
4. Specify the file path of the Visual Studio installation on your computer. The input assistant (...) and the search tool (magnifying glass) are also available.
5. Specify the file path of the MS Windows SDK installation on your computer. The input assistant (...) and the search tool (magnifying glass) are also available.
6. Specify a file path for CODESYS to store the temporary compile files.

Importing folders with C source files from the file directory

Requirement: A project is open. The project controller supports the integration of C code.

1. Select *"Application"* in the device tree and click *"Project ➔ Add Object ➔ C Code Module"*.
2. If necessary, specify a new name for your C code module in the *"Add C Code Modules"* dialog. If you do not, then your object will be given the standard name *"C Code Module"*.
3. Click the symbol (...) next to the *"Source directory"* input field.
4. The *"Find Folder"* dialog opens.

5. In the *"Find Folder"* dialog, select the folder containing the C source files (`*.c`, `*.cpp`, `*.h`, or `*.hpp`).
6. When you select the *"Monitor folder for source code changes"* option, CODESYS displays a message when changes have been made to the C source files in the selected folder of the file system.
7. Click *"Add"*.
   ⇨ CODESYS inserts the C code module into the device tree with the folders *"Extensions"*, *"IEC interface"*, and *"Source Files"*.
8. In the device tree, click the plus symbol (*"+"*) of the *"Source Files"* folder.
   ⇨ The imported C source files are listed in the open folder.
9. If you double-click one of the C source files (), then the C code file opens in your editor.

Importing individual C code files

Requirement: A project is open that already includes a C code module.

1. Click the object *"C Code Module"* in the device tree.
2. Click *"Project ➔ Add Object ➔ C Code File"*.
3. In the *"Add C Code-File"* dialog, use the input assistant (...) to select a file in `*.c`, `*.cpp`, `*.h`, or `*.hpp` format, and then click *"Add"*.
   ⇨ CODESYS inserts the selected C code file into the device tree below the *"C Code Module"*.
4. If you double-click the new C code file () in the device tree, then it opens in the editor for modification.

Generating empty C code files

Requirement: A project is open that already includes a C code module.

1. Click the object *"C Code Module"* in the device tree.
2. Click *"Project ➔ Add Object ➔ C Code File"*.
3. In the *"Add C Code File"* dialog, specify the name for the new C code file with the appropriate file extension and click *"Add"*.
   ⇨ CODESYS inserts the selected C code file into the device tree below the *"C Code Module"*.
4. If you double-click the new C code file () in the device tree, then it opens in the editor for modification.

Converting C code files into IEC objects for use as programming objects in applications

Requirement: A project is open that includes a C code module and C code files. For example, the C code file contains the following C code: `int adder(int a, int b);`

1. Click a C code file with the file extension `*.h`. In this example, it is `test.h`.
2. Click *"Build ➔ C-Integration ➔ Create IEC Interface"*.
   ⇨ The dialog *"Create C Interface"* opens and lists the file `test.h` and its function `adder (int, int)`. Both are activated for the import.
3. Click *"Import"*.
4. CODESYS generates the *"adder (FUN)"* function and inserts it as an object in the *"IEC Interface"* folder in the device tree.

5. When you double-click the *"adder (FUN)"* object, it opens in the editor.
   ⇨ It contains the following declaration part:

```
{attribute 'C_SOURCE_EXPORT' := 'adder'}
{attribute 'external_name' := 'adder_CExt'}
FUNCTION adder : DINT
VAR_INPUT
  → a : DINT;
  → b : DINT;
END_VAR
VAR
END_VAR
```

6. You can now call the `adder` function in the implementation part of a POU (example: `adder (diVar1, diVar2);`).

Creating C stubs

Requirement: A project is open that includes a C code module. A POU is added to the C code module and this POU has implemented code.

▷ In the device tree, select the POU below the C code module and click *"C-Integration ➔ Create Stub Implementation in C"*.

⇨ CODESYS creates the objects *"iec_external.c"* and *"iec_external.h"* and adds them to the *"Extensions"* folder in the device tree.

In the message view (*"C Code module"* category), you will find a message that an m4 file has been successfully created.

> *When you click "Create Stub Implementation in C", the application is compiled automatically. If errors occur in the process, then these are indicated in the message view. In addition, please monitor the messages in the "C Code Module" category.*

Assigning pre-compiled run-time modules to devices and saving them in libraries

Requirement: A library (*.library) is open in CODESYS.

1. Click *"View ➔ POUs"*.
   ⇨ The *"POUs"* view opens and displays the library project and its objects.
2. Select the library project and click *"Project ➔ Add Object ➔ C-Implemented Library"*.
3. Click *"Add"* in the *"Add C-Implemented Library"* dialog.
   ⇨ CODESYS adds the object *"C Implemented Library"* to the *"POUs"* view.
4. Double-click the *"C Implemented Library"* object.
   ⇨ The object opens in its editor
5. Click *"Add"* in this editor.
   ⇨ The *"Select Device"* dialog opens.

6. In the *“Object file”* input field, specify the name of a dynamically loadable module in the format *.dll or *.so.

> **NOTICE!**
> The *.dll file must contain the title of the library project in its name. For example, if the library project is named XYlib, then the *“Object file”* must be called: `<Name>_XYlib.dll`.

7. In the *“Device”* window, select a device for assignment of the *“Object file”*.
8. Click *“Select Device”*.
   ⇨ CODESYS displays the created device file assignment in the editor on the tab *“Compiled Components”*.
9. Save the library project.

1.4.1.8.11 Programmatic Access to I/Os

CODESYS provides the following features for mapping project variables to input, output and memory addresses:

- Assignment of project variables to input, output and memory addresses in the *“I/O Mapping”* tab of the device editor
- Programmed access to I/Os
  – Variables configuration
  – AT declaration

> **NOTICE!**
> We recommend that you define the mapping of project variables to input, output and memory addresses in the *“I/O Mapping”* of the editor of the respective device.

See also

- *Chapter 1.4.1.7.1 “Configuring Devices and I/O Mapping” on page 213*

Variables configuration - VAR_CONFIG

Use the variables configuration for mapping variables of functions blocks to the process map. For declarations in the function block, assign the variables to the device inputs/outputs without providing the full address. Later, the exact address is provided centrally for all function block instances of the application in a global variable list including `VAR_CONFIG` declarations. This global variables list with the `VAR_CONFIG` declarations is termed the "variables configuration".

> **NOTICE!**
> For changes to variables that are assigned to I/O addresses, CODESYS displays them immediately In the process map. For changes to variables that are mapped by a variables configuration, CODESYS displays them not until the end of the responsible task.

Declaration of variables in functions blocks

When declaring variables in a function block, declare the variables between the keywords `VAR` and `END_VAR` and assign incomplete addresses to the variables. Mark these incomplete addresses with an asterisk (`*`).

Syntax:

```
<identifier> AT %<I|Q>*:<data type>;
```

Example

Define two local I/O variables: the input variable `xLocIn` and the output variable `xLocOut`.

```
FUNCTION_BLOCK locio
VAR
  xLocIn AT %I*: BOOL := TRUE;
  xLocOut AT %Q*: BOOL;
END_VAR
```

Final definition of addresses in the variables configuration of the global variables list

In the global variables list that you use as the variables configuration, define the variable declarations with the absolute addresses between the keywords `VAR_CONFIG` and `END_VAR`.

You must declare the `VAR_CONFIG` variables with the complete instance path, separating the individual POU and instance name by a dot (.). The declaration must include an address whose class (input/output) agrees with the class of the incomplete address (%I*, %Q*) in the function block. The data type must also agree.

Syntax:

```
<instance variable path> AT %<I|Q><location>: <data type>;
```

If the path instance does not exist, then an error is reported. CODESYS prints an error also if there is not an address configuration available for a variable that you declared with an incomplete address.

Example

The `locio` function block in the example above is used in a program as follows:

```
PROGRAM PLC_PRG
VAR
  locioVar1: locio;
  locioVar2: locio;
END_VAR
```

A correct variables configuration in a global variable list could then look like this:

```
VAR_CONFIG
  PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;
  PLC_PRG.locioVar1.xLocOut AT %QX0.0 : BOOL;
  PLC_PRG.locioVar2.xLocIn AT %IX1.0 : BOOL;
  PLC_PRG.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

See also

- *Chapter 1.4.1.19.2.10 “Configuration variables - VAR_CONFIG” on page 534*
- *Chapter 1.4.1.8.11.2 “AT declaration” on page 281*
- *Chapter 1.4.1.19.4.10 “Addresses” on page 643*

Creating a variables configuration

Requirement: You have a project open that includes a controller configuration with a field device. The project contains a program (e.g. `PLC_PRG`) and a function block (e.g. `func1`). The field device has inputs and outputs. The textual view is selected in the options for the declaration editor.

In the function block, assign variables to device I/Os with incomplete addresses and then create a variables configuration.

1. Double-click a function block in the device tree (e.g. `func1`).
   ⇨ The function block editor opens.
2. Type the following between the keywords `VAR` and `END_VAR`: `xLocIn AT %I*: BOOL := TRUE;` and `XLocOut AT %Q*:BOOL;` in the next line.
   ⇨ You have declared an input variable `xLocIn` and assigned it to the incomplete input address `%I*` of a field device. You have assigned the declared output variables have to the incomplete output address `%Q*`.

3. Click the `PLC_PRG` object in the device tree and add the following to the declaration section of the program between `VAR` and `END_VAR`:

   ```
   locioVar1: func;
   locioVar2: func;
   ```

4. Right-click the *"Application"* object in the device tree and click *"Add Object ➔ Global Variable List"* and then click *"Add"* in the *"Add Global Variable List"* dialog box.

   ⇨ The global variables list is added to the device tree and opens in the editor.

5. Change the keyword `VAR_GLOBAL` to `VAR_CONFIG`.

6. Click *"Declarations ➔ Add All Instance Paths"*.

   ⇨ The following instance paths are added:

   ```
   PLC_PRG.logioVar1.xLocIn AT %I*;
   PLC_PRG.logioVar2.xLocIn AT %I*;
   PLC_PRG.logioVar1.xLocOut AT %Q*;
   PLC_PRG.logioVar2.xLocOut AT %Q*;
   ```

7. Now, replace the incomplete addresses `%I*` and `%Q*` with the absolute, complete addresses.

See also

- *Chapter 1.4.1.20.3.17.4 "Command 'Add all instance paths'" on page 1124*

AT declaration

In the variables declaration, the code `AT` assigns a project variable to a specific input address, output address, or memory address of the PLC that is configured in the device tree. You can also define the assignment of variables to an address in the *"I/O Mapping"* dialog of the device in the PLC configuration.

Syntax

```
<variable name> AT <address> : <data type>;

<address> : %<memory area prefix> ( <size prefix> )? <memory position>
```

The AT declaration allows you to give the address a meaningful name. You can make any necessary changes for the input or output signals at just one location, for example in the declaration.

Examples

`VAR wInput AT %IW0 : WORD; END_VAR`	Variable declaration with address information of an input word
`VAR xActuator AT %QW0 : BOOL; END_VAR`	Boolean variable declaration Note: For Boolean variables, one byte is allocated internally if a single bit address is not specified. A change in the value of `xActuator` affects the range from `QX0.0` to `QX0.7`.
`VAR xSensor AT IX7.5 : BOOL; END_VAR`	Boolean variable declaration with explicit specification of a single bit address. On access, only the input bit 7.5 is read.
`VAR xSensor AT IX* : BOOL; END_VAR`	For the address specification, the placeholder `*` is given instead of the memory position. The final address specification is done in the variables configuration. Note: This is possible in function blocks.

If you assign a variable to an address, please note the following:

- You cannot write to variables that are placed at inputs. This will cause a compiler error.
- You can perform AT declarations only for local and global variables, not for input/output variables of POUs.
- Furthermore, AT declarations cannot be used in persistent variable lists.
- If you use AT declarations for structure components or function block variables, then all instances use the same memory. This is just like using static variables in classic programming languages, such as C.
- The memory layout of structures also depends on the target system.

NOTICE!
If you do not specify a single bit address explicitly, then Boolean variables are allocated byte-by-byte.

Example

```
PROGRAM PLC_PRG
VAR
    xVar AT %QW0 : BOOL;
END_VAR

xVar := TRUE;
```

When the variable `xVar` is written, the output memory range from `QX0.0` to `QX0.7` is affected.

See also

- *Chapter 1.4.1.8.11.1 “Variables configuration - VAR_CONFIG” on page 279*
- *Chapter 1.4.1.19.2.10 “Configuration variables - VAR_CONFIG” on page 534*
- *Chapter 1.4.1.19.4.10 “Addresses” on page 643*

1.4.1.8.12 Checking Syntax and Analyzing Code

CODESYS provides useful functions for detecting errors and assisting you while you create programs. The syntax check flags errors and prints them to the message view as early as the programming phase.

The static code analysis in CODESYS also assists you in complying with defined coding guidelines and detecting weak constructs.

See also

- *Chapter 1.4.1.8.12.1 “Checking Syntax” on page 283*
- *Chapter 1.4.1.8.12.2 “Analyzing code statically” on page 283*

Checking Syntax

When you input code, the precompile in CODESYS already runs some basic checks. Then, wavy underlines appear under buggy code in the editor and an error message is printed to the messages view.

CODESYS automatically generates the application code from the source code that was written in the development system. This is done automatically before downloading the application to the PLC. Before the application code is generated, a test is performed for checking the allocations, the data types, and the availability of libraries. Moreover, the memory addresses are allocated when the application code is generated. You can click *“Build ➔ Generate Code”* to execute this command explicitly, or press the *[F11]* key. This is useful for detecting any errors in your source code, even when the PLC is not connected yet.

CODESYS prints all errors and warnings to the "Build" category of the messages view. Double-clicking the error message opens the respective POU in the editor with the buggy code marked. As an alternative, you can also jump to the buggy code by right-clicking the error message.

Note the settings for this in the CODESYS options.

See also

- *Chapter 1.4.1.20.4.13.23 “Dialog 'Options' - 'SmartCoding'” on page 1201*

Analyzing code statically

You can subject your source code also to static analysis (lint) during the code generation. This determines whether or not your source code complies with the coding guidelines that you defined - according to the idea behind the lint analysis tool.

- You activate the rules to the checked in the *“Project Settings”* dialog in the *“Static Analysis Light”* category. The check itself is performed automatically each time code is generated, for example when you click *“Build ➔ Generate Code”*. If divergence from the rules is determined, then it is reported as an error message in the *“Build”* category of the message view. The reported errors have the prefix `SA<number>`.

NOTICE!

For static code analysis with *“Static Analysis Light”*, only the application code of the project is checked. Libraries are excluded from the check.

GVL variables in the *“POUs”* view are not necessarily checked: If you have a project with several applications, then only the objects in the active application are checked. If you have only one application, then the objects in the common POU pool are also checked.

“Static Analysis Light” includes only a reduced set of rules in the default development system. A larger set of rules, additional naming conventions, and metrics are available when you install the CODESYS Static Analysis add-on.

Deactivating lines of code in the implementations with pragmas from the static analysis

By means of the pragma `{analysis ...}`, you can mark code so that the specified rules are not checked. As a result, the marked lines of code are not subjected to static analysis. The marked code is ignored during the check.

Syntax:

`{analysis <sign><rule number>|,<other combinations of signs and rules, comma-separated>}`

`-<rule number>`: Deactivate the rule SA<rule number>.

`-<rule number>`: Activate the rule SA<rule number>.

Excluding implementation code

☑ Requirement: Rules are activated in the *"Project Settings"* dialog.

1. Add the pragma `{analysis -<number>}` above the line of code that contains code not to be checked first of all. For example, for the rule SA0024
   ⇨ The line of code is the first line of the code snippet that is not checked with rule 24.
2. Add the pragma `{analysis -<number>}` below the line of code that contains code not to be checked first of all. For example, for the rule SA0024
   ⇨ The line of code above is the last line of the code snippet that is not checked with rule 24.

Example: Ignore untyped literal

```
{analysis -24}
nTest := 99;
iVar := INT#2;
{analysis +24}
```

The rule *"SA0024: Untyped literals only"* is deactivated for two lines. An error is not issued although the code does not correct to: `nTest := DINT#99;`

Example: Ignore several rules

```
{analysis -10, -24, -18}
...
{analysis +10, +24, +18}
```

"SA0010: Arrays with only one component"

"SA0018: Unusual bit access"

"SA0024: Untyped literals only"

However, you cannot deactivate the rule `SA0004`: "Multiple Write Access on Output" with a pragma.

Excluding programming objects with pragmas from the static analysis

Syntax:

`{attribute 'analysis' := '-<rule number>[,<other negative rule numbers, comma-separated>]'}`

When you insert the attribute pragma in the declaration part of a programming object, the specified rules are excluded for the entire programming object. If multiple rules are excluded, then the rules are each comma-separated with a dash and a number. A pragma statement for activation is not required.

Example

```
{attribute 'analysis' := '-33, -31'}
TYPE LocalData :
STRUCT
    iLocal : INT;
    uiLocal : UINT;
    udiLocal : UDINT;
END_STRUCT
END_TYPE
```

The rules SA0033 and SA0031 are ignored for the structure `LocalData`.

```
{attribute 'analysis' := '-100'}
big: ARRAY[1..10000] OF DWORD;
```

The rule SA0100 is ignored for the array `big`.

See also

- *Chapter 1.4.1.20.4.11.8 “Dialog 'Project Settings' - 'Static Analysis Light'” on page 1177*

1.4.1.8.13 Orientation and Navigation

Using the cross-reference list to find occurrences

The occurrences of symbols of a variable, a POU (program, function block, function), or a DUT can be displayed in a cross-reference list. Then you can jump from the list directly to the corresponding locations in the project.

There are two ways to search for occurrence locations of a symbol:

- Plain text search: You manually specify a text (symbol name, placeholder) in the *“Cross-Reference list”* view.
- Search for a specific declaration:
  - In the *“Cross-Reference List”* view, you select the declaration from the input assistant.
  - The focus is on a symbol name in the POU editor and you start the cross-reference search from the context menu.
  - The focus is on a symbol name in the POU editor, the *“Cross-Reference List”* view is open, and the cross-reference search executes automatically.
  - In the *“Cross-Reference List”* view which already lists occurrence locations for several declarations, you limit these results to a specific declaration.

Cross references with text search by symbol name

Requirement: The *“Cross-Reference List”* view is open.

1. Specify a string in the field next to the name, for example the identifier of the variable for which you want to find the occurrence location in the project. Example: "`iCounter`".

   For the text search, you can use the asterisk "`*`" (for any number of characters) or the question mark "`?`" (for an exact number of characters) combined with a substring of a variable identifier.

   Use the percent sign "`%`" to search for IEC addresses. Examples: "`%MW8`", "`%M*`".

2. Click the button to start a text search in the project.

   ⇨ The view *"Cross-Reference List"* opens and displays the occurrence locations for the `iCounter` variable. The declaration parts are always displayed in the project with the occurrence location indented.

3. Double-click an occurrence location in the cross-reference list.

   ⇨ The respective object opens in the editor with the marked occurrence location.

Cross-references for a specific symbol declaration

Requirement: A POU is open in the editor.

From the POU editor, with a menu command

1. Set the cursor at the identifier of the symbol (variable, POU) in the declaration part or implementation part.
2. Click *"Browse for Symbol ➔ Browse Cross-References"* in the context menu or *"Edit"* menu.

   ⇨ The *"Cross-Reference List"* view opens and shows the occurrence locations of the variables or POU.

If the *"Cross-Reference List"* view is already open, then you can also search the occurrence locations for a specific result as follows:

From the POU editor, automatic

▷ Select the *"Automatically list selection in cross reference view"* check box in *"Tools ➔ Options"* (*"SmartCoding"* category). Select the name of the symbol in the POU, or set the cursor in the name.

⇨ Depending on the position of the selection or cursor, the cross-reference list automatically shows the occurrence locations for the respective symbol.

In the cross-reference list view, with input assistance

▷ In the *"Cross-Reference List"* view, use the input assistant to specify a symbol name in the field next to *"By declaration"*.

⇨ The cross-reference list displays the occurrence locations for the symbol.

In the cross-reference list view, limited to a specific declaration

▷ If multiple declarations for a symbol are listed in the *"Cross-Reference List"* view, for example after a text search, then you can reduce the display to one result: Select the line with the desired declaration and click the button or click *"Limit Results to Current Declaration"* in the context menu.

⇨ the cross-reference list includes only the occurrence locations for the selected declaration.

See also

- *Chapter 1.4.1.20.3.3.13 "Command 'Cross Reference List'" on page 990*
- *Chapter 1.4.1.20.3.2.29 "Command 'Browse Cross References'" on page 974*
- *Chapter 1.4.1.20.4.13.23 "Dialog 'Options' - 'SmartCoding'" on page 1201*

Finding declarations

CODESYS provides the capability of searching the entire project for the definition location of a variable or function. The block that includes the definition opens in the editor with the marked declaration.

Finding the declaration of a variable

Requirement: You have opened a POU in the editor.

1. Set the cursor at an identifier in the implementation section.
2. Click *“Edit ➔ Browse ➔ Go to Definition”*.
   ⇨ The POU with the declaration opens in the editor with the variable definition marked. If the definition is located in a compiled library, then the respective block opens in the library manager.

You can execute this command in both online and offline mode.

Examples

The following block includes a function block definition (`fbinst`), a program call (`prog_y()`), and a function block call (`fbinst.out`):

```
VAR fbinst:fb1; ivar:INT; END_VAR prog_y(); ivar:=prog_y.y;
res1:=fbinst.out;
```

If the cursor is located at `prog_y`, then the command opens the program `prog_y` in the editor.

If the cursor is located at `fbinst`, then this command focuses in the declaration section at line `fbinst:fb1;`

If you set the cursor at `out`, then this command opens the function block `fb1` in its editor.

See also

- *Chapter 1.4.1.20.3.2.37 “Command 'Go to Definition'” on page 979*

Setting and using bookmarks

Bookmarks are used for easy navigation through long programs. You can use bookmarks in all implementation language editors, except SFC (sequential function chart). Commands help to navigate directly to the marked position in the program.

Setting and deleting bookmarks

Requirement: The POU is open in the editor.

1. Set the cursor at any program line.
2. Click *“Edit ➔ Bookmarks ➔ Toggle Bookmark”*.
   ⇨ A bookmark is set at this position in the program. This is marked by the bookmark symbol .
3. Set several bookmarks at different places in the program.
4. Set the cursor at a bookmarked program line.

5. Click *“Edit ➔ Bookmarks ➔ Toggle Bookmark”*.

   ⇨ The bookmark is removed. The bookmark symbol is deleted.

   As an alternative to this, you can delete one or more bookmarks in the *“Bookmarks”* view by clicking the button. For this purpose, the corresponding bookmarks have to be selected in the *“Bookmarks”* view.

> *Click “Edit ➔ Bookmarks ➔ Clear All Bookmarks (Active Editor)” to remove all bookmarks from the active POU.*

> *In order to delete all bookmarks in a project, click “Clear All Bookmarks”. However, for this command to be available, you first have to add it to a menu by means of the command “Tools ➔ Customize”.*

See also

- *Chapter 1.4.1.20.3.2.22 “Command 'Toggle Bookmark'” on page 972*
- *Chapter 1.4.1.20.3.2.27 “Command 'Clear All Bookmarks (Active Editor)'” on page 974*
- *Chapter 1.4.1.20.3.2.28 “Command 'Clear All Bookmarks'” on page 974*

Jumping to bookmarks within a POU

Requirement: The POU is open in the editor. Multiple bookmarks are set.

1. Click *“Edit ➔ Bookmarks ➔ Next Bookmark (Active Editor)”*.

   ⇨ Depending on the current cursor position, the cursor jumps to the next bookmark (see below).

2. Click *“Edit ➔ Bookmarks ➔ Previous Bookmark (Active Editor)”*.

   ⇨ Depending on the current cursor position, the cursor jumps to the previous bookmark (see above).

See also

- *Chapter 1.4.1.20.3.2.23 “Command 'Next Bookmark (Active Editor)'” on page 973*
- *Chapter 1.4.1.20.3.2.25 “Command 'Previous Bookmark (Active Editor)'” on page 973*

Jumping to bookmarks of different POUs in a project

A project is open with multiple POUs. Multiple bookmarks are set in different POUs.

1. Click *“View ➔ Bookmarks”*.

   ⇨ The *“Bookmarks”* view opens.

   All bookmarks in the project are listed in a table in the view.

2. Click the *“Next Bookmark”* button.

   ⇨ In the *“Bookmarks”* view, the bookmark in the row below the selected bookmark is selected.

   The POU with the recently selected bookmark in the table opens in the editor and the row with the bookmark is selected in the POU.

3. As in step 2, you can click the *"Previous Bookmark"* button to jump to the bookmark in the project that is displayed in the row above it in the *"Bookmarks"* view.

See also

- *Chapter 1.4.1.20.3.3.11 "Command 'Bookmarks'" on page 988*
- *Chapter 1.4.1.20.3.2.26 "Command 'Previous Bookmark'" on page 973*
- *Chapter 1.4.1.20.3.2.24 "Command 'Next Bookmark'" on page 973*

1.4.1.8.14 Searching and replacing in the entire project

In CODESYS you can search for strings in single objects or project-wide. If required, you can replace the string found.

1. Choose the command *"Search"* in the main menu *"Edit ➔ Search Replace"*.
   ⇨ The dialog *"Find"* opens.
2. Enter the string to be found in the field *"Find what"*.
3. Activate the search options
4. Define the objects to be searched by choosing an entry from the combobox *"Search"*.
5. Click on the button *"Find Next"*.
   ⇨ The first hit is displayed.
6. Click on the button *"Replace"* to replace the string found by a different one.
7. Click on the button *"Find All"* to get a list of all hits.

See also

- *Chapter 1.4.1.20.3.2.2 "Command 'Find', 'Find in Project'" on page 966*
- *Chapter 1.4.1.20.3.2.3 "Command 'Replace', 'Replace in Project'" on page 967*

1.4.1.8.15 Refactoring

In general, refactoring is a technique for improving the design of existing software code without changing the way it functions.

In CODESYS, refactoring provides functions for renaming objects and variables and updating referenced pins. You can display all occurrences of renamed objects and variables and then rename them all at once or individually. In *"Tools ➔ Options"*, you can also configure where CODESYS will prompt you for refactoring.

Renaming global variables

Requirement: A project is open that includes at least a function block *"FB"* and a global variable list. The global variable list *"GVL"* is open in the editor and contains a variable declaration (example: `iGlobal`). *"FB"* uses `iGlobal`.

Renaming global variables throughout the project

1. Select the global variable name `iGlobal`.
2. Right-click the variable and click *"Refactoring ➔ Rename iGlobal"*.
3. In the *"Rename"* dialog, type a name in the *"New name"* input field, for example `iGlobalOK`, and click *"OK"*.
   ⇨ The *"Refactoring"* dialog opens. In the device tree view on the left, the *"GVL"* and *"FB"* objects are highlighted in red and yellow. In the view on the right, *"FB"* in is open in its editor and `iGlobal` has already been renamed as `iGlobalOK`.
4. Click *"OK"*.
   ⇨ No global variable `iGlobal` is in your project. Now `iGlobalOK` is everywhere.

Renaming global variables throughout the project (except for a POU)

1. Select the global variable name `iGlobal`.
2. Right-click the variable and click *"Refactoring ➔ Rename iGlobal"*.
3. In the *"Rename"* dialog, type a name in the *"New name"* input field, for example `iGlobalTest`, and click *"OK"*.

   ⇨ The *"Refactoring"* dialog opens. In the device tree view on the left, the *"GVL"* and *"FB"* objects are highlighted in red and yellow. In the window on the right, the function block *"FB "* is open in its editor. `iGlobalTest` is listed instead of `iGlobal`.
4. Right-click in the view on the right.
5. Click *"Reject this Object"* and click *"OK"*.

   ⇨ The global variable `iGlobal` is available in *"FB"* in your project. The variable `iGlobalTest` is now specified in the objects where the previous variable occurred.

   The error message in the message view reports that the `iGlobal` identifier is not defined.

Adding and removing input variables

In the declaration part of blocks, you can add and delete input and output variables by using the refactoring commands. CODESYS performs updates at the occurrence locations and calling locations of the blocks. You can accept or reject these updates individually. The *"Refactoring"* dialog also opens for this purpose.

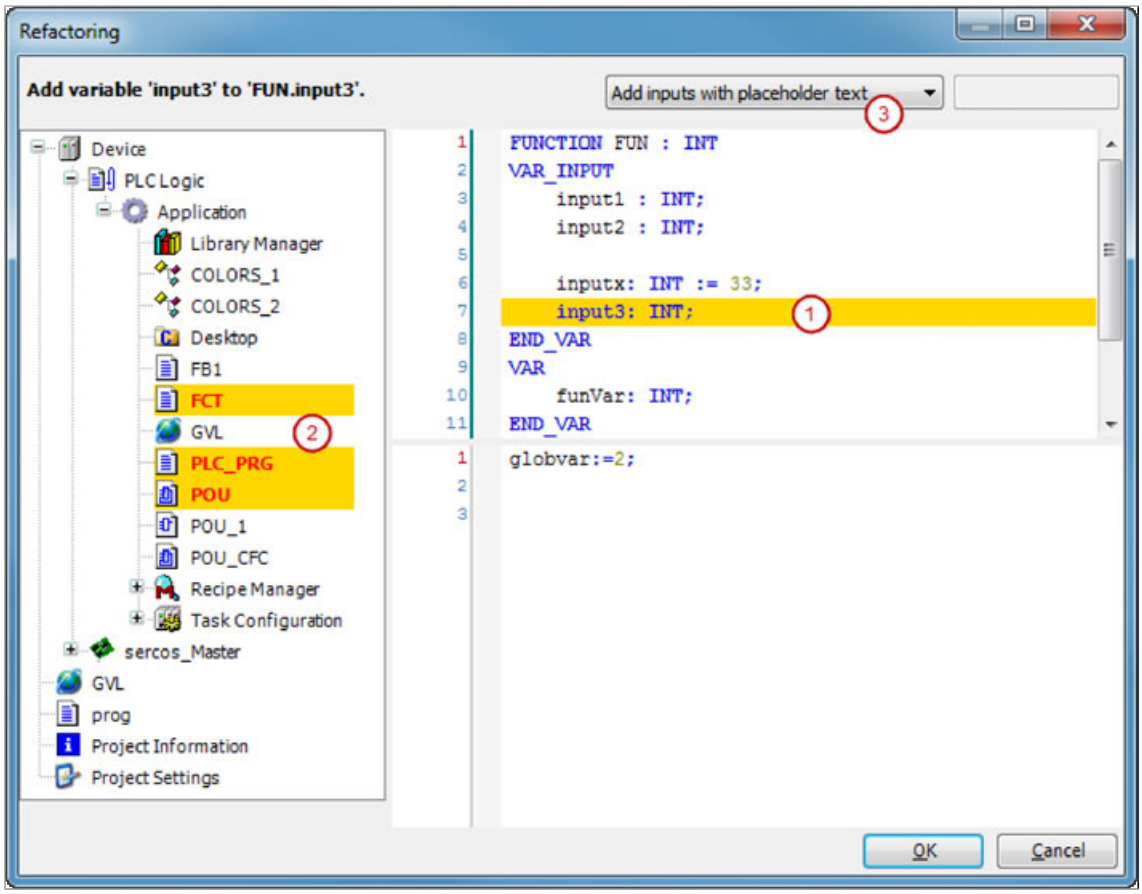
Requirement: The `FCT` (function type) POU is open in the editor. The function already contains the input variables `input1`, `input2`, and `inputx`. They are called in the `PLC_PRG` and `POU` programs.

1. Set the focus in the declaration part of the FCT function.
2. Click *"Refactoring ➔ Add Variable"*.

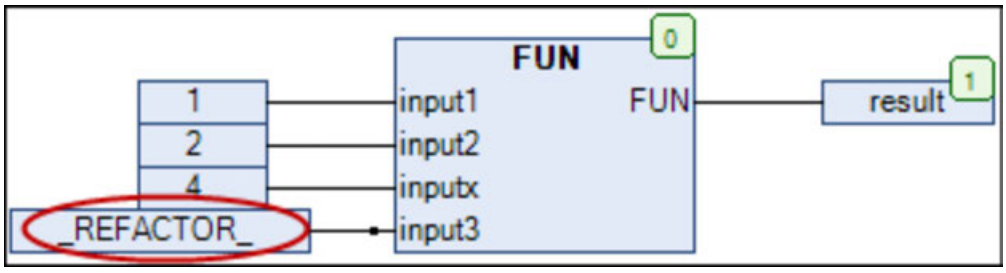
   ⇨ The default dialog opens for declaring variables.
3. Declare the variable `input_3` with the scope of `VAR_INPUT` and data type `INT`. Click *"OK"* to close the dialog.

   ⇨ The *"Refactoring"* dialog opens (see figure below). The affected locations are marked in yellow. (1)+(2)
4. In the upper right corner, select *"Add inputs with placeholder text"* from the drop-down list. (3).
5. In the left side of the window, click one of the highlighted objects (for example, `PLC_PRG`). Right-click and choose the *"Accept Whole Project"* command to add the new variable at the new location of use in `FCT` for the entire project.

   ⇨ You see the change in the implementation part of `PLC_PRG` in the view on the right: The placeholder `_REFACTOR_` appears at the location where the new variable was added.
6. Click *"OK"* to close the *"Refactoring"* dialog.
7. Click *"Edit ➔ Find"*. Search the project for "_REFACTOR_" to check and edit the affected locations.
8. Note: As an alternative, you can insert the new variable with another initialization value without working with a placeholder first. In this case, in Step 4 you select "Add inputs with the following value" and type the value in the field on the right side of the drop-down list.



Example of a new variable with placeholder text in a CFC block:



Please note that you can also remove variables with refactoring.

Reordering variables in the declaration

In the declaration part of function blocks, you can change the order of declarations by refactoring This is possible for declarations with scope `VAR_INPUT`, `VAR_OUTPUT`, or `VAR_IN_OUT`.

Requirement: The declaration part of a POU is open and includes declarations, for example:

```
VAR_INPUT
    invar2 : INT;
    invar1 : INT;
    in : DUT;
    bvar : BOOL;
    invar3 : INT;
END_VAR
```

1. Right-click in this declaration block to access the context menu.
2. Click *“Refactoring ➔ Reorder Variables”*.
   ⇨ The *“Reorder”* dialog opens with a list of `VAR_INPUT` variables.

3. Drag the *“invar1 : INT;”* entry to the position before the *“invar2.”* entry.
   ⇨ The `invar1` declaration is now at the top position.
4. Click *“OK”* to close the dialog.
   ⇨ The *“Refactoring”* dialog opens. The affected locations are marked in yellow (see figure above).
5. Click *“OK”* to accept the new order for the function block.

Changing a variable declaration and applying refactoring automatically

Refactoring helps you in the declaration when renaming variables (by means of "Auto declare").

☑ Requirement: Function block `fb_A`.

1. Click *“Tools ➔ Options”*.
   ⇨ The *“Options”* dialog opens.
2. Select the *“Refactoring”* category.
3. In *“Auto-Declare”*, activate the options *“On renaming variables”* and *“On adding or removing variables, or for changing the namespace”*.
4. Double-click the function block `fb_A`.
5. Select a variable in the declaration of `fb_A`, for example `iA`. As an alternative, you can set the cursor before or in the variable.
6. Specify *“Edit ➔ Declare variable”* (*[Shift]+[F2]*).
   ⇨ The *“Declare Variable”* dialog opens. The dialog includes the settings of `iA`.
7. Change the name of `iA` to `iCounter_A`.
8. The option *“Changes by means of refactoring”* appears and is activated.
9. Click *“OK”*.
   ⇨ The dialog *“Refactoring” “Renaming from iA to iCounterA”* opens. All locations affected by the variable renaming are marked there.
10. Click *“OK”* to close the dialog.
    ⇨ The changes are applied.

See also

- *Chapter 1.4.1.20.3.2.40 “Command 'Refactoring' - 'Rename <...>'” on page 980*
- *Chapter 1.4.1.20.3.2.41 “Command 'Refactoring' - 'Update Referenced Pins'” on page 981*
- *Chapter 1.4.1.20.3.2.42 “Command 'Refactoring' - 'Add Variable'” on page 981*
- *Chapter 1.4.1.20.3.2.43 “Command 'Refactoring' - 'Remove <variable>'” on page 983*

1.4.1.8.16 Task Configuration

In the task configuration, you define one or more tasks for controlling and executing the application program in the controller. Each application must include a *“Task Configuration”* object.

A task is a time-based flow unit of an IEC program. You define a task with a name, a priority, and a type, which determines which condition triggers the start of the task. You can define this condition either by time (cyclic-interval, freewheeling) or by the occurrence of an internal or external event to process the task. Examples of an event are the rising edge of a global project variable or an interrupt event of the controller.

A task calls one or more program blocks (POUs). These programs can be application-specific (objects below the application in the device tree) or project-specific (objects available in the POU window). In the case of a project-specific program, the application instances the project-global program. If CODESYS processes the task in the current cycle, then the programs are executed for the duration of a cycle.

With the combination of priority and condition, you define the order in which the tasks are processed. You can configure a watchdog for each task, and you can link a start, stop, and reset directly to the execution of the project block.

Rules for the processing order of the defined tasks:

- If the task condition is satisfied, then CODESYS processes the task.
- If several tasks satisfy the condition for processing at the same time, then CODESYS processes the tasks with the highest priority first.
- If several tasks with the same priority level satisfy the condition for processing at the same time, then CODESYS processes the longest waiting task first.
- The program calls are processed in the order they appear in the configuration dialog of the task.
- If a called program has the same name in the device tree of the application and in a library or project-global in the POU window, then the application program is used.

Attention

All tasks share one process map. The reason is as follows: When each task has its own individual process map, performance is compromised. However, the process map can be consistent only with one task. When you create a project, you must ensure that the application copies the input and output data to a safe location in case of conflicts. Modules, such as the library `SysSem`, provide the capability of solving consistency and synchronization problems.

Consistency problems can also occur when accessing other global objects, such as global variables or blocks. Consistency problems always occur if several tasks read and write to one variable. Modules, such as the library `SysSem`, are available as a solution.

Creating a task configuration

Requirement: The open project includes a program-type POU and a *“Task Configuration”* with a *“Task”* object has been inserted below *“Application”* in the device tree.

1. Double-click the task object below *“Task Configuration”* in the device tree.

   ⇨ The *“Configuration”* tab of the task object opens.

2. In the *“Type”* dropdown list., click *“Cyclic”*.

   ⇨ The *“Interval (e.g. t#200ms)”* input field appears.

3. Enter `t#300ms` in the *“Interval (e.g. t#200ms)”* input field.

4. Click *“Add Call”*.

   ⇨ The Input Assistant opens.

5. In *“Input Assistant ➔ Categories” -> “Programs”*, click the desired POU and then click *“OK”*.

   ⇨ CODESYS inserts the selected POU into the POU list of the *“Configuration”* tab and below the task object in the device tree.

   When the application is executed from the controller, CODESYS executes the selected POU in cyclical intervals of `300 ms`.

- ↳ *Chapter 1.4.1.20.2.27.1 “Tab 'Configuration'” on page 942*

Definitions of Jitter and Latency

In the *"Task Configuration"* object, on the *"Watchdog"* tab, you can monitor the periodic jitter values of the individual tasks at runtime. The periodic jitter is differentiated from latency-based release jitter. See the following definitions:

Periodic jitter

Periodic jitter (J_{per}) is the deviation of the cycle time of a task (T_{per}) from the desired task cycle time (T_0).

$J_{per} = T_{per} - T_0$

The desired (ideal) cycle time T_0 is specified in the configuration of the task as *"Interval"*.

You can monitor the current value, as well as the maximum and minimum value of the periodic jitter, on the *"Watchdog"* tab of the *"Task Configuration"*.

If the sum of all negative J_{per} values and the sum of all positive J_{per} values do not balance each other, then a drift results.

Latency

Latency is the delay between the invocation of a task and the actual start of its release.

Release jitter

The release jitter J_r is the difference between the maximum and the minimum latency (L) that has ever occurred.

$J_r = L_{max} - L_{min}$

In the case that $L_{max} = L_{min}$, a release jitter J_r of 0. results. This corresponds to a plain offset shift.

See also

- *Chapter 1.4.1.8.16 "Task Configuration" on page 292*
- *Chapter 1.4.1.20.2.27.1 "Tab 'Configuration'" on page 942*
- *Chapter 1.4.1.20.2.26.3 "Tab 'Monitor'" on page 940*

1.4.1.8.17 Encrypting an application

You achieve the know-how protection and copy protection of a boot application with the help of PLC -specific license management and its settings in the object properties of the application. In this case, the download code and boot application are encrypted.

Encryption with a dongle

Requirements: You have a project with an application that you want to download to the controller as an encrypted boot application. A security key for license management is connected to your computer.

1. Select the application in the device tree.
2. Select the *"Properties"* command in the context menu.
   ⇨ The *"Properties - <application name>"* dialog opens.
3. Click the *"Encryption"* tab.
4. For *"Encryption Technology"*, select the *"Simple Encryption"* option and type the *"Product Code"* that you received from the hardware manufacturer for the controller. Depending on the controller, it is protected either by a security key (firmcode is shown automatically) or by an integrated Wibu SD card for example.

5. Click *“Online ➔ Login”* and download the application.
   ⇨ If the matching security key and/or valid license is available, then you can download the application to the controller. By default, a boot application is automatically created at this time in the controller directory. The default setting is defined in the application *“Properties”*, in the *“Boot Application”* category.
6. Logout, change the application, and login again.
   ⇨ You are prompted to perform an online change. The dialog provides the option of updating the boot application on the PLC. If the security key and license match, then you can log in. If not, then you receive a corresponding message.

Encrypting with certificates

Requirements: You have a project with an application that you want to download to the controller as an encrypted boot application. In the Windows Certificate Store of your computer, you have a certificate of this controller for encrypting the application. Note: In case you want to download the application to different controllers, you will need the appropriate certificate for each controller.

1. Select the application in the device tree.
2. Select the *“Properties”* command in the context menu.
   ⇨ The *“Properties - <application name>”* dialog opens.
3. Click the *“Encryption”* tab.
4. On *“Encryption Technology”*, select the *“Encryption with certificates”* option.
   ⇨ The *“Certificates”* group is enabled.
5. If there are not any certificates listed in the table, then click the  button.
   ⇨ The *“Certificate Selection”* dialog opens for selecting a certificate from the local Windows Certificate Store.
6. In the lower area, select a certificate and add it to the upper area by clicking the  button, Click *“OK”* to confirm.
   ⇨ The certificate is shown in the *“Certificates”* group of the *“Encryption”* dialog.
7. Select the certificate and click *“Apply”* or *“OK”*.
   ⇨ The certificate is now used to encrypt the application. It can only be transferred to the controller on computers that have an corresponding key installed in the Windows Certificate Store.

See also

- *Chapter 1.4.1.18.3 “Security for the Runtime/PLC” on page 455*
- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*
- *Chapter 1.4.1.5.7 “Encrypting Projects with Certificates” on page 207*
- *Chapter 1.4.1.20.4.10.3 “Dialog 'Properties' - 'Encryption'” on page 1158*

Signing a boot application

1. Click  in the status bar of CODESYS to open the *“Security Screen”* view. Then select a certificate with a private key for a user profile for the *“Digital signature”*. The procedure is described in the instructions "Configuring a certificate for the digital signature in a user profile".
2. Double-click the certificate for the *“Digital signature”* in the *“User”* tab.
   ⇨ The *“Certificate”* dialog opens.
3. On the *“Details”* tab, click *“Copy to file”*.
   ⇨ The *“Certificate Export Wizard”* starts.

4. In the *“Export Private Key”* prompt, select the *“No, do not export the private key”* option.
5. For *“Export File Format”*, select the *“DER encoded binary X.509 (.CER)”* option.
6. In the next step, select a file name and the location for the certificate.
7. After the last step *“Finish”*, a message appears that the export was successful.
8. After successful export to CODESYS, open the device editor by double-clicking the controller in the device tree and selecting the *“Files”* tab for the file transfer.
9. Select the *“Path”* `cert/import` in the right side of the *“Runtime”* dialog.
10. On the left side of the dialog for *“Host”*, select the path in the file system where you saved the exported certificate and selected the certificate.
11. Click >>.
    ⇨ The certificate is copied to the `cert/import` folder.
12. Click the *“PLC Shell”* tab.
13. Type the command `cert-import trusted <file name of the certificate.cer>` in the input line of the tab and press the *[Enter]* key. Note that the file name is specified with the extension `.cer`; otherwise the certificate is not imported successfully.
    ⇨ The certificate is created on the controller under `trusted`. With this certificate, the controller can test the integrity of the boot application.
14. Open the *“Security Screen”* by double-clicking in the status bar.
15. If you want that downloads, online changes, and boot applications of your project are always encrypted, then select the *“Enforce signing of downloads, online changes and boot applications”* option in the *“Security level”* group on the *“User”* tab. To do this, the *“Enforce encryption of downloads, online changes and boot applications”* option also has to be selected.

See also

- *Chapter 1.4.1.5.7 “Encrypting Projects with Certificates” on page 207*
- *“Encryption, signature” on page 453*
- *Chapter 1.4.1.20.3.3.18 “Command 'Security Screen'” on page 995*
- *Chapter 1.4.1.20.4.10.3 “Dialog 'Properties' - 'Encryption'” on page 1158*

Encrypting the download, online change, and boot application

Requirement: The CODESYS Security Agent add-on product is installed.

The *“Security Screen”* view provides an additional tab: *“Devices”*. This allows for the configuration of certificates for the encrypted communication with controllers. In this case, see the help for CODESYS Security Agent.

Alternatives:

If you the CODESYS Security Agent is not available to you, then you can proceed as follows by means of the PLC shell of the device editor:

In order to use certificates on the controller for the encryption of downloads, online changes, and boot applications, these certificates first have to be generated on the controller and loaded from the controller and installed in the Windows Certificate Store.

☑ Requirement: You are connected to the controller.

1. Open the device editor by double-clicking the controller in the device tree, and select the *“PLC Shell”* tab.
   ⇨ The tab appears with a blank display window. Below that is a command line.
2. Type `?` in the command line and press the *[Enter]* key.
   ⇨ All commands are listed in the display window.

3. Type the following command in the command line: `cert-getapplist`.
   ⇨ All used certificates are listed with information about components and availability with certificates.
4. If no certificate is available for the `CmpApp` component, then type the command `cert-genselfsigned <Number of the Component in the applist>`.
5. Click the *"Log"* tab and then the refresh button ().
   ⇨ The display shows whether or not the certificate was generated successfully.
6. Type in `cert-getcertlist` and press the *[Enter]* key.
   ⇨ Your own certificates are listed that can be used for encryption. The information `Number` and `Key usage(s)` are useful in the next step.

   `Number`: The number is specified as a parameter in the next step.

   `Key usage(s): Data encryption` means that this is a certificate of the controller for a download, online change, and boot application.
7. Export the required certificate by typing in the command `cert-export own 0` and press the *[Enter]* key. `0` is the `Number` of the certificate with `Key usage(s):Data encryption`.
   ⇨ The display shows that the certificate has been exported to a `cert` directory.
8. Click the *"Files"* tab of the device editor.
9. Click the refresh button () in the right part of the dialog in *"Runtime"*.
   ⇨ The list of files and directories is refreshed.
10. Open the *"cert"* folder in the list and then the *"export"* subfolder.
11. In the left part of the dialog in *"Host"*, open the directory where the certificate of the controller will be loaded.
12. In the right part of the dialog, select the certificate that you have exported and click  .
    ⇨ The certificate is copied to the selected directory.
13. In the file explorer, go to the directory where the certificate was copied and double-click the certificate.
    ⇨ The *"Certificate"* dialog opens and shows the information about this certificate.
14. On the *"General"* tab, click *"Install Certificate"*.
    ⇨ The *"Certificate Import Wizard"* starts.
15. In the *"Certificate Storage"* dialog, for *"Certificate Import Wizard"*, select the *"Store all certificates in the following store"* option and then select the *"Controller Certificates"* folder.
    ⇨ The controller certificate is imported into the Windows Certificate Store in the *"Controller Certificates"* folder. Now the certificate is available for the encryption of boot applications, downloads, and online changes.
16. Open the *"Security Screen"* by double-clicking  in the status bar.
17. If you want that downloads, online changes, and boot applications of your project are always encrypted, then select the *"Enforce encryption of downloads, online changes and boot applications"* option in the *"Security level"* group on the *"User"* tab.
18. Open the *"Project"* tab and double-click the application in the *"Encryption of boot application, download and online change"* area.
    ⇨ The properties dialog of the application opens.
19. Click the *"Encryption"* tab, select *"Encryption with certificates"* in the *"Encryption technology"* list box, and click  .

    If the *"Enforce encryption of downloads, online changes and boot applications"* option is selected in the *"Security Screen"*, then *"Encryption with certificates"* is already selected.

20. In the *“Certificate Selection”* dialog, select the respective certificate from the *“Controller Certificates”* folder and click .
21. Click *“OK”* to confirm the dialog.
    ⇨ The certificate is displayed in the properties dialog.
22. Confirm the properties dialog of the application.
    ⇨ The certificate is shown on the *“Project”* tab of the *“Security Screen”* in the *“Encryption of boot application, download and online change”* group.

    The boot application, download, and online change are encrypted.

See also

- Help for the CODESYS Security Agent add-on product
- *Chapter 1.4.1.20.2.8.10 “Tab 'PLC Shell'” on page 852*
- *Chapter 1.4.1.20.3.3.18 “Command 'Security Screen'” on page 995*

Deleting a certificate for the encryption of boot application, download and, online change

Requirement: The CODESYS Security Agent add-on product is installed. A certificate with the information "Encrypted Application" is already installed on your computer.

1. In the *“Security Screen”* view, on the *“Project”* tab, in the bottom view, click the entry for the application.
   ⇨ The *“Properties”* dialog for the application opens with the *“Encryption”* tab.
2. For *“Encryption Technology”*, select *“Encryption with certificates”*. In the *“Certificates”* group, click .
3. In the *“Certificate Selection”* dialog, delete the certificate as described above.
4. Click *“OK”* to close the *“Certificate Selection”* dialog.
   ⇨ The certificate is no longer displayed in the *“Properties”* dialog.

See also

- Help for the CODESYS Security Agent add-on product
- *Chapter 1.4.1.20.3.3.18 “Command 'Security Screen'” on page 995*

1.4.1.8.18 Unit conversion

You define a conversion rule when you want to convert data for another system of units. This data is executed for a specific order of magnitude and unit of measure.

Conversion rules are defined in a *“Unit Conversion”* object. CODESYS automatically implements each conversion rule as a function block `<name>_Impl` and instances it as `<name>`. Each conversion rule includes `Convert` and `Reverse` methods for use as function blocks. Locations where you access a variable, you can link the variable to a conversion rule. The input assistant provides conversion rules in the *“Function Blocks”* and *“Instance Calls”* categories. After execution, the result is a converted value according to the conversion rule.

In a visualization, an IEC variable that is configured in an element property can also be linked to a conversion rules.

Defining unit conversions

1. Double-click a *“Unit conversion”* object in the device tree.
   ⇨ The respective editor opens with a table of the defined conversion rules. You edit a rule in *“Type setting”* and a respective condition in *“Condition setting”*.
2. Double-click the *“Add new entry”* field and type a name.
   ⇨ CODESYS implements the `<name>_Impl` function block and instances it as `<name>`.
3. Double-click the *“Type”* field and click a type from the drop-down list.
   ⇨ Input fields are displayed below the table for editing the conversion rule. The input fields vary according to selected type.
4. Change the conversion rule in the input fields.
   ⇨ The changes are displayed in the *“Setting”* category of the table.
5. Double-click the *“Condition”* field and click a condition type from the drop-down list.
   ⇨ Input fields are displayed below the *“Condition Setting”* category of the table to edit the condition. The input fields vary according to selected type.
6. Edit the condition.
   ⇨ The changes are displayed in the *“Condition Setting”* category of the table.

Defining switchable unit conversions

You can define which conversion rule is applied to a specific language or condition.

1. Double-click a *“Unit Conversion”* object in the device tree.
   ⇨ The respective editor opens with a table of the predefined conversion rules.
2. Click the *“Add new entry”* field and type a name.
   ⇨ Example: `Conv_A_LanguageDependent`
3. Double-click the *“Type”* field and click *“Switchable conversion”*. Double-click the *“Condition”* field and click *“Language”*.
   ⇨ Below the main table, the *“Switchable Conversion”* table is displayed with *“Condition setting”*.
4. In the *“Switchable Conversion”* table, double-click a predefined conversion rule from the drop-down list in the *“Switchable conversion name”* column, for example `Conv_AInInch`.

   In *“Condition Setting”*, type a value in the *“For condition 'Language'”* input field, for example `en`.
   ⇨ CODESYS executes the `Conv_AInInch` conversion rule only if the language set in the visualization manager is *“en ”*.
5. In the *“Switchable Conversion”* table, double-click a predefined conversion rule from the drop-down list in the *“Switchable conversion name”* column, for example `Conv_AInMM`.

   In *“Condition Setting”*, type a value in the *“For condition 'Language'”* input field, for example `de`.
   ⇨ CODESYS executes the `Conv_AInMM` conversion rule only if the language set in the visualization manager is *“de”*.
6. Apply the `Conv_A_LanguageDependent` conversion rule in the application or visualization.
   ⇨ If the set language in the visualization is English, then the application visualization apply the `Conv_AInInch` conversion rule. If the set language in the visualization is German, then the application visualization applies the `Conv_AInMM` conversion rule. The current visualization language is located in the `VisuElems.CurrentLanguage` variable.

Applying conversion rules

Add a conversion rule to objects that access IEC variables.

1. In the device tree, double-click an object that accesses IEC variables in order to link an IEC variable to a conversion rule at that location.
2. Declare a variable for the conversion result of the IEC variable.
   ⇨ ST sample code: `rConvertedA : REAL;`
3. Use the input assistant to apply the conversion rule with the `Convert` method and then assign the result to the variable.
   ⇨ ST sample code to link the IEC variable to the conversion rule: `rConvertedA := ConvRule_A.Convert(rA);`

Applying reverse conversion rules

1. In the device tree, double-click an object that accesses an IEC variable.
2. Declare a variable for the result of the conversion rule.
   ⇨ ST sample code: `rReverseA: REAL;`
3. Apply the reverse conversion rule with the `Reverse` method and then assign the result to the variable.
   ⇨ `rReverseA := ConvRule_A.Reverse(rConvertedA);`

Example

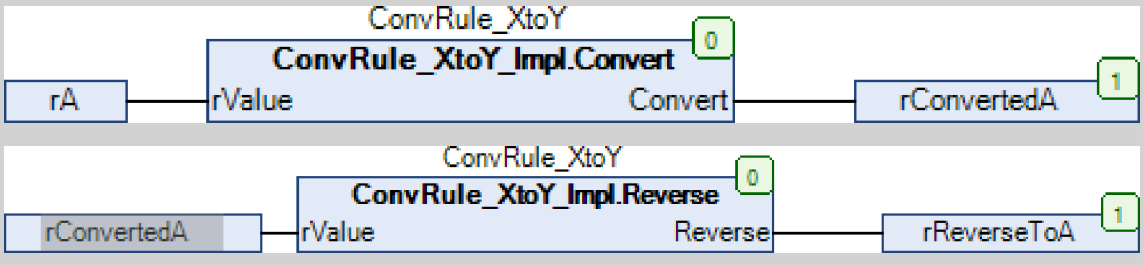
Requirement: The conversion rule is `Conv_XtoY`.

ST call

```
PROGRAM A_PRG
VAR
    rA : REAL;
    rConvertedA : REAL;
    rReverseToA : REAL;
END_VAR

rConvertedA := Conv_XtoY.Convert(rA);
rReverseToA := Conv_XtoY.Reverse(rConvertedA);
```

CFC call

In the CFC editor, define the instance name of the conversion rule via the block. Select the method in the block.



See also

- *Chapter 1.4.1.20.2.33 “Object 'Unit Conversion'” on page 952*
- *Chapter 1.4.1.8.5 “Using input assistance” on page 260*

1.4.1.8.19 Data Persistence

The lifespan of a variable and its data begins at the time when the variable is created and ends at the time when the variable is deleted and its memory is freed. The time when the variable is created, initialized, or instantiated depends on the declared scope. The time when the memory is freed usually depends on the scope as well. For example, the memory of global variables is freed by exiting the application.

They can retain data longer than usual. The following mechanisms are provided for this purpose.

Mechanisms for data retention

- (A): Persistent global variable list with the keyword `PERSISTENT RETAIN`
  Persistent variables retain their values when the application is reloaded. Moreover, the values are restored after a download, warm start, or cold start.
- (B): Retain variables with the keyword `RETAIN`
  Retain variables retain their values after a warm start, but not after reloading the application, a download, or a cold start.
- (C): Variables of the Persistence Manager of the CODESYS Application Composer
  Variables of the Persistence Manager are stored in an external file.
- (D): Recipe variables
  Recipe variables and their values are stored in a recipe file.

See also

- *Chapter 1.4.1.8.19.3 “Retaining data with variables of the persistence manager” on page 307*
- *Chapter 1.4.1.8.19.2 “Preserving data with retain variables” on page 306*
- *Chapter 1.4.1.8.19.4 “Preserving data with recipes” on page 307*
- *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*
- *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537*
- *Chapter 1.4.1.20.2.12 “Object 'Persistent variable list'” on page 872*

Mechanisms in comparison

Which mechanism is suitable for which application? Some common use cases are considered in the table. The specific examples refer to a building control system.

Table 11: Comparison of mechanisms and use cases

	Uses case	**(A) Persistent variables**	**(B) Retain variables**	**(C) Variables of the Persistence Manager**	**(D) Recipe variables**
1	**The application must maintain device settings.** Example: After a power failure, the building control has to have information available about how long a window blind needs to be raised.	**Suitable**[1] Preferred use case In this case, you can also use retain variables instead of persistent variables. This is advantageous for variables whose declaration is often changed.	**Suitable** Preferred use case Retain variables are an advantage when their declarations are changed often.	**Suitable**[2] This is advantageous for controllers that do not have any hardware support. Special functionalities make this possible, such as double file buffering.	Possible, but very complicated and therefore not recommended.
2	**The application must maintain values also after program changes or extensions.**				
	2a: Rare extensions Example: An application programmer extends the program with a new switch and installs a new light. The building control must still have saved values available until then.	**Suitable**[1] Preferred use case	**Suitable**	**Suitable**[2]	Possible, but complicated.
	2b: Unrestricted changes, including deleting or changing the data type of variables The building control is running and is persistent. When an application programmer adds a new functionality to the controller and therefore adds another persistent variable to a function block, the values saved up to that point must be retained. For example, the program in an FB is extended with a variable that controls the automatic switching off of a previously uncontrolled lamp after a certain time. The building control must have the times of all controlled lamps available after the extension.	Not suitable	**Suitable** Data from retain variables are preserved as far as possible after an online change.	**Suitable** as far as possible [2] Preferred use case	Possible if textual, but complicated
	2c: The application must maintain values after a download.	**Suitable**	Not suitable	**Suitable**	**Suitable**

	Uses case	(A) Persistent variables	(B) Retain variables	(C) Variables of the Persistence Manager	(D) Recipe variables
3	**The application must be able to use different value sets.** Example: The operating settings for summer, winter, and holidays must be saved and imported when needed.	Not suitable	Not suitable	Not suitable	**Suitable** Preferred use case
4	**The application must be able to use settings from another system.** It must be possible to transfer settings to another plant using similar variables.	Not suitable	Not suitable	**Suitable**[2]	Suitable[3]
5	**The application must provide human readable data.** The user must be able to read, compare, and edit the data.	Not suitable	Not suitable	**Suitable**[2]	Suitable[3]

[1] Disadvantage: Only possible if the runtime system supports this mechanism and an NVRAM memory or UPS is available. Advantage: Speed; recommended application: 1 and 2a

[2] Disadvantage: In the case of large variable sets (> 10000), long delays during initialization and shutdown are to be expected. Advantage: No special memory is required; value retention exists even in case of changes, extensions, or deletions.

[3] Advantage: Editable remotely, transferable. Disadvantage: Complicated

Lifespan of variables when calling online commands

User input in the *"Online"* menu	Variable with usual lifespan
	Neither `RETAIN` nor `PERSISTENT`
		`RETAIN`	`PERSISTENT`
			`RETAIN PERSISTENT`
			`PERSISTENT RETAIN`

Command *"Online Change"*	x	x	x
Command *"Reset Warm"*	i	x	x
Command *"Reset Cold"*	i	i	x
Command *"Download"*	i	i	x [1]
Command *"Reset Origin"*	i	i	i

x : The variable retains its value.

i : The variable is initialized.

[1] Note: For the structure of persistent data, see the information in "Mechanism for downloading".

See also

- *"Mechanism for downloading" on page 304*
- *Chapter 1.4.1.20.3.6.6 "Command 'Online Change'" on page 1033*
- *Chapter 1.4.1.20.3.6.12 "Command 'Reset Origin'" on page 1039*
- *Chapter 1.4.1.20.3.6.10 "Command 'Reset Cold'" on page 1038*
- *Chapter 1.4.1.20.3.6.5 "Command 'Load'" on page 1032*

Lifespan of variables when downloading a boot project

The values of ordinary variables lose their value and are reinitialized.

The values of persistent variables are protected when:

- The structure of the persistent variable in memory matches the structure in the persistent data list.

The values of retain variables are protected when:

- The structure of the persistent variable in memory matches the structure in the persistent data list.
- The persistent variables match the application (GUID has to agree).

A "Retain mismatch" occurs when the requirements for restoring the values of retain variables and persistent variables are not met when the application is booted. The response to this discrepancy is described in the documentation of the hardware manufacturer.

Note: For the structure of persistent data, refer to the information in "Mechanism for downloading".

See also

- *“Mechanism for downloading” on page 304*

Preserving data with persistent variables

Persistent variables retain their values after reloading the application, and after a download, warm start, or cold start.

A special non-volatile memory area on the controller, for example as NVRAM or UPS, is required to extend the lifespan. Securing the data in such a memory does not require any additional time, which is an advantage over data retention with the Persistence Manager. If the controller does not provide hardware support, then the data is usually stored in a file. Then the data will be retained if you shut down the controller correctly. In the event of a power failure or a pulled plug, however, data will be lost.

Behavior

Value retained for

- Uncontrolled exit
- Warm start by calling the *“Reset Warm”* command
- Cold start by calling the *“Reset Cold”* command
- Repeated download of the application

Reinitialization for

- Call of the *“Reset Origin”* command

Therefore, persistent variables are reinitialized only if you reset the controller to the factory settings (for example, when you click *“Online ➔ Reset Origin”*).

If, on the other hand, you download the application again, the persisted data is retained if possible. That depends on how profound the changes that led to the download were. Changing the application name always leads to a full reinitialization. Changes to the implementations never lead to a reinitialization: the data persistence is completely preserved. Changes to the declarations lead to an initialization of the new variables only if the existing variables are persistent, when you change the declarations so that the persistent variable list remains consistent. This is the case when you add a new variable or delete an existing one. Inconsistencies can occur if you edit and change the identifiers or data types of previously declared persistent variables.

Mechanism for downloading

Editing the variable list in the persistence editor causes the variable list to be edited automatically before it is saved, not to be saved as it is shown in the editor.

During post-processing, a variable that you have removed is replaced by a placeholder variable with the same memory requirement. As a result, the subsequent variables retain their addresses in the process image. Moreover, a variable you add is moved to the end of the list. Post-processing can neutralize changes that would lead to a loss of persistence. But you create gaps that use additional memory.

When downloading, the CRC value of the variable list and the length of the list (number of variables) are stored on the controller. When downloading again, the new test value is compared with the test value currently on the controller. Then the variable list is compared successively up to the specified length. If you have edited a declaration (for example, the name or data type), then the variable is reinitialized. Otherwise its value is retained. When the download is repeated, CODESYS checks whether the variable list declared in the persistence editor is still consistent with the variable list already on the controller.

The mechanism works well when the variables themselves are not modified significantly. Too extensive changes of the identifiers and the data types continue to lead to a reinitialization and the loss of persistence. If you anticipate frequent changes due to your application requirements, then this kind of a list is not recommended. Moreover, in an online change after a data type change, a persistent variable is less robust than a variable with a normal lifespan.

It is good practice to clear any gaps in the variable list after a while (command *“Reorder List and Clear Gaps”*). After cleaning, however, the list no longer matches the list on the controller and you have triggered an initialization of all persistent variables. The persistence of all variables is lost.

For versions before V3.5 SP1, changes in the persistence editor always lead to reinitialization.

Recovering data with the recipe manager

To clean up the global persistent variable list without losing persistence, you can save the data in a recipe using the Recipe Manager. This creates a list for all variables of the persistent variable list in the recipe manager, and at the same time its current values are stored by the controller as a recipe. Then execute the command *“Reorder List and Clear Gaps”* and perform a download again. Now when you execute the command *“Restore Values from Recipe”*, the values saved in the recipe are restored.

Changing an existing declaration in the persistent variable list

if you change the name or data type of a variable, this is interpreted as a new declaration and causes a re-initialization of the variables at the next online change or download. For complex data types, a change occurs when a new component is added, or when you change the type of a variable from `INT` to `UINT` in the depth of a used structure used, for example.

Basically, complex user-defined data types are not suitable for administration in a persistent variable list, because even small changes cause the variable to be initialized with all components.

Double allocation of memory in the case of instance paths

You can persist global variables or variables declared locally in a function block or program. To do this, add the keyword `PERSISTENT` to the declaration. In addition, you insert the instance path to this variable in the persistent global variable list. To do this, execute the *“Add All Instance Paths”* command in the persistence editor.

Persistence is guaranteed by the following mechanism:

- The cyclic tasks in which the variable is accessed are determined.
- At the end of the first cyclic task (in each cycle), the variable is copied to the persistent global variable list.
- After restarting the controller, the value of the persistent variable is copied to the ordinary variable.

The disadvantage of this mechanism is that memory is allocated both at the place of declaration and at the place of the instance path. This persistent variable has a **double** memory allocation. Moreover, the data is copied to both places in each cycle. This can be time consuming, especially when large structured values are involved.

Memory location in the case of persistent function block instances

A function block instance is always stored completely in memory. This is necessary so that the same code can work on different instances. If only one variable in a function block is marked with `PERSISTENT`, then the function block instance is stored completely with all variables in remanent memory, although only the one variable is treated as persistent. However, non-volatile memory is not available to the same extent as main memory.

A function block with a pointer to an instance in SRAM as a variable is not stored in the protected memory.

Importing from CoDeSys V2.3 projects

When you open a CoDeSys V2.3 project to import it into CODESYS V3, the declarations of persistent variables are not preserved. You have to revise the declarations and create then again in a separate persistent global variable list.

See also

- *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*
- *Chapter 1.4.1.20.3.17.4 “Command 'Add all instance paths'” on page 1124*
- *Chapter 1.4.1.2.2 “Opening a V2.3 project” on page 187*

Preserving data with retain variables

Retain variables preserve their values after a warm start. However, the degree of value retention for persistent variables is higher.

A special non-volatile memory area on the controller, for example as NVRAM or UPS, is required to extend the lifespan. Securing the retain variables in such a memory does not require any additional time, which is an advantage over data retention with the Persistence Manager. If the controller does not provide hardware support, then the data is usually stored in a file. Then the data will be retained if you shut down the controller correctly. In the event of a power failure or a pulled plug, however, data will be lost.

Declaration

To declare a retain variable, add the `RETAIN` keyword to a variable declaration.

Behavior

Value retained for

- Uncontrolled exit
- Call of the *“Reset Warm”* command

Reinitialization for

- Repeated download of the application
- Call of the *“Reset Cold”* command (in contrast to persistent variables)
- Call of the *“Reset Origin”* command

When you restart an application, its variables are usually initialized with an explicitly preset initial value or with a default value. Variables marked with the `RETAIN` keyword are managed in a separate memory area depending on the target system and retain their value. Then the variables are protected from power failure, for example. This means that you can apply retain variables to a parts counter in a production line so that you can continue counting even after a power failure.

Memory location of persistent function block instances

Function block instances are stored as one block in memory. This is necessary so that the same code can work on different instances. If a variable is marked with `RETAIN` in a function block, then each instance of the function block is protected with all variables. This is also true for the variables of the function block that are not marked this way. However, non-volatile memory is not available to the same extent as main memory.

A function block with a pointer to an instance in SRAM as a variable is not stored in the protected memory.

Importing of CoDeSys V2.3 projects

When you open a CoDeSys V2.3 project to import it into CODESYS V3, the declarations of retain variables are preserved and remain effective as before.

See also

- *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537*
- *Chapter 1.4.1.2.2 “Opening a V2.3 project” on page 187*

Retaining data with variables of the persistence manager

Persistent variables are managed in the Persistence Manager of the CODESYS Application Composer. The functionality of the *“Persistence Manager”* does not need any special memory on the controller in order to preserve values and data.

Declaration

In the declarations, the variables managed in the Persistence Manager are marked with the pragma `{attribute 'ac_persist'}`.

The pragma makes sure that the variable with this attribute is managed in the Persistence Manager of the Application Composer. The variable value is retained even if you change the declaration of the variable, delete a variable from the application, or add a new one. The value is retained even if you change the data type and use the appropriate conversions.

Mechanism

The variables of the Persistence Manager are stored with their values in an external archive file in TXT format.

The application code is extended with the code of the Persistence Manager, which leads to a greater memory requirement. This is at the expense of performance. Moreover, reading and especially writing a large number of persistent variables can take a long time. As a result, the executing task also blocks the execution for a long time.

Functionality

- You can load and edit the TXT file in an external editor such as Notepad++.
- You can use the persistent variables of the file in another application.
- You can configure the behavior of persistent variables by defining persistence groups, assigning variables to them, and configuring the groups with their own save and read behavior.

Preserving data with recipes

Variables are managed persistently in the Recipe Manager. The Recipe Manager does not need any special memory on the controller in order to preserve values and data.

Declaration

A recipe definition consists of a set of variables with values and is created and edited in the *“Recipe Manager”* object and saved to a file.

Functionality

- You can include a variable in multiple recipes, each with different values.
- In online mode, you can read in the actual values of the variables from the controller and save them as recipe values (specified value).
- You can use the `Recipe Management` library to programmatically implement the creation and editing of a recipe.
- You can save and backup a recipe as a recipe file.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.17.2 “Command 'Save Current Values to Recipe'” on page 1123*
- *Chapter 1.4.1.20.3.17.1 “Command 'Reorder List and Clean Gaps'” on page 1123*
- *Chapter 1.4.1.20.3.17.3 “Command 'Restore Values from Recipe'” on page 1123*
- *Chapter 1.4.1.20.2.22 “Object 'Recipe Manager'” on page 923*
- *Chapter 1.4.1.20.2.23 “Object 'Recipe Definition'” on page 926*

Declaring VAR PERSISTENT Variables

Below you will declare persistent variables in a persistent variable list and in a POU.

Requirement: A project is opened and contains a program POU. You have selected the option for the textual view in the *“Declaration Editor”* category of the options (menu command in *“Tools ➔ Options”*).

1. Add the *“Persistent Variables”* object to the application object with the menu command *“Project ➔ Add Object”*.
   ⇨ CODESYS adds the persistent variable list *“PersistentVars”* below the application object in the device tree and the editor opens.
2. In the editor, enter a variable declaration, for example `ivarpersist1 : INT;` between `VAR_GLOBAL PERSISTENT RETAIN` and `END_VAR`.
3. Double-click the POU in the device tree.
   ⇨ The editor of the POU opens.
4. Specify the following declaration in the declaration part:

   ```
   VAR PERSISTENT RETAIN
   ivarpersist2 : INT;
   END_VAR
   ```

5. Click *“Build ➔ Build”*.
   ⇨ The message view opens. If CODESYS has compiled the application without errors, then close the message window and continue with the next step. Otherwise, correct the error(s) and select the menu command *“Build ➔ Build”* again.
6. Set the focus in the *“PersistentVars”* editor. Click *“Declarations ➔ Add All Instance Paths”*
   ⇨ CODESYS adds the persistent variable from the persistent variable list *“PersistentVars”* to the POU:

   ```
   // instance path of the persistent variables created
   POU.IVARPERSIST2 : INT
   ```

See also

- *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*
- *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537*
- *Chapter 1.4.1.20.2.12 “Object 'Persistent variable list'” on page 872*
- *Chapter 1.4.1.20.3.17.4 “Command 'Add all instance paths'” on page 1124*

Saving the values of a persistent variable list in a recipe

Requirement: a project is opened and a persistent variable list with declarations of persistent variables exists under an application object.

1. Double-click on the controller in the device tree and select the tab *“Communication Settings”*.
2. Select your gateway and click on the button *“Scan Network”*.
   ⇨ Your device is shown in bold in the tree view of the gateway.
3. Select your device and click on the button *“Set Active Path”*.
4. Select your application object in the device tree and select the context menu command *“Set Active Application”*.
   ⇨ The application object is displayed in bold.
5. Select the menu command *“Online ➔ Login”*
   ⇨ Your application is logged in to the controller and the controller and the application object in the device tree have a green background.
6. Double-click on the persistent variable list and select the command *“Declarations ➔ Save Current Values to Recipe”*.
   ⇨ CODESYS creates the objects *“Recipe Manager”* and *“PersistentVariables”* under the application object.
7. Select the menu command *“Online ➔ Logout”*.
   ⇨ The application is logged out from the controller.

See also

- *Chapter 1.4.1.20.3.17.2 “Command 'Save Current Values to Recipe'” on page 1123*
- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

1.4.1.8.20 Alarm Management

For information about alarm management and alarm visualization, see the help for CODESYS Visualization.

1.4.1.8.21 Using POUs for implicit checks

CODESYS provides special POUs that implement implicit monitoring functions. At runtime, these functions check the array limits or subrange types, the validity of pointer addresses, or division by zero.

1. Select the *“Application”* object in the device tree.
   Click *“Project ➔ Add Object ➔ POU for Implicit Checks”*
   ⇨ The *“Add POU for Implicit Checks”* dialog box opens.
2. Select the desired functions.
3. Click *“Add”*.
   ⇨ The selected POUs are inserted below the *“Application”* in the device tree.
4. Open the POUs in the editor.

5. Adapt the implementation suggestion to your requirements.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

See also

- *Chapter 1.4.1.20.2.18 “Object 'POU'” on page 881*
- *Chapter 1.4.1.20.2.19 “Object 'POUs for Implicit Checks'” on page 904*

1.4.1.8.22 Object-Oriented Programming

CODESYS supports object oriented programming with function blocks and for this purpose provides the following features and objects:

- Methods
- Interfaces
- Properties
- Inheritance
- Method call, virtual function call
- Definition of function blocks as extensions of other function blocks

Basic information on dealing with object-oriented programming with AC500 V3 PLCs is given in the *application example*.

See also

- *Chapter 1.4.1.20.2.18.4 “Object 'Interface'” on page 888*

Extension of function blocks

The extension of a function block is based on the concept of inheritance in object-oriented programming. A derived function block thereby extends a basic function block and in doing so is given the properties of the basic function block in addition to its own properties.

The extension of a function block means:

- The inherited function block contains all data and methods that are defined by the basic function block. You can use an instance of the basic function block in every context in which CODESYS expects a function block of the type of the basic function block.
- The derived function block can overwrite the methods that you have defined in the base function block. This means that the inherited function block can define a method with the same name, the same inputs and the same output as is defined by the basic function block.
  Tip: You have the following support when overwriting methods, actions, attributes, and transitions that are inherited by the base block: When you insert a method, action, etc. below an inherited block, the *“Add Object”* dialog includes a combo box with a list of methods, actions, etc. used in the base block. You can accept these and adapt them accordingly.
- The derived function block may not contain function block variables with the same names as used by the basic function block. The compiler reports this as an error.
  The only exception: If you have declared a variable in the basic function block as `VAR_TEMP`, then the inherited function block may define a variable with the same name. In this case, the inherited function block can no longer access the variable of the basic function block.
- You can directly address the variables and methods of the basic function block within the scope of the inherited function block by using the `SUPER` pointer.

NOTICE!
Multiple inheritance is not permitted.

Exception: A function block can implement multiple interfaces, and an interface can extend other interfaces.

Extension of a basic function block by a new function block

Requirement: the currently opened project possesses a basic function block, for example *"POU_1(FB)"*, which is to be extended by a new function block.

1. Right-click the *"Application"* object in the device tree and select *"Project ➔ Add Object ➔ POU"*.
   ⇨ The *"Add POU"* dialog opens.
2. Type the name for the new POU in the *"Name"* input field, for example *"POU_Ex"*.
3. Select *"Function block"*.
4. Click *"Advanced"* and then the more button (...).
5. In the category *"Function blocks"* under *"Application"* in the input assistant, select the POU(FB) that is to serve as the basic function block, for example `POU_1`, and click *"OK"*.
6. As an option, you can select an *"Access modifier"* for the new function block from the drop-down list.
7. Select from the *"Implementation language"* combo box (example: *"Structured text (ST)"*.
8. Click *"Add"*.
   ⇨ CODESYS adds the `POU_Ex` function block to the device tree and opens the editor. The first line contains the text:

   ```
   FUNCTION_BLOCK POU_Ex EXTENDS POU_1
   ```

   The function block `POU_Ex` extends the basic function block `POU_1`.

Extension of a basic function block by an existing function block

Requirement: The open project possesses a base function block (example: `POU_1(FB)`) and another function block (example: `POU_Ex(FB)`). The function block `POU_Ex(FB)` is also to be given the properties of the basic function block. This means that `POU_Ex(FB)` should extend `POU_1(FB)`.

1. Double-click the function block `POU_Ex(FB)` in the device tree.
   ⇨ The function block editor opens.
2. Extend the existing entry in the top line `FUNCTION_BLOCK POU_Ex` with `EXTENDS POU_1`.
   ⇨ The function block POU_Ex extends the basic function block `POU_1`.

See also

- *Chapter 1.4.1.8.22.2 "Implementing interfaces" on page 312*
- *Chapter 1.4.1.8.22.3 "Extending interfaces" on page 314*
- *Chapter 1.4.1.19.2.14 "SUPER" on page 538*
- *Chapter 1.4.1.19.2.15 "THIS" on page 539*
- *Chapter 1.4.1.20.2.18.2 "Object 'Function Block'" on page 883*
- *Chapter 1.4.1.20.2.18.8 "Object 'Property'" on page 897*
- *Chapter 1.4.1.20.2.18.9 "Object 'Action'" on page 901*
- *Chapter 1.4.1.20.2.18.10 "Object 'Transition'" on page 903*

Implementing interfaces

Implementing interfaces is based on the concept of object-oriented programming. With common interfaces, you can use different but similar function blocks the same way.

A function block that implements an interface has to include all methods and attributes that are defined in that interface (interface methods and interface attributes). This means that the name and the inputs and outputs of the methods or attributes must be exactly the same. When you create a new function block that implements an interface, CODESYS adds all methods and attributes of the interface automatically to the tree below the new function block.

NOTICE!

If you add more interface methods afterwards, then CODESYS does not add these methods automatically to the affected function block. To perform this update, you must execute the *"Implement Interfaces"* command explicitly.

For inherited function blocks, you have to make sure that any methods or attributes that were derived through the inheritance of an interface also receive the appropriate implementation. Otherwise they should be deleted in case the implementation that was provided in the basis should be used. Respective compile error messages or warnings are displayed, prompted automatically by added pragma attributes. For more information, refer to the help page for the *"Implementing Interfaces"* command.

NOTICE!

- You must assign the interface of a function block to a variable of the interface type before a method can be called via the variable.
- A variable of the interface type is always a reference of the assigned function block instance.

A variable of the interface type is a reference to instances of function blocks. This kind of variable can refer to every function block that implements the interface. If there is no assignment to a variable, then the variable in online mode contains the value `0`.

Examples

The `I1` interface contains the `GetName` method.

```
METHOD GetName : STRING
```

The functions blocks `A` and `B` implements the interface `I1`:

```
FUNCTION_BLOCK A IMPLEMENTS I1
FUNCTION_BLOCK B IMPLEMENTS I1
```

For this reason, both function blocks must include a method named `GetName` and the return type `STRING`. Otherwise the compiler reports an error.

A function includes the declaration of a variable of interface `I1` type.

```
FUNCTION DeliverName : STRING
VAR_INPUT
    l_i : I1;
END_VAR
```

Function blocks that implement the `I1` interface can be assigned to these input variables.

Examples of function calls:

```
DeliverName(l_i := A_instance); // call with instance of type A
DeliverName(l_i := B_instance); // call with instance of type B
```

Calling of interface methods:

In this case, it depends on the actual type of `l_i` whether the application calls `A.GetName` or `B.GetName`.

```
DeliverName := l_i.GetName();
```

- ↳ *Chapter 1.4.1.20.3.22.2 “Command 'Implement Interfaces'” on page 1148*

Implementing an interface in a new function block

☑ Requirement: The open project has at least one interface object.

1. Right-click *“Application”* in the device tree and select *“Project ➔ Add Object ➔ POU”*.
   ⇨ The *“Add POU”* dialog box opens.
2. Type the name for the new POU in the *“Name”* input field, for example *“POU_Im”*.
3. Select *“Function block”*.
4. Click *“Implemented”* and then the more button (...).
5. In the input assistant, select the interface from the category *“Interfaces”*, for example `ITF1`, and click on *“OK”*.
6. To insert more interfaces, click ... and select a another interface.
7. As an option, you can select an *“Access modifier”* for the new function block from the selection list.
8. Select from the *“Implementation language”* combo box (example: *“Structured text (ST)”*.
9. Click *“Add”*.
   ⇨ CODESYS adds the *“POU_Ex”* function block to the device tree and opens the editor. The first line contains the text:

   ```
   FUNCTION_BLOCK POU_Im IMPLEMENTS ITF1
   ```

   The interface and its methods and properties are now inserted below the function block in the device tree. Now you can type program code into the implementation part of the interface and its methods.

Implementing an interface in an existing function block

☑ Requirement: The currently open project has a function block (example: *"POU_Im"*) and at least one interface object (example: *"ITF1"*).

1. Double-click the *"POU_Ex(FB)"* POU in the device tree.
   ⇨ The POU editor opens.
2. Extend the existing entry in the uppermost line `FUNCTION_BLOCK POU_Im` with `IMPLEMENTS ITF1`
   ⇨ The *"POU_Im"* function block implements the *"ITF1"* interface.

See also

- *Chapter 1.4.1.20.2.18.2 "Object 'Function Block'" on page 883*

Extending interfaces

You can extend interfaces just like function blocks. The interface is then also given the interface methods and interface properties of the basic interface in addition to its own.

Creation of an interface that extends another interface.

1. Select the object *"Application"* in the device tree.
2. Select the command *"Project ➔ Add Object ➔ Interface"*.
   ⇨ The dialog box *"Add Interface"* opens.
3. Enter a name for the new interface.
4. Activate the option *"Extended"* and click on the button ....
5. The input assistant opens.
6. From the category *"Interfaces"*, select the interface that is to be extended by the new interface.

- *Chapter 1.4.1.20.2.18.4 "Object 'Interface'" on page 888*

Calling methods

To implement a method call, the actual parameters (arguments) are passed to the interface variables. As an alternative, the parameter names can be omitted.

Depending on the declared access modifier, a method can be called only within its own name-space (`INTERNAL`), only within its own programming module and its derivatives (`PROTECTED`), or only within its own programming module (`PRIVATE`). For `PUBLIC`, the method can be called from anywhere.

Within the implementation, a method can call itself recursively, either directly by means of the `THIS` pointer, or by means of a local variable for the assigned function block.

Method call as a virtual function call

Virtual function calls can occur due to inheritance.

Virtual function calls enable one and the same call to call various methods in a program source code during the runtime.

In the following cases the method call is dynamically bound:

- You call a method via a pointer to a function block (for example `pfub^.method`).
  In this situation the pointer can point to instances of the type of the function block and to instances of all derived function blocks.
- You call the method of an interface variable (for example `interface1.method`).
  The interface can refer to all instances of function blocks that implement this interface.
- A method calls another method of the same function block. In this case the method can also call the method of a derived function block with the same name.
- The call of a method takes place by means of a reference to a function block. In this situation the reference can point to instances of the type of the function block and to instances of all derived function blocks.
- You assign `VAR_IN_OUT` variables of a basic function block type to an instance of a derived FB type.
  In this situation the variable can point to instances of the type of the function block and to instances of all derived function blocks.

Example

Overloading methods

The function blocks `fub1` and `fub2` extend the function block `fubbase` and implement the interface `interface1`. The methods `method1` and `method2` exist.

```
PROGRAM PLC_PRG
VAR_INPUT
  b : BOOL;
END_VAR

VAR  pInst : POINTER TO fubbase;
  instBase : fubbase;
  inst1 : fub1;
  inst2 : fub2;
  instRef : REFERENCE to fubbase;
END_VAR

IF b THEN
  instRef REF= inst1;            (* reference to fub1 *)
  pInst := ADR(instBase);
ELSE
  instRef REF= inst2;            (* reference to fub2  *)
  pInst := ADR(inst1);
END_IF
pInst^.method1();            (* If b is TRUE, fubbase.method1 will
be called, otherwise fub1.method1 is called *)
instRef.method1();        (* If b ist TRUE, fub1.method1 will be
called, otherwise fub2.method1 is called*)
```

On the assumption that `fubbase` in the above example contains two methods `method1` and `method2`, it overwrites `fub1 method2`, but not `method1`. The call of `method1` takes place as follows:

```
pInst^.method1();
```

If `b` is TRUE, then CODESYS calls `fubbase.method1`. If not, then `fub1.method1` is called.

Additional outputs

In accordance with the IEC 61131-3 standard, methods can have additional outputs declared, like normal functions. With the method call, you assign variables to the additional outputs.

Detailed information about this can be found in the topic “Function”.

Syntax for the call:

```
<function block name>.<method name>(<first input name> := <value> (,
<further input assignments>)+ , <first output name> => <first output
variable name> (,<further output assignments>)+ );
```

Example

Declaration

```
METHOD PUBLIC DoIt : BOOL
VAR_INPUT
    iInput_1 : DWORD;
    iInput_2 : DWORD;
END_VAR
VAR_OUTPUT
    iOutput_1 : INT;
    sOutput_2 : STRING;
ENDVAR
```

Call

```
fbInstance.DoIt(iInput_1 := 1, iInput_2 := 2, iOutput_1 =>
iLocal_1, sOUtput_2 => sLocal_2);
```

When the method is called, the values of the method outputs are written to the locally declared output variables.

Calling a method even if the application is in the STOP state

In the device description it is possible to define that a certain function block instance (of a library function block) always calls a certain method in each task cycle. If the method contains the input parameters of the following example, CODESYS processes the method even if the active application is presently in the STOP state:

Example

```
VAR_INPUT
    pTaskInfo : POINTER TO DWORD;
    pApplicationInfo: POINTER TO _IMPLICIT_APPLICATION_INFO;
END_VAR

(*Now the status of the application can be queried via
pApplicationInfo and the instructions can be implemented: *)
IF pApplicationInfo^.state = RUNNING THEN <instructions> END_IF;
```

Calling methods recursively

Use recursions mainly for processing recursive data types such as linked lists. Generally, we recommend that you be careful when using recursion. An unexpectedly deep recursion can lead to stack overflow and therefore to machine downtime.

Within their implementation, a method can call itself:

- Directly by means of the `THIS` pointer
- Indirectly by means of a local function block instance of the basic function block

Usually, a compiler warning is issued for such a recursive call. If the method is provided with the pragma `{attribute 'estimated-stack-usage' := '<sstimated_stack_size_in_bytes>'}`, then the compiler warning is suppressed. For an implementation example, refer to the section "Attribute 'estimated-stack-usage'".

See also

- *Chapter 1.4.1.19.6.2.13 “Attribute 'estimated-stack-usage'” on page 695*
- *Chapter 1.4.1.19.2.15 “THIS” on page 539*
- *Chapter 1.4.1.19.2.14 “SUPER” on page 538*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*

1.4.1.8.23 Motion Solution

Basic Motion

Cams

The SoftMotion cam is integrated in the development interface of CODESYS. In the cam editor, cams and tappets can be implemented graphically or by means of tables. As soon as code is generated for the corresponding application, global data structures ("Cam Data") are created which the IEC program can access. For this purpose, the `SM3_Basic` is also linked automatically into the project when inserting a SoftMotion drive.

See also

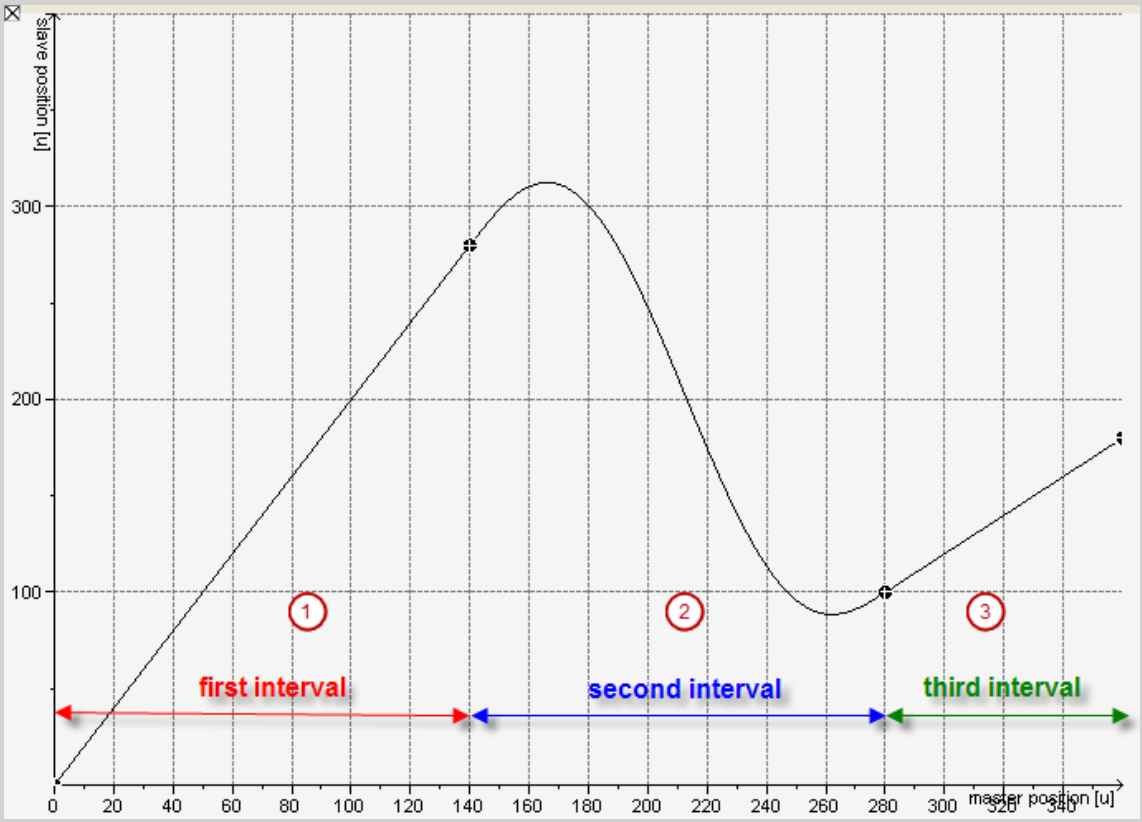
- *Chapter 1.4.1.8.23.1.1.1 “Definition of a SoftMotion Cam” on page 317*
- *Chapter 1.4.1.8.23.1.1.3 “Creating Cams” on page 319*

Definition of a SoftMotion Cam

A cam describes the functional dependency of one drive (slave) on another drive (master). The relationship is described by a continuous function (or curve) that maps a defined range of master values to slave values. To be more precise: After dividing the master axis into suitable segments, the graph of these functions can be represented on each of these intervals by a line or a 5th degree polynomial.

Example

The master values are applied to the horizontal axis and the slave values to the vertical axis in the cam graph.



In the example, the master values are between 0 and 360. This range is divided into three intervals:

- (1) First interval: [0, 140]
- (2) Second interval: [140, 280]
- (3) Third interval: [280, 360]

The function (graph) is linear in the first and third intervals and its graph is displayed as a line. As a result, its first derivative (slope) is constant and all higher derivatives are 0.

In the second interval, the graph is described by a 5th degree polynomial. Therefore, its first derivative is a 4th degree polynomial, its second derivative (curvature) is a 3rd degree polynomial, and its third derivative is a 2nd degree polynomial, etc.

When the function describes the movement of the slave depending on the position of the master, its first derivative corresponds to the velocity of the slave and the second derivative to its acceleration.

When you keep this physical interpretation in mind, it is obvious that the mapping has to be continuous. This means that its graph is not allowed to have any jumps. In particular, the continuity also has to be fulfilled at each point where two intervals meet. Furthermore, the continuity in general is also required by the first and second derivative. (In fact, these three continuity conditions at the start and end points of an interval determine the coefficients of the 5th degree polynomial inserted between two straight segments.

Moreover, you may add tappets (binary switches) to the cam at any position. In this way, you can create cam tables which contain tappets only. The slave position is then set to zero over the entire master value range.

Compiling cam definitions

At compile time, variables of type `MC_CAM_REF` are created for a cam. They include a description of each segment of the cam. Data structures of this kind are passed to the `MC_CamTableSelect` function block. The structure is part of the `SM3_Basic` library.

See also

- *MC_CAM_REF*
- *MC_CamTableSelect*

Structure of the Cam Editor

Open the cam editor by double-clicking the *“Cam”* object in the device tree.

The editor consists of the following tabs:

- Tab *“Cam”*: Includes a graphical editor for creating a cam path. Here, you can display and modify the slave position, slave velocity, slave acceleration, and slave jerk. In the graphical editor, you recognize very quickly when you program a movement with high acceleration.
- Tab *“Cam table”*: Includes an editor for listing base points in a table. Here, you can specify the exact positions and velocities.
- Tab *“Tappets”*: Includes an editor for programming tappets (switch points) in a diagram. This display provides a very good overview of the sequential order of the tappets.
- Tab *“Tappet table”*: Includes an editor for listing switch points in a table. Here, you can specify the exact switch points.

The tabs are split into an editor, as well as a *“ToolBox”* view and *“Properties”* view.

See also

- *Chapter 1.4.1.8.23.3.1.1.1.1 “Tab 'Cam'” on page 344*
- *Chapter 1.4.1.8.23.3.1.1.1.2 “Tab 'Cam table'” on page 345*
- *Chapter 1.4.1.8.23.3.1.1.1.3 “Tab 'Tappets'” on page 346*
- *Chapter 1.4.1.8.23.3.1.1.1.4 “Tab 'Tappet table'” on page 347*

Creating Cams

The steps for creating a cam are explained by means of a sample application that describes a rotary table with eight slots (45° division). Inside, there is a component that is fused ultrasonically. The welding tool is fed in by a linear drive after the rotary table has turned. After welding, the linear axis returns and the rotary table continues turning.

Work steps

- Rotary table turns 45° (duration: 400 ms).
- The welding head is moved down by a vertical axis of 250 mm (duration: 200 ms).
- Start welding (duration: 1200 ms).
- The welding head is moved up by a vertical axis of 250 mm (duration: 200 ms).

A cycle time of 2000 ms results from total times.

The application is implemented by means of a virtual master axis that runs continuously (modulo). The end value of the axis is projected according to the cycle time of 2000 ms. The rotary table is achieved as a cam (modulo; end value: 45°). The vertical axis is also achieved as a cam (restricted; end value: 300 mm). The welding process is controlled by a tappet.

See also

- *Chapter 1.4.1.8.23.3.1.1.1.5 “Dialog 'Properties - 'Cam'” on page 348*

Adding a cam to the device tree

Requirement: A SoftMotion controller is selected.

1. Select the *“Application”* object in the device tree.
2. Click *“Project ➔ Add object ➔ Cam table”*.
3. Specify the name *“Rotary table”* for the cam and click *“OK”*.
   ⇨ The object is inserted into the device tree. The cam editor opens.

4. Insert another cam named *“Vertical axis”*.

Setting the properties of the cam

1. Select the *“Rotary table”* cam in the device tree.
2. Click *“Properties”* in the *“View”* menu or in the context menu.
3. Select the *“Cam”* tab.
4. Specify the following values:
   - *“Master start position”*: 0
   - *“Master end position”*: 2000
   - *“Slave start position”*: 0
   - *“Slave end position”*: 45
   - *“Smooth transition”*: ☐ (deactivated)
5. Click *“OK”* to close the dialog. Confirm the dialog for changing the cam object.
6. Change the values for the *“Vertical axis”* cam in the same way:
   - *“Master start position”*: 0
   - *“Master end position”*: 2000
   - *“Slave start position”*: 0
   - *“Slave end position”*: 300
   - *“Smooth transition”*: ☑ (activated)
7. Click *“OK”* to close the dialog. Confirm the dialog for changing the cam object.

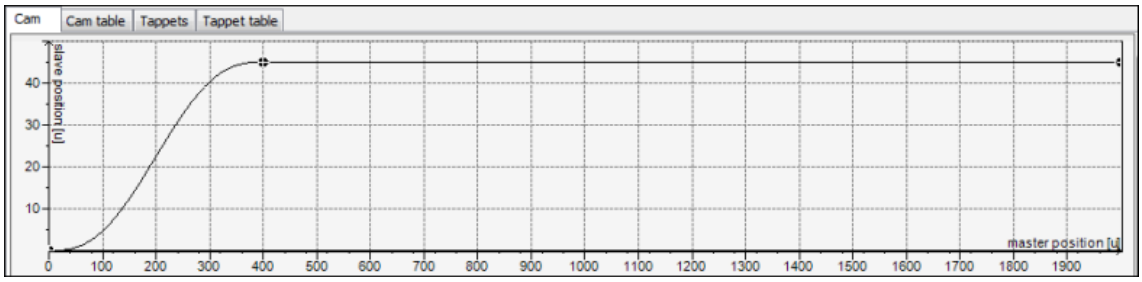
Changing the Cam Path

These instructions use the example from the section "Creating Cams" to demonstrate how to change a cam.

Changing the path with the graphical editor

1. Open the *“Rotary table”* cam in the editor.
   ⇨ The *“Cam”* tab is visible.
2. Select the point at 120 and delete it by pressing the delete key (*[Del]*). Also delete the point at 240.
3. Select the *“Add point”* tool from the *“ToolBox”* view.
   ⇨ The mouse pointer turns into crosshairs when you move it into the editor.
4. Click near *“Master position”* 400 and *“Slave position”* 45 in the upper graphs (slave position).
   ⇨ The curve of the slave position is changed. The curves of velocity, acceleration, and jerk also change.
5. Select the new inserted point by clicking it.
6. Drag the point to another position.
   ⇨ The curve of the slave position is adjusted accordingly.
7. Change the *“X”* and *“Y”* properties to the exact values of 400 and 45, respectively.
8. In the same way, change the x-value to 45 of the point at master position 2000.
9. Select the *“Select”* tool from the *“ToolBox”* view.

10. Select the second curve element (between 400 and 2000).
11. Change the *“Segment type”* property to *“Line”*.
12. Check the curve in the graphical editor.
    ⇨ Display:



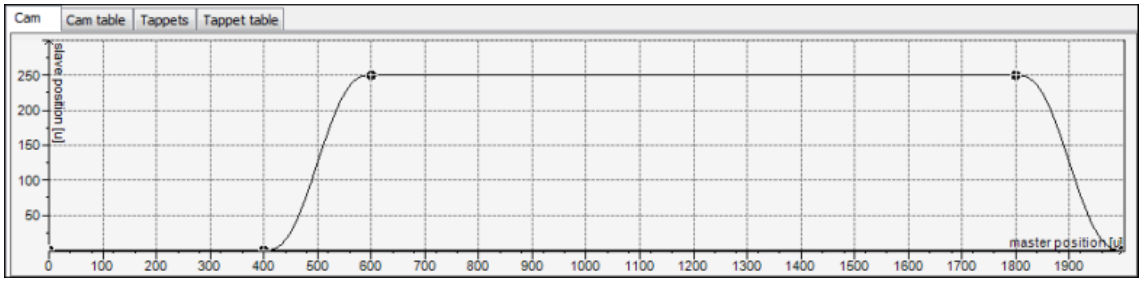
See also

- ↳ *Chapter 1.4.1.8.23.3.1.1.1.1 “Tab 'Cam'” on page 344*

Changing the path with a cam table

1. Open the *“Vertical axis”* cam in the editor.
   ⇨ The *“Cam”* tab is visible.
2. Select the *“Cam table”* tab.
3. Delete the point at 120 by clicking the symbol. Also delete the point at 240.
4. Click the symbol.
   ⇨ A new point and a new segment are inserted at (1000/150).
5. Add two more points.
6. Change the values X / Y of the following points:
   - Point 1: `0 / 0`
   - Point 2: `400 / 0`
   - Point 3: `600 / 250`
   - Point 4: `1800 / 250`
   - Point 5: `2000 / 0`

   ⇨ The curve of the slave position is changed. The curves of velocity, acceleration, and jerk also change.
7. In the cam table, change the *“Segment type”* of the first and third segments to *“Line”*.
8. Check the curve in the graphical editor.
   ⇨ Display:



In practice, the curves of the different cams are defined frequently as overlapping in order to save on cycle time. In the example above, the vertical axis could already begin the movement while the rotary table is still in motion (for example, at X: 350).

See also

- *Chapter 1.4.1.8.23.3.1.1.1.2 “Tab 'Cam table'” on page 345*

Displaying generated code

By clicking “Display generated code”, you can display the automatically created global variables.

```
{attribute 'linkalways'}

VAR_GLOBAL
Vertical_axis_A: ARRAY[0..4] OF SMC_CAMXYVA := [
    (dX := 0, dY := 0, dV := 0, dA := 0),
    (dX := 400, dY := 0, dV := 0, dA := 0),
    (dX := 600, dY := 250, dV := 0, dA := 0),
    (dX := 1800, dY := 250, dV := 0, dA := 0),
    (dX := 2000, dY := 0, dV := 0, dA := 0)];
Vertical_axis: MC_CAM_REF := (nElements := 5, byType := 3, xStart :=
0, xEnd := 2000, nTappets := 2, strCAMName := 'Vertical_axis', pce :=
ADR(Vertical_axis_A), pt := ADR(Vertical_axis_T), xPartofLM := TRUE);
END_VAR
```

See also

- *Chapter 1.4.1.8.23.3.1.2.1.1 “Command 'Display generated code'” on page 350*

See also

- *Chapter 1.4.1.8.23.1.1.3 “Creating Cams” on page 319*

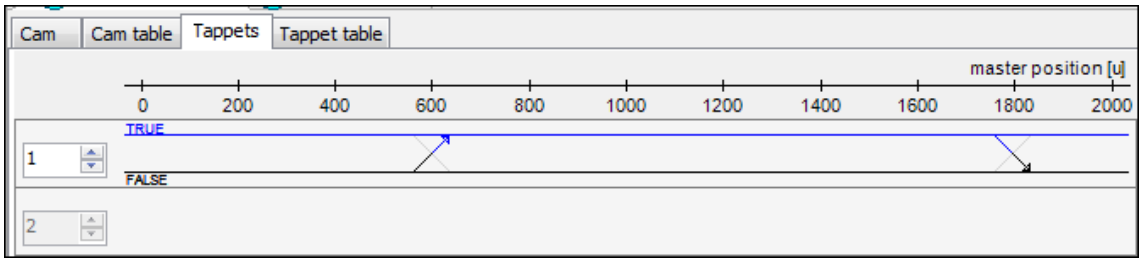
Defining Switch Points

Use switch points to trigger events depending on the master position. For example, this can be the setting of an output or the calling of a function block.

These instructions use the example from the section "Creating Cams" to demonstrate how to define switch points. In this example, the tappet starts and stops the welding process.

1. Open the *“Vertical axis”* cam in the editor.
   ⇨ The *“Cam”* tab is visible.
2. Select the *“Tappets”* tab.
3. Select the *“Add tappet”* tool from the *“ToolBox”* view.
   ⇨ The mouse pointer turns into crosshairs when you move it into the editor.
4. Click below the master position near position 600.
   ⇨ A tappet is inserted to the tappet path 1.
5. Select the tappet.

6. Change the values of the tappet in the *“Properties”* view.
   - *“X”*: 600
   - *“Positiver pass”*: *“Switch ON”*
   - *“Negative pass”*: *“No action”*
7. Insert another tappet to tappet path 1 at X: 1800.
   - *“X”*: 1800
   - *“Positiver pass”*: *“Switch OFF”*
   - *“Negative pass”*: *“No action”*
8. Check the result.



You can also change the values for “Positive pass” and “Negative pass” by clicking the respective end of the crosshairs (⌧).

Please note the possibility of also setting switch points in the “Tappet table” tab. This editor provides you with the same options, but in tabular form.

See also

- *Chapter 1.4.1.8.23.1.1.3 “Creating Cams” on page 319*
- *Chapter 1.4.1.8.23.3.1.1.1.3 “Tab 'Tappets'” on page 346*
- *Chapter 1.4.1.8.23.3.1.1.1.4 “Tab 'Tappet table'” on page 347*

Displaying generated code

By clicking “Display generated code”, you can display the automatically created global variables.

```
{attribute 'linkalways'}

VAR_GLOBAL
Vertical_axis_A: ARRAY[0..4] OF SMC_CAMXYVA := [
    (dX := 0, dY := 0, dV := 0, dA := 0),
    (dX := 400, dY := 0, dV := 0, dA := 0),
    (dX := 600, dY := 250, dV := 0, dA := 0),
    (dX := 1800, dY := 250, dV := 0, dA := 0),
    (dX := 2000, dY := 0, dV := 0, dA := 0)];
Vertical_axis_T: ARRAY[0..1] OF SMC_CAMTappet := [
    (x := 597.32540861812777, ctt := 0, iGroupID := 1, cta := 0),
```

```
    (x := 1800, ctt := 0, iGroupID := 1, cta := 1)];
Vertical_axis: MC_CAM_REF := (nElements := 5, byType := 3, xStart :=
0, xEnd := 2000, nTappets := 2, strCAMName := 'Vertical_axis', pce :=
ADR(Vertical_axis_A), pt := ADR(Vertical_axis_T), xPartofLM := TRUE);
END_VAR
```

See also

- *Chapter 1.4.1.8.23.3.1.2.1.1 “Command 'Display generated code'” on page 350*

Important Cam Settings

The `SM3_Basic` library provides function blocks for handling cams. If you insert a SoftMotion drive into the device tree, then this library is included automatically into the project. You can also include this library manually by means of the *“Add Library”* command.

The following sections are intended to explain in detail the meaning of certain parameters (periodicity, offset, etc.), as well as the possibility of switching between different cams:

Periodic cam

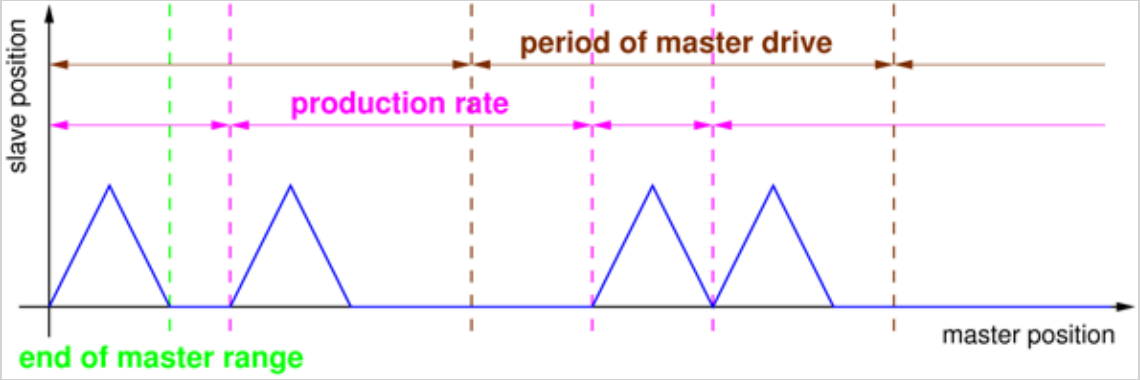
A cam can be run repeatedly when the `Periodic` input of the `MC_CamTableSelect` function block is set to TRUE. Then the cam is restarted automatically after reaching the end position. If this input is FALSE, then the `EndOfProfile` output variable of the `MC_CamIn` function block is set to `TRUE` when the end position of the master is reached. The slave pauses at its current position. Note that the cam activity does not stop after leaving the master value range. When entering the master drive again in the master value range, the slave drive is also checked by the cam.

Behavior in the case of `Slave.EndPosition` <> `Slave.StartPosition`: The function block `MC_CamIn` calculates an internal offset at the end of a period. In the subsequent period, the cam is shifted by this offset so that it continues at the current position of the slave and consequently prevents jumps.

For a periodic cam, you can activate the *“Smooth transition”* option in the cam properties. This is used for preventing jumps from occurring when transitioning from one period to another. Then the slave has the same velocity and acceleration at the end position as at the start position. The period and feed are measured in the units of slave scaling. Even if you do not activate *“Smooth transition”*, the cam can be operated continuously. In this case, your task is to make sure the consistency of the transitions are satisfied to a sufficient degree.

Example

The image below illustrates the differences between the occurring time spans. A practical use case is a conveyor belt that transports identical objects. A tool, such as a punch, is positioned above the conveyor belt and controlled by a slave drive (blue graph). The length of these objects is defined as the value range of the master. The tool runs travels within this range to and from the object.



Of course, the master value range of the cam (this is the range of defined positional values of the master) is not identical to the period of the master drive (in the example: one cycle of the conveyor belt). Therefore, the statement `'SlavePosition = CAM( MasterPosition )'` (the definition of the slave position as a function of the master position using the cam) is valid only for the first run of the cam.

When a new object arrives, a new cam cycle has to be started for controlling the tool. After the objects are placed on the conveyor belt with a specific, variable distance from one another, the production rate (the time span between successive starting of the cam) is not identical to the master value range of the cam.

See also

- *Chapter 1.4.1.8.23.3.1.1.1.5 “Dialog 'Properties - 'Cam'” on page 348*
- *MC_CamTableSelect*

Function block 'MC_CamTable-Select' and 'MC_CamIn'

- `MC_CamTableSelect.MasterAbsolute`:
  If the `MasterAbsolute` input is `TRUE`, then the cam is started at the current master position. This point may be at any position in the master value range of the cam. If the point is outside of the value range of the cam, then an error is issued.
  If the `MasterAbsolute` input is `FALSE`, then the cam is relocated to the current position. The zero point of the master is also shifted to the current master position. This mode is permitted only if the value 0 is in the master value range. Otherwise, an error is issued ("...master leaving specified range...").
- `MC_CamTableSelect.SlaveAbsolute`
  The parameter `CamTableSelect.SlaveAbsolute` influences the `StartMode` of the slave drive. This mode is defined by the `CamIn.StartMode` parameter. The following table documents the `StartMode` that results from the interaction of the two parameters.

- `MC_CamIn.StartMode`:
  - `absolute`: When starting a new cycle, the cam is evaluated independent of the current position of the slave. This can lead to jumps if the slave position to the master start position deviates from that of the master end position.
  - `relative`: The new cam is started allowing for the current slave position. The position that the slave has after the end of the previous cycle is added as a slave offset to the new evaluations of the cam. Jumps can also occur if the slave position at the master start position is not `0`.
  - `ramp_in`, `ramp_in_pos`, `ramp_in_neg`: `ramp_in`: When starting the cam, occurring jumps are prevented by compensating movements. Its dynamics values are limited by `VelocityDiff`, `Acceleration`, and `Deceleration`. If the slave drive is rotary, then the `ramp_in_pos` option compensates in the positive directions only, while `ramp_in_neg` compensates in the negative direction. For linear slave drives, the direction of the compensation is automatic, and `ramp_in_pos` and `ramp_in_neg` are interpreted like `ramp_in`.
- `MC_CamIn.MasterOffset`, `MC_CamIn.MasterScaling`:
  These parameters transform the master position according to the following formula: `X = MasterScaling*MasterPosition + MasterOffset`. The transformed position X is then used for evaluating the cam. In this way, the cam is run at a higher velocity when the value of `MasterScaling` is greater than `1`; on the other hand, the velocity is reduced for values less than `1`.
- `MC_CamIn.SlaveOffset`, `MC_CamIn.SlaveScaling`:
  This input moves or scales the graph of the cam function in the direction of the slave (vertical axis). First the cam is scaled and then moved according to the following formula: `Y = SlaveScaling*CAM( X ) + SlaveOffset`. A `SlaveScaling` > 1 magnifies the slave value range. Accordingly, a `SlaveScaling` < 1 reduces the magnification.

Table 12: Interaction of `MC_CamIn.StartMode` and `CamTableSelect.SlaveAbsolute`

`MC_CamIn.StartMode`	**`MC_CamTableSelect.SlaveAbsolute`**	**`MC_CamIn.StartMode`: New value**
`absolute`	`TRUE`	`absolute`
`absolute`	`FALSE`	`relative`
`relative`	`TRUE`	`relative`
`relative`	`FALSE`	`relative`
`ramp_in`	`TRUE`	`ramp_in absolute`
`ramp_in`	`FALSE`	`ramp_in relative`
`ramp_in_pos`	`TRUE`	`ramp_in_pos absolute`
`ramp_in_pos`	`FALSE`	`ramp_in_pos relative`
`ramp_in_neg`	`TRUE`	`ramp_in_neg absolute`
`ramp_in_neg`	`FALSE`	`ramp_in_neg relative`

See also

- *Chapter 1.4.1.8.23.1.1.1 “Definition of a SoftMotion Cam” on page 317*

Switching Between Cams

Basically, you can switch between different cams at any time. However, you should consider some points:

- In the cam editor, the position of the slave is defined uniquely as the function value of the cam function. This function is defined in the master value range and can be expressed as follows:
  `SlavePosition = CAM( MasterPosition )`
- Because the current position of the master drive usually deviates from the master value range, you must scale the master position in the definition range of the cam function in order to represent a valid argument:
  `SlavePosition = CAM( MasterScale*MasterPosition + MasterOffset )`
- In an analog way, you must scale the function value (the slave position) if the start of the cam in the mode `Absolute` would lead to a jump:
  `SlavePosition = SlaveScale*CAM( MasterPosition ) + SlaveOffset`
- You may have to apply both scaling values, which results in the following:
  `Slaveposition = SlaveScale*CAM( MasterScale*Masterposition + MasterOffset ) + SlaveOffset`
- The appropriate values for scaling and offset parameters can vary from period to period.
- Restarting the `MC_CamIn` function block deletes the corresponding memory area and also the values of scaling and offset. In this way, the cam function is applied in the original definition, which usually results in other values for the slave position. For this reason, it is recommended to restart the `MC_CamIn` function block to start another cam.

Example

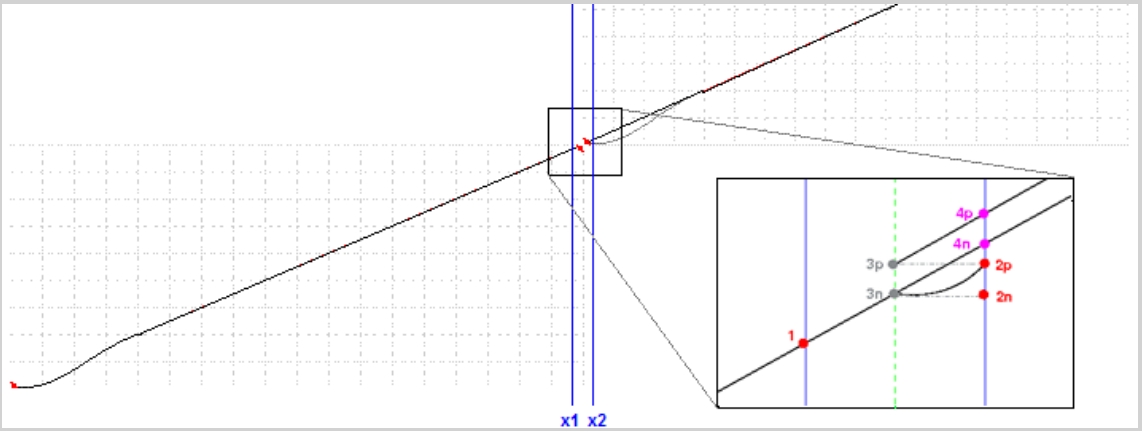
In the following example, it switches from `CAM1` to `CAM2`:

`CAM1` consists of a 5th order polynomial followed by two line segments.

`CAM2` consists of two line segments followed by one 5th order polynomial.

When switching between both cams, you should consider the following:

- To prevent jumps, the values of velocity and acceleration at the end point of the first cam should agree with the values at the start point of the second cam. In the example, this condition is fulfilled because the same velocity (=1) and acceleration (=0) is assigned to the end point of `CAM1` and the start point of `CAM2`.
- You can start the second cam in `Relative` mode when you have defined the start position of the slave as 0. However, the first cam must be running in `non-periodic` mode. Otherwise, if `CAM1` were periodic, then the `Relative` setting would result in a jump.



The magnification shows the transition from `CAM1` to `CAM2`. The blue lines marks the evaluations of the cam functions at the master positions `x1` and `x2`.

Now, we will look at the unfavorable case of `periodic`:

`MasterAbsolute := TRUE; SlaveAbsolute := FALSE;`	
`CAM(x1, CAM1, PERIODIC:=TRUE);`	The call starts an evaluation of the cam at the master position `x1`, which is less than the end position of the master of `CAM1`. Then `CAM1` is evaluated by default and yields point `1` as the position for the slave.
`CAM(x2, CAM1, PERIODIC:=TRUE);`	For the following call of the module, the master position `x2` is outside of the master value range of `CAM1`, whose limit is marked by the green dashed line and agrees with the horizontal axis of the point `3p`. Therefore, the `EndOfProfile` is set. Because `CAM1` was started in `periodic` mode, its restart occurs at the end of the value range, which finally yields the point `2p` as the result of the module call.
`CAM(EXECUTE:=FALSE);`	Switch to the new cam
`CAM(x2, CAM2, PERIODIC:=TRUE);`	Second evaluation at master position `x2`. This time, the new `CAM2` is evaluated. After `CAM2` is started in `Relative` mode, the current slave position (`2p`) is added as offset to the image of the cam function of `CAM2`. This moves the start point of its graph to the point `3p` and its evaluation at the master position `x2` yields the point `4p`, and therefore an unfavorable jump.

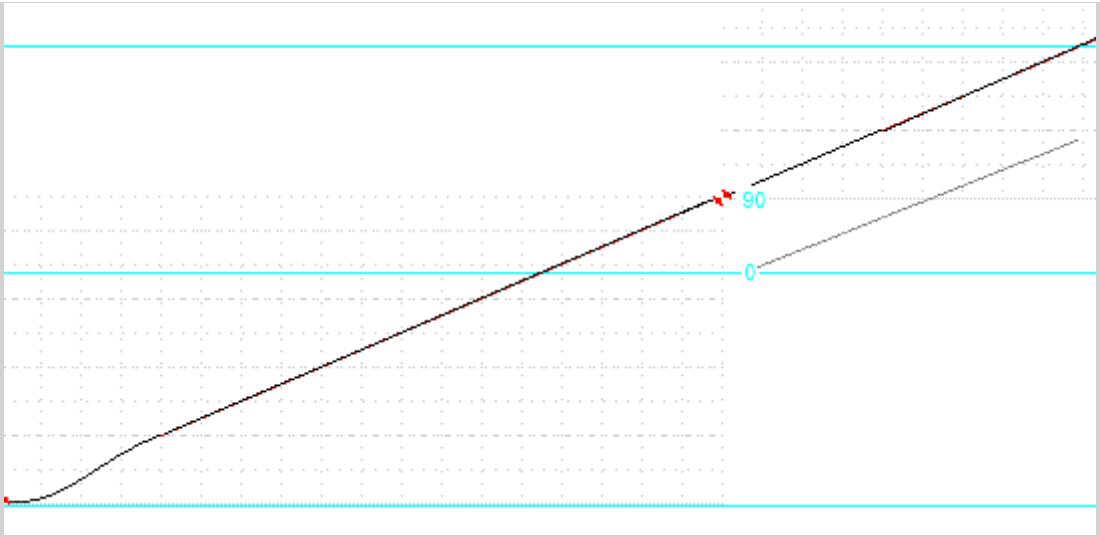
Select the `non-periodic` mode in order to prevent jumps:

`MasterAbsolute := TRUE; SlaveAbsolute := FALSE;`	
`CAM(x1, CAM1, PERIODIC:=FALSE);`	The call starts an evaluation of the cam at the master position `x1`, which is less than the end position of the master of `CAM1`. Then `CAM1` is evaluated by default and yields point 1 as the position for the slave.
`CAM(x2, CAM1, PERIODIC:=FALSE);`	For the following call of the module, the master position `x2` is outside of the master value range of `CAM1`, whose limit is marked by the green dashed line and agrees with the horizontal axis of the point `3n`. Therefore, the `EndOfProfile` is set. Because `CAM1` was started in `non-periodic` mode, slave position (`2n`) assigned to master position `x2` is identical to the position of the slave upon reaching the end of the value range of `CAM1` (`3n`).
`CAM(EXECUTE:=FALSE);`	Switch to new cam.
`CAM(x2, CAM2, PERIODIC:=FALSE);`	Second evaluation at master position `x2`.This time, the new `CAM2` is evaluated. After `CAM2` is started in `Relative` mode, the current slave position (`2n`) is added as offset to the image of the cam function of `CAM2`. This moves the start point of its graph to the point `3n` and its evaluation at the master position `x2` yields the point `4n`, which is on the specific line through the points `1` and `3n`.

To start the cam in `Absolute` mode, you have to make sure that the slave is in an appropriate start position. If the value range of the master agrees with the period of the slave, then switching between cams does not have any complications, regardless of whether the cams are periodic or not.

In the example above, you can start `CAM2` in `Absolute` mode when the periods of the master and slave agree with the master value range of `CAM2` (each is 360°).

If not, for example when the period of the slave is `270°` (indicated by the light blue line), then the `Absolute` option is not permitted without taking additional actions.



In this case, the slave is at 90° when switching from `CAM1` to `CAM2`. Starting `CAM2` in `Absolute` mode causes a jump to 0° (indicated by a gray line).

However, the jump can be prevented by setting the slave offset to the appropriate value of 90°.

Data Structure

Data structures of cams

On project compile, the created cam data is converted internally into a global variable list.

Each cam is represented by the data structure `MC_CAM_REF`. You can access this data structure by means of the IEC program or by preprocessing functions and function blocks. It is available by the `SM3_Basic` library.

A function block that describes a cam can also be generated or populated by the IEC program at runtime.

Example

Definition of the data structure:

```
TYPE mySMC_CAMTable_LREAL_10000_2 :
STRUCT
    Table: ARRAY[0..9999] OF ARRAY[0..1] OF LREAL;
    (* set all scaling definitions to 0 and 1
       result: all values of the table are not scaled *)
    fEditorMasterMin: REAL := 0;
    fEditorMasterMax: REAL := 1;
    fEditorSlaveMin: REAL := 0;
    fEditorSlaveMax: REAL := 1;
    fTableMasterMin: REAL := 0;
    fTableMasterMax: REAL := 1;
    fTableSlaveMin: REAL := 0;
    fTableSlaveMax: REAL := 1;
END_STRUCT
END_TYPE
```

Instantiating the data structure:

```
Cam: MC_CAM_REF;
Cam_PointArray : mySMC_CAMTable_LREAL_10000_2;
```

Calculating the cam:

```
Cam.byType:=2;
Cam.byVarType:=6;
Cam.nTappets:=0;
Cam.strCAMName:='myCAM';
Cam.pce:= ADR(CAM_PointArray);
FOR i:=0 TO 9999 DO
    (* example cam: master 0..360, slave 0..100,
       constant velocity *)
    Cam_PointArray.Table[i][0]:=UDINT_TO_LREAL(I)/10000 *
360;             (* X *)
    Cam_PointArray.Table[i][1]:=UDINT_TO_LREAL(I)/10000 *
100;             (* Y *)
END_FOR
Cam.nElements:=10000
Cam.xStart:=0.0;
Cam.xEnd:=360.0;
```

In order to allow for easy access to the function blocks, they are collected and listed in the `g_CAMManager` global variable with the `Count` property and the `GetCAM(int n)` method.

Example

Access to data objects of the MC_CAM_REF function block:

```
PROGRAM CAMManageRef
VAR
    pCAM_Ref: POINTER TO MC_CAM_REF;
    n: INT;
    i: INT;
END_VAR

n := g_CAMManager.Count;
FOR i:=0 TO n-1 DO
    pCAM_Ref := g_CAMManager.GetCAM(i); (* Processing pCAM_Ref*)
END_FOR
```

See also

- *MC_CAM_REF*

Manually generated cams A cam can be created in an IEC program without using the cam editor.

Example

Declaration:

```
VAR
i: INT;
CAM: MC_CAM_REF := (
    byType:=2, (* not-equidistant *)
    byVarType:=2, (* UINT *)
    nElements:=128,
    xStart:=0,
    xEnd:=360);
Table: SMC_CAMTable_UINT_128_2 := (
    fEditorMasterMin := 0, fEditorMasterMax := 360,
    fTableMasterMin := 0, fTableMasterMax := 6000,
    fEditorSlaveMin := 0, fEditorSlaveMax := 360,
    fTableSlaveMin := 0, fTableSlaveMax := 6000);
END_VAR
```

Implementation:

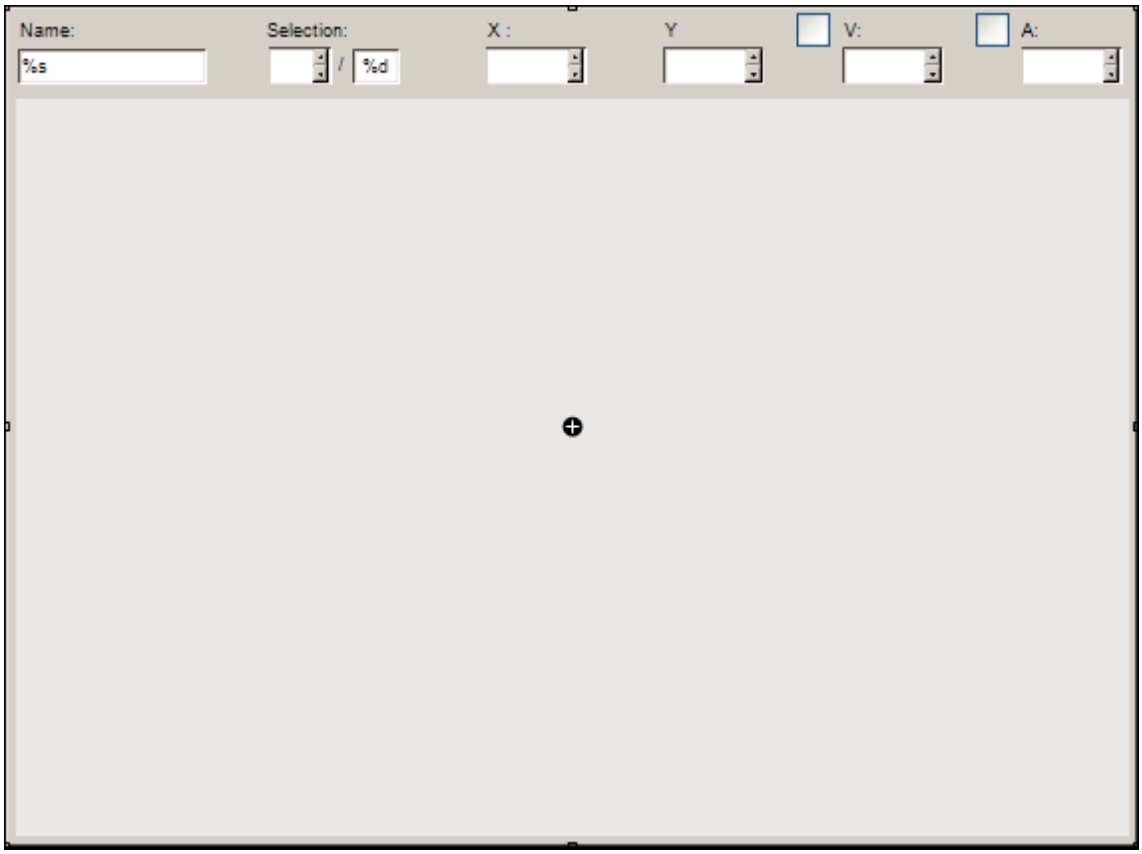
```
(* Generate cam (example: line); unique *)
FOR i:=0 TO 127 DO
    Table.Table[i][0] := Table.Table[i][1] := REAL_TO_UINT(i /
127.0 * 6000);
END_FOR
(* Link pointer; must be done in every cycle *)
CAM.pce := ADR(Table);
```

This generated cam can be specified in the `MC_CamTableSelect` function block and its output used again for `MC_CamIn`.

Visualization Element 'Online cam editor'

The online cam editor is a visualization template that displays a cam table in the visualization. With this element, you can modify the cam in online mode.

The visualization element is made available in a visualization template (*"SMC_VISU_CamEditor"*) of the `SM3_Basic` library. You find it in the visualization editor in the *"ToolBox"* view in the *"SM3_Basic"* tag. For more information about the "Frame" visualization element, refer to the CODESYS Visualization standard help.



In addition to the properties of the frame element, this template contains the following properties:

Property	Description
"xReadOnly"	If TRUE, then the cam cannot be modified.
"dIntervalX"	Step size for the X-value of the SpinControl element
"dIntervalY"	Step size for the Y-value of the SpinControl element
"dIntervalV"	Step size for the V-value (velocity) of the SpinControl element
"dIntervalA"	Step size for the A-value (acceleration) of the SpinControl element
"Editor"	Instance of the `SMC_CamEditor` function block

To visualize the cam, you must declare and call an instance of the `SMC_CamEditor` function block in your application.

```
PROGRAM PLC_PRG
VAR
    myCamEditor: SMC_CamEditor;
END_VAR
```

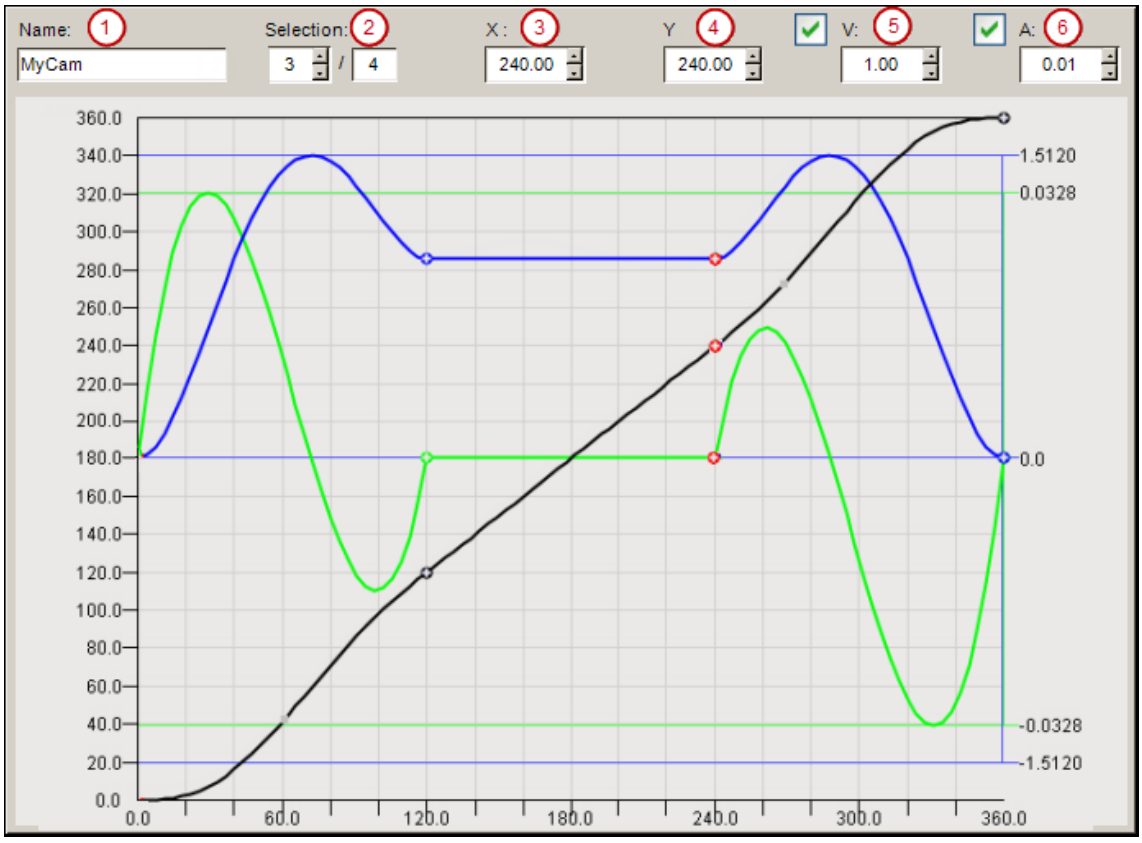
```
myCamEditor(cam := MyCam, bEnable :=TRUE);
```

See also

- *Visualization Element 'Frame'*
- *SMC_CamEditor*

Cam editor in online mode

In online mode, the graphs of position (black), velocity (blue), and acceleration (green) are displayed.



(1) *“Name: ”*	Name of the curve table (read-only)
(3) *“Selection:”*	Selection of a curve point by mean of the SpinControl element. The selected point is displayed red.
(4) *“X:”*	Moves the curve point of the position in the X-direction (horizontal) by means of the SpinControl element or by specifying a value. The first and last points cannot be moved in the X-direction.
(5) *“Y:”*	Moves the curve point of the position in the Y-direction (vertical) by means of the SpinControl element or by specifying a value.
(6) *“V:”*	☑ Moves the curve point of the velocity by means of the SpinControl element or by specifying a value. The display of the velocity curve can be shown and hidden by means of a checkbox.
(7) *“A:”*	☑ Moves the curve point of the acceleration by means of the SpinControl element or by specifying a value. The display of the acceleration curve can be shown and hidden by means of a checkbox.

The cam tappets are displayed as small gray boxes.

It is also possible to move the curve points by dragging and dropping. To do this, the left mouse button must be pressed on the curve point longer than 500 ms. Then the curve point changes to a large red point.

BufferMode

Some function blocks have an input `BufferMode` which is used to control the chronological order of movements. The buffer mode defines whether the function block works in non-buffered mode ("Aborting", standard behavior) or in buffered mode ("Buffered"). The difference between these two modes is the time when they begin their actions:

- "Non-buffered Mode": The movement command is effective immediately, even if this interrupts another movement. The buffer of the commanded movements is deleted.
- "Buffered Mode": The movement command waits until the current function block sets its output `Done` (or `InPosition`, or `InVelocity`, etc.). The buffer modes are also used to define how the velocity curve should look at the transition of the movements.

Table 13: The input "BufferMode" is an ENUM of type MC_BUFFER_MODE.

Aborting	Default mode without buffering. The function block starts immediately and aborts an active movement. The command takes immediate effect on the axis.
Buffered	The function block starts as soon as the last commanded movement is terminated. No blending takes place here. The new movement starts at the velocity that the previous movement has when the end condition is reached (`Done`, `InVelocity`, `InEndVelocity`, `InGear`, `InSync`, `EndOfProfile`, etc.). If the previous movement was `MC_MoveAbsolute` or `MC_MoveRelative`, then the new movement starts at standstill.
BlendingLow	The function block starts as soon as the last commanded movement is terminated. The axis does not stop between movements, but passes through the end position of the first movement at the lower velocity of the two movement commands.
BlendingPrevious	The function block starts as soon as the last commanded movement is terminated. The axis does not stop between movements, but passes through the end position of the first movement at the velocity of the first movement command.
BlendingNext	The function block starts as soon as the last commanded movement is terminated. The axis does not stop between movements, but passes through the end position of the first movement at the velocity of the second movement command.
BlendingHigh	The function block starts as soon as the last commanded movement is terminated. The axis does not stop between movements, but passes through the end position of the first movement at the higher velocity of the two movement commands.

See also

- **linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'**

Supported Function Blocks

All function blocks that can be specified as buffered/blending commands have the following inputs and outputs:

- Input `BufferMode` (type `MC_BUFFER_MODE`)
- Output `Active` (type `BOOL`)

A command is accepted when the function block switches to the state `Busy` after a new movement has been commanded.

Function Block	Can be Defined as a Buffered/Blending Command	Can be Followed by a Buffered/Blending Command	Relevant Signal for Activating the Next Buffered/Blending FB
MC_Power	No	No	
MC_Home	No	No	
MC_Stop	No	No	
MC_Halt	No	No	
MC_MoveAbsolute MC_MoveRelative	Yes	Yes	Done
MC_MoveAdditive	No	Yes (Buffered only)	Done
MC_MoveSuperimposed	No	No (see chapter 'Behavior of MC_MoveSuperimposed')	
MC_MoveVelocity	Yes	Yes (Buffered only)	InVelocity
SMC_MoveContinuousAbsolute SMC_MoveContinuousRelative	No	Yes (Buffered only)	InEndVelocity
MC_PositionProfile MC_VelocityProfile MC_AccelerationProfile	No	Yes (Buffered only)	Done
MC_CamIn	No	Yes, also if periodic (only Buffered)	EndOfProfile
MC_CamOut	No	Yes (Buffered only)	Done
MC_GearIn	Yes (BlendingPrevious only)	Yes (Buffered only)	InGear
MC_GearOut	No	Yes (Buffered only)	Done
MC_GearInPos	Yes (BlendingPrevious only)	Yes (Buffered only)	InSync
SMC_FollowPosition SMC_FollowVelocity SMC_FollowPositionVelocity SMC_FollowSetValues	No	No	
SMC_SetTorque	No	No	
MC_Phasing	No	No	

Function Block	Can be Defined as a Buffered/Blending Command	Can be Followed by a Buffered/Blending Command	Relevant Signal for Activating the Next Buffered/Blending FB
MC_Jog SMC_Inch	No	No These function blocks should not be used when movements are commanded with buffer mode Buffered or Blending*. Jogging and the commanded movements could interrupt each other.	
SMC_BacklashCompensation	No	No	

Note for MC_GearInPos and MC_GearIn: The behavior of other buffer modes as BlendingPrevious is difficult to establish. The main problem is that the velocity of these function blocks can change at any time depending on the master axis. Because blending works best when the blending speed is known as early as possible, only BlendingPrevious is supported.

In the case of BlendingPrevious, the direction of the master axis can also change at any time. This means that the direction that the slave axis should have for MC_GearInPos is known only when the blending is complete. However, we need a direction for the blending movement right when the blending begins. This is why the first movement defines both the blending velocity and the direction, regardless of the direction defined by the subsequent MC_GearIn(Pos).

See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'
- *Chapter 1.4.1.8.23.1.2.8 “Behavior of MC_MoveSuperimposed” on page 340*

Buffering/Blending from Continuous or Synchronized Movement

According to PLCopen, the blending buffer mode determines the velocity at the end of the first movement.

In some cases, the velocity is already entirely determined by the first movement. This is the case when the first movement is of one of the following types:

- Continuous movement (MC_MoveVelocity, SMC_MoveContinuousRelative, or SMC_MoveContinuousAbsolute)
- Synchronized movement (MC_CamIn, MC_GearIn, or MC_GearInPos)

In these cases, Motion Solution supports only the buffer modes Buffered and Aborting. Using one of the blending buffer modes causes an FB error (SMC_BLENDING_NOT_SUPPORTED_BY_PREVIOUS_MOVEMENT).

When the subsequent buffered command becomes active, the output CommandAborted is set to TRUE for a previous movement command. In addition, the "Inxxx" outputs (for example, InVelocity for MC_MoveVelocity or InGear for MC_GearIn) and the output Busy are set for one cycle. This is in contrast to PLCopen, Section 2.4.1, in which CommandAborted and "Inxxx" as well as Busy are mutually exclusive.

See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'

Using One Function Block Instance to Control Multiple Movements

A single function block instance (for example, from `MC_MoveAbsolute`) cannot be used to control multiple buffered/blending movements as long as it is "Busy".

When a function block instance is `Busy`, the command for a new buffered or blended movement with this instance results in the error `SMC_MORE_THAN_ONE_MOVEMENT_PER_INSTANCE`.

To command multiple buffered or blended movements of the same type in a short order, multiple function module instances are required.

See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'

Behavior in Case of Error

If an axis error occurs (for example, the axis switches to the state `errorstop`), the active movement will report an error along with all other accepted movements.

If an FB error occurs in the function block of an active movement, then all movements accepted later also report an error. This is in contrast to PLCopen, Section 2.2.2, in which subsequent commands will continue the execution after an FB error.

See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'

Execution Order of Movement Function Blocks

When buffered movements or blending movements are commanded, the function block instance that commands the subsequent movement must not be executed earlier than the function block instance that commanded the previous movement.

If this order is violated, then the new error `SMC_MOVING_WITHOUT_ACTIVE_MOVEMENT` is reported and the axis switches to the state `Errorstop`.

See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'

Behavior in the Case of Buffered Movements

When a buffered movement is commanded after `MC_MoveAbsolute` or `MC_MoveRelative`, the buffered movement is active in the same cycle where the previous movement reports `Done` and reaches the velocity 0. However, the interpolation of the buffered movement does not start until the next cycle, so that the velocity of the axis at the end of the cycle is equal to 0.

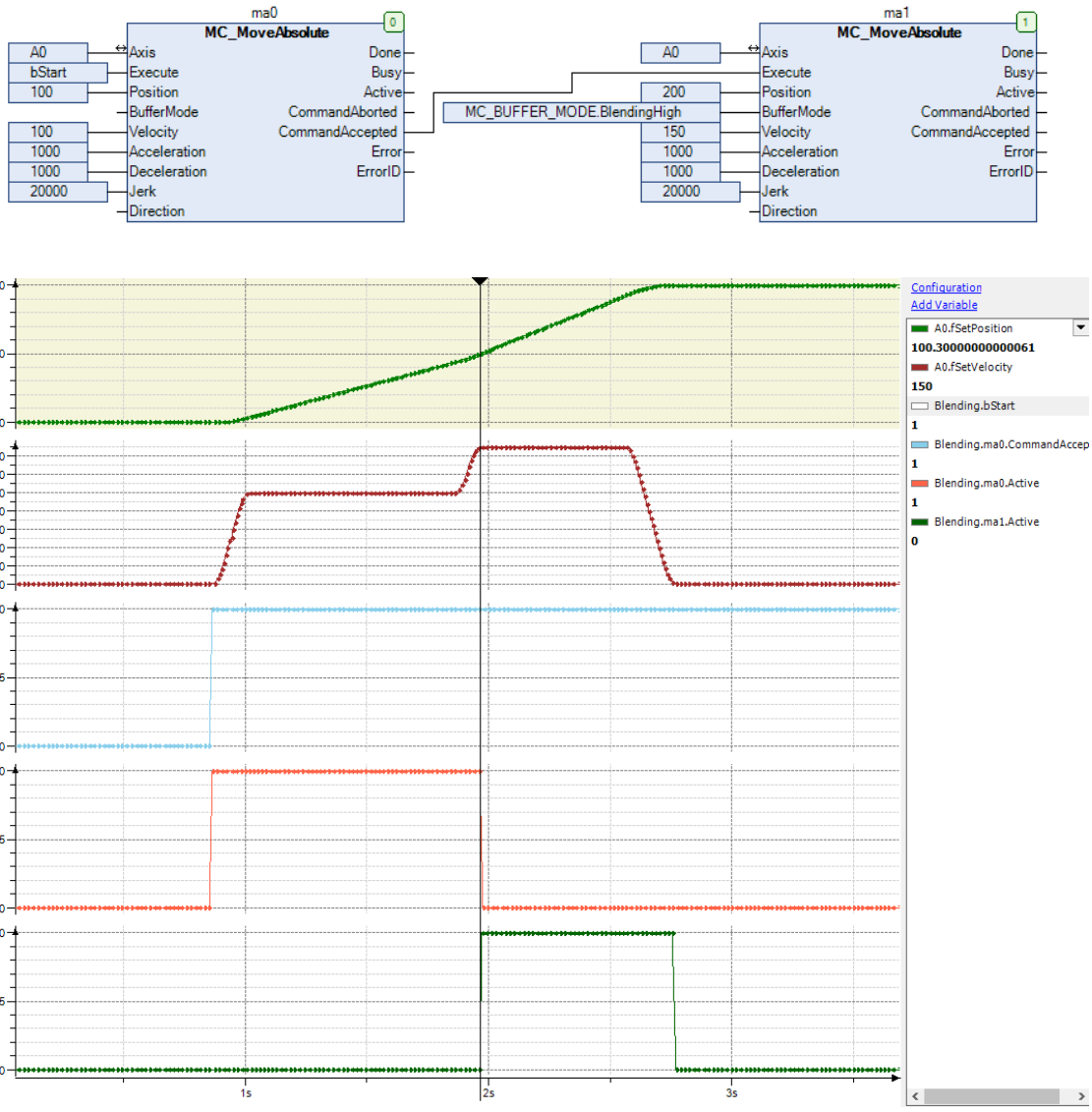
See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'

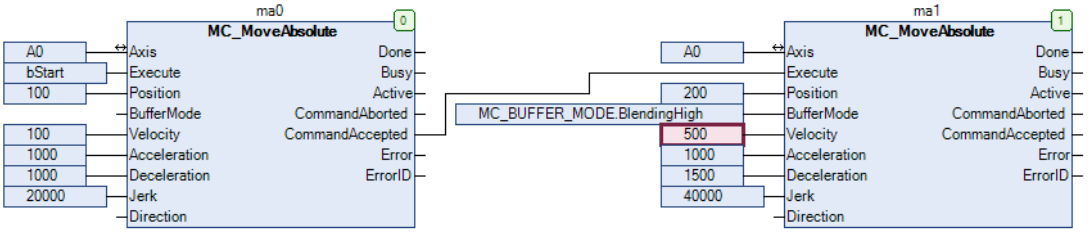
Behavior in the Case of Blending

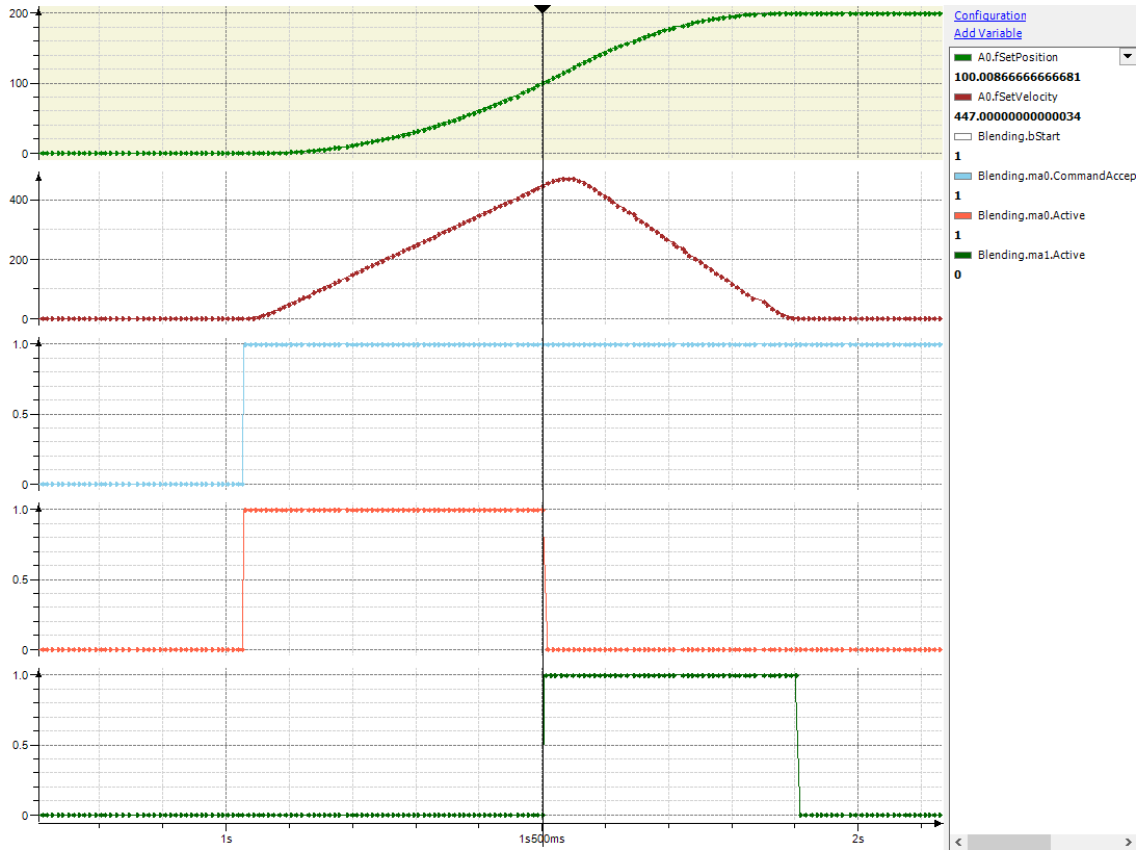
A basic property of the blending behavior of Motion Solution is that the axis moves along the same positions during blending as during a buffered movement. The only difference is the velocity along these positions.

This is obvious for simple cases. See the following example for this:



There are cases in which the property of traversing the same positions by the axis independently of the buffer mode influences the effective blending velocity between the two movements. This is the case, for example, if the above example is modified so that the maximum velocity of the second movement is so high that it cannot be reached at the blending position. According to the rules described in PLCopen, the blending velocity should be 500 u/s. However, to achieve this velocity at position 100 u, the axis would have to reverse, move in the negative direction to a position less than 0 u, and then accelerate to 500 u/s. Instead, in such cases the effective blending velocity is limited to the maximum velocity that can be achieved without reversal and position overshoot. In this example, the maximum velocity is 447 u/s.





The following rules for the effective blending velocity result from the property that the buffer mode does not change the driven positions:

- If the blending velocity cannot be reached without position overshoot, then the effective blending velocity is the next possible velocity that can be reached without overshoot (see example above).

  Note: The effective blending velocity can be higher or lower than the blending velocity.
- If the direction at the beginning of the second movement is opposite to the direction of the first movement, then the effective blending velocity is set to 0. This prevents the position from overshooting in the direction of the first movement beyond its target position.
- If the path of the second movement is too short to allow deceleration from the blending velocity to standstill, then the effective blending velocity is adjusted. It is set to the maximum velocity that allows for safe braking to a standstill on the path of the second movement.
- In the case of modulo axes, the effect of the input `Direction` of `MC_MoveAbsolute` is not affected by blending to a second movement. This means that the target position of the first movement is always in the same modulo period, regardless of whether or not a blending movement follows.
- In the case of modulo axes and a second movement of type `MC_MoveAbsolute`, the blending velocity does not affect the modulo period of the target position of the second movement when `Direction` = `fastest` is used. This means that the same target period is selected regardless of whether the second movement is commanded with `Buffered` or `Blending`.

See also

- **linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'**

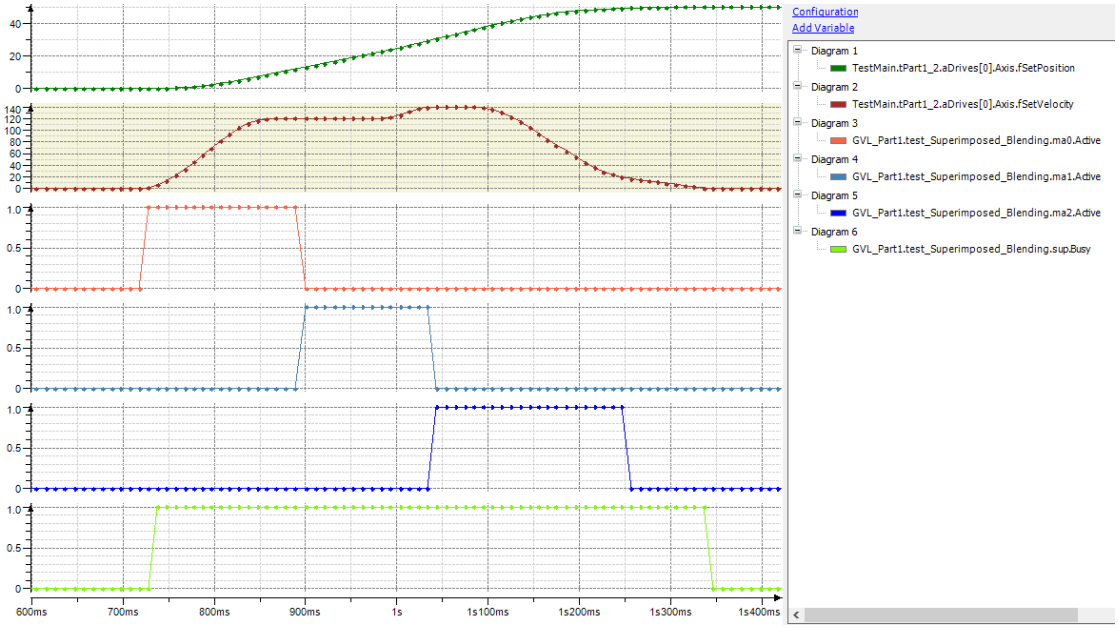
Behavior of MC_MoveSuperimposed

If `MC_MoveSuperimposed` is active and the underlying movement is aborted, then `MC_MoveSuperimposed` is also aborted.

If the underlying movement is not aborted, but rather another movement is commanded with the mode `Buffered` or one of the modes of the blending buffer mode, then the behavior is as follows:`MC_MoveSuperimposed` is not aborted when the blending begins or the new movement is active. Instead, `MC_MoveSuperimposed` is continued in the background until it is done.

If an `MC_MoveAbsolute` assigned with buffered mode or a blending mode is commanded while an `MC_MoveSuperimposed` is active, then the resulting end position depends on the status of `MC_MoveSuperimposed` at the time when the `MC_MoveAbsolute` is active. If `MC_MoveSuperimposed` is still active at this time, then the resulting end position is the sum for the position of `MC_MoveAbsolute` and the path of `MC_MoveSuperimposed`. On the other hand, if `MC_MoveSuperimposed` is no longer active at this time, then the resulting end position is the position of `MC_MoveAbsolute` without the distance of `MC_MoveSuperimposed`. In a similar way, the resulting velocity of `MC_MoveVelocity` depends on the status of `MC_MoveSuperimposed` when `MC_MoveVelocity` is active.

The curve below shows an `MC_MoveSuperimposed` (*"sup"* function block) parallel to three absolute movements with blending buffer mode `BlendingHigh`. The first and second movements are commanded with a velocity of 100 u/s with the function blocks *"ma0"* and *"ma1"*. The function block *"ma2"* commands the third movement with a velocity of 120 u/s. The first target position is 10 u, the second is 25 u, and the third is 40 u. The velocity of the superimposed movement is 20, and the distance is 10. The resulting position is 50 u: the position of the last absolute movement plus the path of `MC_MoveSuperimposed`.



See also

- linktarget [CODESYS_Softmotion] doesn't exist but @y.link.required='true'
- *MC_MoveSuperImposed (FB)*

Examples of Use

Controlling a Cam Drive with a Virtual Time Axis

Refer to the sample project `PLCopenMulti.project` in the installation directory of CODESYS.

This example demonstrates how to implement a periodic cam on a linear drive. The example also demonstrates how to use the tappet function.

1. Insert a cam named `Example` in the device tree below *"Application"*. Open the cam in the editor.
2. Define a tappet in the *"Tappet"* tab.
   - *"X"*: `8.0`
   - *"Positive pass"*: `Invert`
   - *"Negative pass"*: `Invert`
3. Insert a virtual drive named `Drive` in the device tree below *"SoftMotion General Axis Pool"*. For this axis, select the axis type *"Modulo"* with a modulo value of `360`.
4. Insert another virtual drive named `Virtual`. For this axis, select the axis type *"Modulo"* with a modulo value of `10`.
5. Create a *"MOTION_PRG"* program in CFC.

```
PROGRAM MOTION_PRG
VAR
    power1, power2: MC_Power;
    TableSelect: MC_CamTableSelect;
    CamIn: MC_CamIn;
    Tappet: SMC_GetTappetValue;
    MoveVirtual: MC_MoveVelocity;
END_VAR
```

6. Insert a box element and assign the variable `power1` to it. The box element is used for switching on the `Drive`.

   Configure the inputs as follows:

   - *"Axis"*: `Drive`
   - *"Enable"*: `TRUE`
   - *"bRegulatorOn"*: `TRUE`
   - *"bDriveStart"* `TRUE`
7. Insert a box element and assign the variable `power2` to it. The box element is used for switching on the `Virtual` drive.

   Configure the inputs as follows:

   - *"Axis"*: `Virtual`
   - *"Enable"*: `TRUE`
   - *"bRegulatorOn"*: `TRUE`
   - *"bDriveStart"* `TRUE`
8. Insert a box element and assign the variable `MoveVirtual` to it. The box element is used for moving the virtual master.

   Configure the inputs as follows:

   - *"Axis"*: `Virtual`
   - *"Execute"*: `power2.Status`
   - *"Velocity"*: `2`
   - *"Acceleration"* `10`
   - *"Deceleration"* `10`
   - *"Direction"* `positive`

9. Insert a box element and assign the variable `TableSelect` to it. The box element is used for selecting a cam.

   Configure the inputs as follows:

   - *"Master"*: `Virtual`
   - *"Slave"*: `Drive`
   - *"CamTable"*: `Example`
   - *"Execute"* `TRUE`
   - *"Periodic"* `TRUE`
   - *"MasterAbsolute"* `TRUE`
   - *"SlaveAbsolute"* `TRUE`

10. Insert a box element and assign the variable `CamIn` to it. The box element implements the selected cam plate.

    Configure the inputs as follows:

    - *"Master"*: `Virtual`
    - *"Slave"*: `Drive`
    - *"Execute"* `power1.Status`
    - *"MasterOffset"*: `0`
    - *"SlaveOffset"*: `0`
    - *"MasterScaling"*: `1`
    - *"SlaveScaling"*: `1`
    - *"StartMode"*: `absolute`
    - *"CamTableID"*: `TableSelect.CamTableID`
    - *"VelocityDiff"*: `1`
    - *"Acceleration"*: `1`
    - *"Deceleration"*: `1`
    - *"TappetHysteresis"*: `1`

11. Insert a box element and assign the variable `Tappet` to it. The box element checks the setting of the cam switch.

    Configure the inputs as follows:

    - *"Tappets"*: `CamIn.Tappets`
    - *"iID"*: `1`
    - *"bInitValue"* `FALSE`
    - *"bSetInitValueAtReset"*: `FALSE`

    ⇨ The tappet is defined as an inverting tappet. For this reason, its value is changed every 10 seconds.

12. The sample project provides a visualization for checking the individual function blocks and the position of the axes.

13. Add the call of the `MOTION_PRG` program to the task *"MainTask"*.

14. Load the project to the controller and start it.

See also

- linktarget [_sm_edt_drive_general] doesn't exist but @y.link.required='true'
- *MC_Power*
- *MC_CamTableSelect*
- *MC_CamIn*
- *SMC_GetTappetValue*
- *MC_MoveVelocity*

Alternating Cams

Refer to the sample project `PLCopenMultiCAM.project` in the installation directory of CODESYS.

This example demonstrates how a cam movement can be created with two alternating cams. The program is implemented in ST and executes the same actions as the sample "Cam Drive Control using a Virtual Time Axis". At the end of the first cam, the `MC_CamIn` function block sets the `EndOfProfile` output. In this way, the other curve table is assigned to `MC_CamTableSelect` and `MC_CamIn` is restarted.

Reference

User Interface

Objects

Object 'Cam Table'

Tab 'Cam'

In this graphical editor, the cam graphs are defined. You can switch between the graphical editor and the alternative tabular editor at any time (*"Cam table tab"* tab).

The editor window displays the curves of four graphs:

- Slave position (black)
- Slave velocity (blue)
- Slave acceleration (green)
- Slave jerk (yellow)

The horizontal axis of all four coordinate systems shows the range of the master values ([0,360]). The vertical axis in the position diagram shows the value range that is defined in the cam properties. The vertical axis of velocity, acceleration, and jerk is scaled automatically.

A new inserted cam is assigned with default values. It consists of four points that subdivide the graph into three sections: [0,120], [120,240], and [240,360]. Each of the interval parts of the cam graphs is type Poly5 (5th degree polynomial).

You can modify all curves, except the jerk curve. As velocity, acceleration, and jerk are derived curves, changes to one graph causes changes to the other graphs.

You change the height of the diagram by moving the horizontal separation bars.

Table 14: “View 'ToolBox'”

: *“Select”*	Select a line in the table by using this tool. Selected points are deleted by pressing the *[Del]* key.
“Add point”	Add new points with this tool. Click the insertion point in the diagram. The graph is then adapted automatically so that its curve runs through the new inserted point.

Table 15: “View 'Properties'”

X	X-position of the slave axis
Y	Y-position of the slave axis
V	Velocity of the slave axis
A	Acceleration of the slave axis
J	Jerk of the slave axis

See also

- *Chapter 1.4.1.8.23.3.1.1.1.5 “Dialog 'Properties - 'Cam'” on page 348*
- *Chapter 1.4.1.8.23.1.1.3 “Creating Cams” on page 319*

Tab 'Cam table'

The cam table is an alternative to the graphical editor for defining the cam graphs (*“Cam”* tab). You can switch between the table editor and the graphical editor at any time.

The first line of the table always contains the start position of the master (and the related slave values) and the last line is always the end position. The lines in-between alternately define segments and points.

Table 16

	Inserts a new line.
	Deletes the selected segment
“X”	X-position of the slave axis
“Y”	Y-position of the slave axis
“V”	Velocity of the slave axis
“A”	Acceleration of the slave axis
“J”	Jerk of the slave axis
“Segment type”	• *“Poly5”*: 5th degree polynomial • *“Line”* • Linear
The following values result from the values of the respective segment. They cannot be modified.	
min(Position)	Minimum value of the slave position
max(Position)	Maximum value of the slave position
max(Velocity)	Maximum value of the velocity of the slave, based on the master axis
max(Acceleration)	Maximum value of the acceleration of the slave, based on the master axis

Table 17: “View 'ToolBox'”

: *“Select”*	Select a line in the table by using this tool. Selected points are deleted by pressing the *[Del]* key.

See also

- *Chapter 1.4.1.8.23.3.1.1.1.5 “Dialog 'Properties - 'Cam'” on page 348*
- *Chapter 1.4.1.8.23.1.1.3 “Creating Cams” on page 319*

Tab 'Tappets'

The tappet paths are defined in this table graphical editor. A tappet path defines one or more tappets depending on the master position. At the upper edge of the editor window, a horizontal axis approaches the range of the master positions. The individual tappet paths are defined below.

You can switch between the graphical editor and the alternative tabular editor at any time (*“Tappet table”* tab).

Table 18

1	Track ID of the tappet path All tappets of a tappet path refer to the same tappet switch (a variable of type BOOL).

Table 19: “View 'ToolBox'”

: *“Select”*	Select the tappets by means of this tool. You can drag the selected tappets to another position. You can modify the switch on/off attribute of a tappet by clicking the relevant end of the crossed line ( ). Delete the selected tappet by pressing the *[Del]* key.
	Add new tappets with this tool. Click the insertion point in the path.

Table 20: “View 'Properties'”

the tappet is assigned to a result, if it is passed from the position of the master axis in the positive (increasing master values) or negative direction.	
“X”	Position of the tappet
“Positive pass”	Switch on/off attribute ● No action ● Switch to ON ● Switch to OFF ● Invert
“Negative pass”	Switch on/off attribute ● No action ● Switch to ON ● Switch to OFF ● Invert

Table 21: Table of the possible combinations of tappet attributes

Tappet symbol	Positive pass	Negative pass
	No action	No action
	Switch to ON	No action
	Switch to OFF	No action
	No action	Switch to ON
	No action	Switch to OFF
	Switch to ON	Switch to OFF
	Switch to ON	Switch to OFF
	Switch to OFF	Switch to ON
	Switch to OFF	Switch to OFF
	Invert	No action
	No action	Invert
	Switch to ON	Invert
	Invert	Switch to ON
	Invert	Switch to OFF
	Switch to OFF	Invert
	Invert	Invert

See also

- *Chapter 1.4.1.8.23.1.1.5 “Defining Switch Points” on page 322*

Tab 'Tappet table'

This tabular editor is an alternative to the graphical editor for configuring the tappet paths (*“Tappets”* tab). A tappet path defines one or more tappets depending on the master position. In the table, the lines with the definitions of the associated tappets follow below each line that defines a tappet path.

You can switch between the table editor and the graphical editor at any time.

Table 22

	Inserts a new tappet.
	Deletes the tappet.
“Track ID”	ID of the tappet path All tappets of a tappet path refer to the same tappet switch (a variable of type BOOL).
“X”	Position of the tappet

“Positive pass”	Switch on/off attribute ● No action ● Switch to ON ● Switch to OFF ● Invert
“Negative pass”	Switch on/off attribute ● No action ● Switch to ON ● Switch to OFF ● Invert

Table 23: “View 'Properties'”

The tappet is assigned to a result, if it is passed from the position of the master axis in the positive (increasing master values) or negative direction.	
“X”	Position of the tappet
“Positive pass”	Switch on/off attribute ● No action ● Switch to ON ● Switch to OFF ● Invert
“Negative pass”	Switch on/off attribute ● No action ● Switch to ON ● Switch to OFF ● Invert

See also

- *Chapter 1.4.1.8.23.1.1.5 “Defining Switch Points” on page 322*

Dialog 'Properties - 'Cam'

Function: Use this dialog to define the global variables of the cam.

Table 24: “Dimensions”

“Master start/end position”	The start and end positions of the master define the range of the master values and therefore the scale of the horizontal axis of the cam. The default settings are given in angular degrees with 0 and 360 as limiting values.
“Slave start/end position”	The associated slave positions are determined by the graph type that is defined for the cam. However, the segment depicted by the curves (this is also the scale of the vertical axis) can be defined by the start and end positions of the slave that are given here.

Table 25: “Period”

These settings affect the work in the cam editor and cam table. Depending on these parameters, the slave start point is adjusted automatically when the end point is changed, as well as the other way around. This adjustment optimizes the period transition to be as smooth and jerk-free as possible.	
“Smooth transition”	☑: The values for position, velocity, and acceleration are adjusted automatically.
“Slave period”	Indicates when the slave period is repeated mechanically. Then the slave position at the start and end of the master period can deviate by one integer multiple of this value. This value is effective only if the *“Smooth transition”* check box is selected.

Table 26: “Continuity requirements”

Activation of these options for the continuity of the curve does not have any effect when editing the cam. It does, however, prompt a continuity check, which reports any violations to the message view (CAM). It is not possible to edit jumps in the position curve. The default setting also requires the continuity of velocity and acceleration. You can clear these options, for example in the special case of a curve that consists of only linear segments. However, this can lead to breaks in the position curve. By default, the jerk (4th derivative) is not tested for jumps.	
“Position”	☑: The entire curve is tested for jumps.
“Velocity”	
“Acceleration”	
“Jerk”	

Table 27: “Compile format”

When compiling, `MC_CAM_REF` structure variables are generated. A cam is described according to the following options:	
“Polynomial (XYVA)”	Polynomial description of the individual points, consisting of master position, slave position, slave velocity, and slave acceleration.
“One-dimensional point array”	1D table of slave positions
“Two-dimensional point array”	2D table of composite master/slave positions
“Elements”	Number of elements in the arrays. This array has already been created in SM3_Basic for the standard cases *“128”* and *“256”*. If you type in another value, you must create the structure in your application (see the following example).

Example of an array with 720 elements

```
TYPE SMC_CAMTable_LREAL_720_2 :
STRUCT
    Table: ARRAY[0..719] OF ARRAY[0..1] OF LREAL;
    fEditorMasterMin, fEditorMasterMax: REAL;
    fEditorSlaveMin, fEditorSlaveMax: REAL;
    fTableMasterMin, fTableMasterMax: REAL;
    fTableSlaveMin, fTableSlaveMax: REAL;
END_STRUCT
END_TYPE
```

See also

- *Chapter 1.4.1.8.23.1.1.3 “Creating Cams” on page 319*

Commands

Cam

Command 'Display generated code'

Function: This command opens the *"Generated code"* dialog where the IEC initialization code of the represented cam is displayed.

Call: Menu bar: *"Cam"*.

Requirement: The cam editor is open and displays a cam.

Dialog 'Generated code'

Example: IEC initialization code

```
{attribute 'linkalways'}

VAR_GLOBAL
Cam_A: ARRAY[0..3] OF SMC_CAMXYVA := [
    (dX := 0, dY := 0, dV := 0, dA := 0),
    (dX := 120, dY := 120, dV := 1, dA := 0),
    (dX := 240, dY := 240, dV := 1, dA := 0),
    (dX := 360, dY := 360, dV := 0, dA := 0)];
Cam: MC_CAM_REF := (nElements := 4, byType := 3, xStart := 0,
xEnd := 360, nTappets := 0, strCAMName := 'Cam', pce := ADR(Cam_A),
xPartofLM := TRUE);
END_VAR
```

Command 'Read cam data from ASCII table'

Function: This command reads an `ASCII` file.

Call: Menu bar: *"Cam"*.

Requirement: The cam editor is open.

When being read, the file data is interpreted as the xy-values of a cam. The *"Number of points"* opens so that you can decrease the number of interpolation points. Then the determined points are interpolated to a cam and displayed in the editor.

The *"Write cam data to ASCII table"* commands creates an appropriate `TXT` file.

See also

- *Chapter 1.4.1.8.23.1.1.3 "Creating Cams" on page 319*
- *Chapter 1.4.1.8.23.3.1.2.1.4 "Command 'Write cam data to ASCII table'" on page 351*

Dialog Box 'Number of points'

"Number of points"	Number of points used for interpolation. Preset: According to the number of xy-values that are stored in the read file. Example: `256` You can decrease the preset value in order to determine the cam with fewer interpolation points. When determining the interpolation points, their x-values are distributed equidistantly.

As the cam is interpolated using a 5th degree polynomial, a larger number of interpolation points may cause oscillations.

Command 'Read cam online file'

Function: This command reads an external file with cam data. The file extension is `CAM`. The cam is displayed in the cam editor.

Call: Menu bar: *"Cam"*.

Requirement: The cam editor is open.

The *"Write cam online file"* command creates an appropriate file in `CAM` format.

See also

- *Chapter 1.4.1.8.23.3.1.2.1.5 "Command 'Write cam online file'" on page 352*

Chapter 1.4.1.8.23.1.1.1 "Definition of a SoftMotion Cam" on page 317

Command 'Write cam data to ASCII table'

Function: This command creates an `ASCII` file (`TXT` extension) on the development system. A specified number of xy-values of the active cam is saved in this file. A standard dialog box opens first and then the *"Number of points"* dialog box.

Call: Menu bar: *"Cam"*.

Requirement: The cam editor is open and displays a cam.

See also

- *Chapter 1.4.1.8.23.3.1.2.1.2 "Command 'Read cam data from ASCII table'" on page 350*
- *Chapter 1.4.1.8.23.1.1.3 "Creating Cams" on page 319*

Dialog Box 'Number of points'

The ASCII table does not contain any information about cams.

"Number of points"	Number of xy-values that are saved in the file and represented in the curve shape. For this purpose, the x-curve is split equidistantly and the respective y-value is determined.

Command 'Write cam online file'

Function: This command creates a file (`CAM` extension). The file contains the data of the cam that is active in the editor.

Call: Menu bar: *"Cam"*.

Requirement: The cam editor is open and displays a cam.

The cam data is composed of a cam description and the positions and types of cams.

A `CAM` file can be read to the editor by means of the *"Read cam online file"* command.

In addition, an instance of the `SMC_ReadCAM` function block can read the file in order to load a cam table into the application at runtime.

See also

- *Chapter 1.4.1.8.23.3.1.2.1.3 " Command 'Read cam online file'" on page 351*
- *Chapter 1.4.1.8.23.1.1.8 "Data Structure" on page 330*
- *Chapter 1.4.1.8.23.1.1.3 "Creating Cams" on page 319*

1.4.1.9 Working with Controller Networks

With the following functionalities, CODESYS supports communication between controllers (PLC) and the insertion of a safety controller below a PLC:

- **Symbol Configuration**: CODESYS creates symbols with certain access rights for the variables in an application. With these symbols, you can access the variables from outside, for example from an OPC server.
- **Data Source Manager**: Manages the connection settings and the data transmission to remote devices (data sources). The transmitted data is mapped in data source variables that are accessed in the visualization or local application. An example of this is a control panel that controls remote devices and displays the state of the device as an HMI application.
- **Network Variables**: Network variables are variables whose values are accessible to different controllers in the network. The variables have to be defined in rigid, identical lists in both the transmitter device and the receiver device. These lists are assigned to applications, but can be located in different projects.
- A **safety controller** can be inserted below a PLC in the device tree. The communication links of the safety controller to the field devices, controller networks, and development system are routed through this controller.

The "DataServer" object is obsolete.

The data link with CODESYS DataServer has already been superseded with SP10 by a data link with data sources. With CODESYS 3.5 SP17, the functionality has now been completely removed.

In case you want to adapt an existing project with a "DataServer" object, you can do the following: Open the existing project with CODESYS V3.5 SP16, select the data server object, and click "Convert Data Server to Data Source Manager" in the context menu. After the conversion of the data link to a data source connection, you can open the project with a current CODESYS version.

See also

- *Chapter 1.4.1.9.2 "Symbol Configuration" on page 357*
- *Chapter 1.4.1.9.4 "Data Link with Data Sources" on page 363*
- *Chapter 1.4.1.9.3 "Network Variables" on page 360*
- *Chapter 1.4.1.9.1.1 "Network topology" on page 353*
- *Chapter 1.4.1.9.1.2 "Addressing and Routing" on page 353*
- *Chapter 1.4.1.9.1.3 "Address Structures" on page 355*
- *Chapter 1.4.1.9.5 "Subordinate safety controller" on page 378*

1.4.1.9.1 Network and Addressing

Constructing a control network hierarchically, so that extensive self-configuration is possible.

In CODESYS the network topology is mapped to clear addresses and the routing algorithm is kept simple by structured addresses. There is direct and relative addressing and automatic address determination during the bootup of the system.

See also

- *Chapter 1.4.1.9.1.1 “Network topology” on page 353*
- *Chapter 1.4.1.9.1.2 “Addressing and Routing” on page 353*
- *Chapter 1.4.1.9.1.3 “Address Structures” on page 355*

Network topology

Information and recommendations for the topology of a control network

It is recommended to set up a network system so that the following are possible:

- Extensive self-configuration (address assignment)
- Transparent support for every communication medium
- Transport of data packets between different networks

The routing mechanism should be so simple that each network node can reroute data packets, even if it has a low memory capacity. Therefore, avoid extensive routing tables, complex calculations or queries at runtime.

Construct the control network hierarchically. Each node may possess a parent node and any number of child nodes. A node without a parent is a "top level" node. Cycles are not permitted, i.e. each control network has the structure of a tree.

Parent-child relationships results from the specification of certain network areas. A network area can be, for example, a local Ethernet or a serial point-to-point connection. We differentiate between the main network (mainnet) and the subnetworks (subnet). Each node belongs at the most to one main network, to which its parent node, if one exists, also belongs. For each node any desired number of subnets can be configured, for which the node acts in each case as a parent.

A network area may have only one parent node. Therefore, a configuration in which a network area is defined at the same time as a subnet of several nodes is invalid.

See also

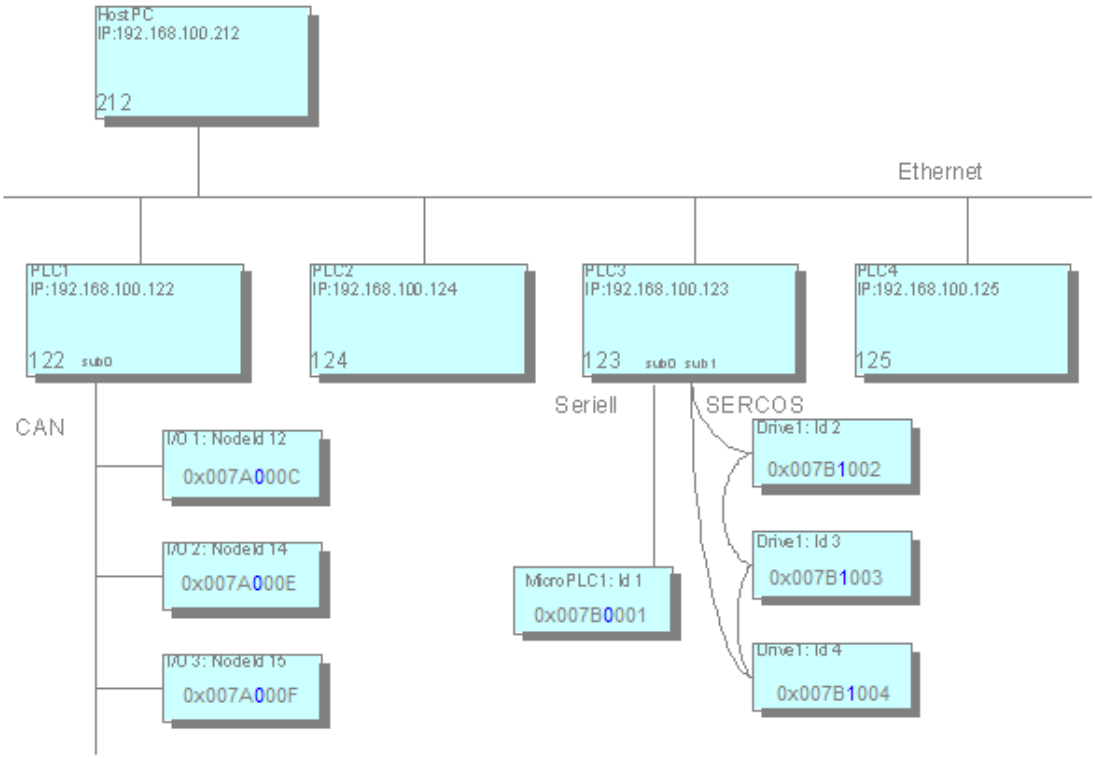
- *Chapter 1.4.1.9.1.2 “Addressing and Routing” on page 353*
- *Chapter 1.4.1.9.1.3 “Address Structures” on page 355*

Addressing and Routing

Addressing means: the topology of the control network is mapped to unique addresses.

A node address is composed hierarchically: for each network connection the associated block driver determines a local address, which uniquely identifies the node within the local network. The complete node address is formed as follows: The local address is placed in front of the subnet index of the local network assigned by the parent. In turn, the subnet index is placed in front of the node address of the parent. The length of the subnet index (in bits) is thereby determined by the device. The length of the local address, conversely, is determined by the type of network. A node without a main network is a top level node with address 0. A node with a main network that contains no parent is likewise a top level node. It is given the local address of the main network.

See an example of a control network here:



In the example the addresses of the nodes are represented in hexadecimal notation. The first 4 digits represent the address of the respective parent in the main network, for example `0x007A=122` for `PLC1`. The next byte (in blue lettering) is reserved for the subnet index and is followed by the local address, for example `C=12` for node ID `12`. The structuring of the addresses makes a lean routing algorithm possible. Routing tables, for example, are thus unnecessary. Information is queried only locally: via its own address and via the address of the parent node. On this basis a node can correctly process the data packets:

- If the destination address corresponds to the address of the current node, then this is meant to be the receiver.
- If the destination address starts with the address of the current node, then the data packet is either meant directly for a child or for a descendant of the node and must be forwarded.
- In all other cases the receiver is not a descendant of the current node and the data packet must be forwarded to its own parent.

Relative addressing is a special case: relative addresses do not contain the node number of the receiver, but directly describe the path from the sender to the receiver. The principle is similar to the relative path in the file system: the address consists of the number of steps via which the packet must be transported upwards. These are the steps to the corresponding parent and from the subsequent path downwards to the destination node.

The advantage of relative addressing is that two nodes in the same subtree can continue to communicate if the complete subtree is shifted to another place in the entire network. Whereas the absolute node addressing has to be modified due to this shift, the relative addressing is still valid.

Address determination

For a node to know its own address it must either know the address of its parent node or know that it is a top level node. For this purpose the node dispatches a message during the bootup to all network devices for address determination. As long as it receives no response to this message, the node considers itself to be a top level node, but continues to search for a possible parent. A parent node responds by announcing its address. The node will thus independently complete its address and will announce it to the subnets. An address determination can be accomplished during the bootup or at the request of the PC used for programming.

See also

- *Chapter 1.4.1.9.1.1 “Network topology” on page 353*
- *Chapter 1.4.1.9.1.3 “Address Structures” on page 355*

Address Structures

Network addresses

Network addresses represent a mapping of the addresses of the network type (for example IP) to logical addresses within a control network. This mapping is carried out by the corresponding block driver. The first three bytes of the IP address are identical for all network devices within an Ethernet network with "Class C" IP addresses. Consequently, the last 8 bits of the IP address suffice as network address, since they enable unambiguous mapping between the two addresses on the block driver.

A node has a different network address for each network connection. Different network connections can have the same network address, since each address need only be locally unique.

Terminology: the network address in the main network is usually designated as the network address of a node with no specification of the network connection.

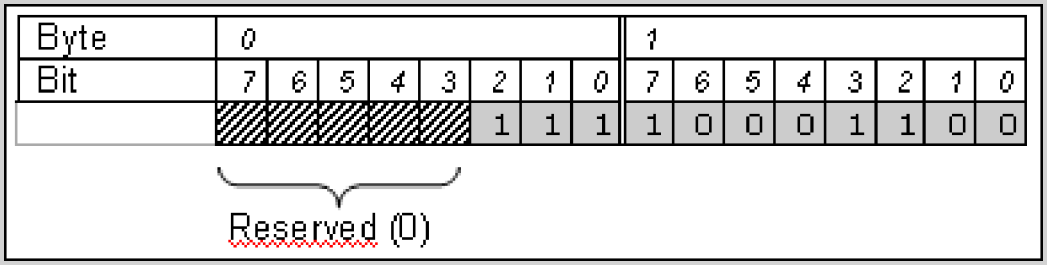
The length of a network address is specified in bits and can be chosen by the block driver as required. The same length must be used for all nodes within a network area. A network address is represented by an array of bytes in accordance with the following coding:

- Length of the network address: n bits
- Necessary bytes: b = (n + 7) DIV 8
- The (n MOD 8) bits of lowest rank of the first byte and all others (n DIV 8) are used for the network address.

Example of network address coding

Length: 11 bit

Address: 111 1000 1100



Node addresses

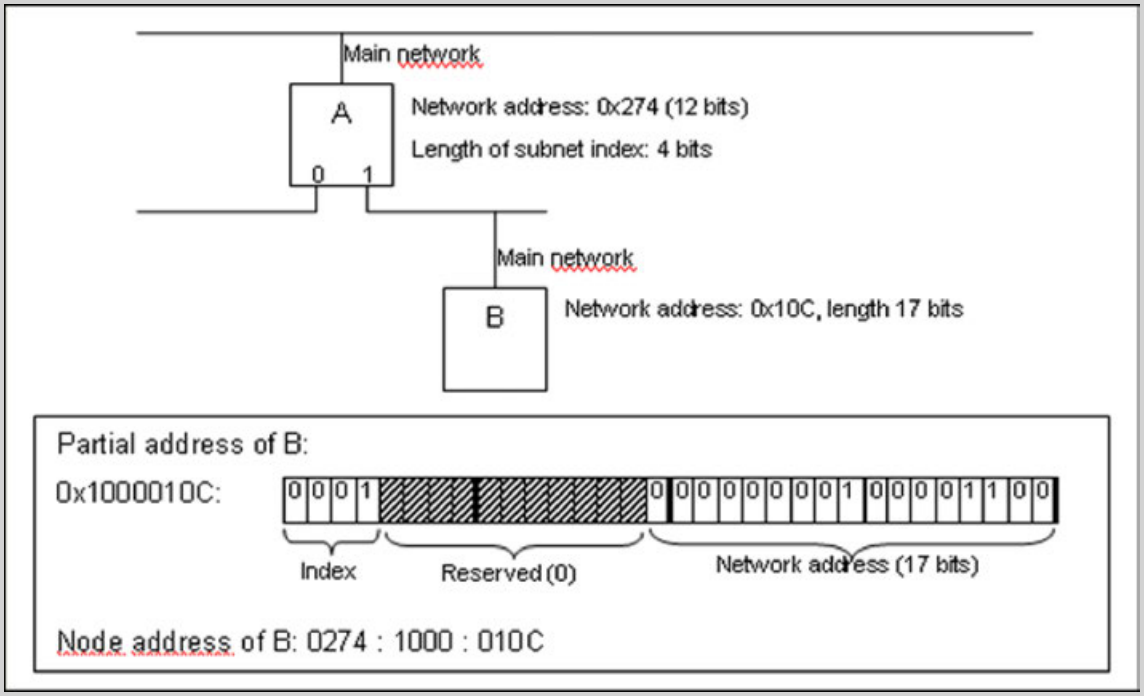
The node address indicates the absolute address of a node within a control network and is therefore unique within the whole "network tree". The address is composed of up to 15 address components, each of which occupies 2 bytes. The lower a node is located within the network hierarchy, the longer its address.

The complete node address consists of the partial addresses of all preceding nodes and the partial address of the node itself. Each partial address consists of one or more address components. The length is therefore always a multiple of 2 bytes. The partial address of a node is formed from the network address of the node in its main network and the subnet index of the main network in the case of the parent node. The bits required for the subindex are determined by the router of the parent node. Filler bits can be inserted between the subnet index and the network address in order to ensure that the length of the partial address is a multiple of 2 bytes.

Special cases:

- A node without a main network: this means that there is neither a subnet index nor a network address in the main network. In this case the address is set to 0x0000.
- A node in the main network without a parent: In this case a subnet index with the length 0 is assumed. The partial address corresponds to the network address, if necessary extended by filler bits.

Example of node addresses



The node address is always specified in hexadecimal. The individual address components (two bytes in each case) are separated by a colon ":". Since this represents an array of bytes and not a 16-bit value, the components are not displayed in the Intel format. For manually input addresses, missing parts in an address component are supplemented by leading zeros: "`274`" = "`0274`". In order to improve the legibility, the display should also always contain the leading zeros.

Absolute and relative addresses

Communication between two nodes can be based on relative or absolute addresses. Absolute addresses are identical to node addresses. Relative addresses specify a path from the sender to the receiver. They consist of an address offset and the descending path to the receiver.

The (negative) address offset describes the number of address components by which a packet must be passed upwards in the tree before it can be passed back down by the common parent node. Since nodes can use partial addresses that consist of more than one component, the number of parent components to be passed is always equal to the address offset. This means that the demarcation between the parent nodes is no longer clear. For that reason the common start of the address of the communication partners is used as the parent address. Each address component is counted as an upward step, independent of the current parent node. Each error resulting from this assumption can be detected by the corresponding parent node and must be handled by it accordingly.

After achieving the common parent node the relative path, as an array of address components, is followed downwards as usual. Formal: the node address of the receiver is formed by removing the last address offset components from the node address of the sender and by appending the relative path to the remaining address.

Example of the formation of node addresses

In the following example each address component is represented by a letter; in each case a dot separates nodes from each other. Since a node can carry several address components, there are some in the example that are represented with several letters.

Node A: `a.bc.d.ef.g`

Node B: `a.bc.i.j.kl.m`

- Address of the lowest common parent: `a.bc`
- Relative address from A to B: `-4/i.j.kl.m` (The number 4 results from the 4 components, `d`, `e`, `f` and `g`, which must pass on the data packet in the upward direction)

In order to guarantee correct operation of the routing, the relative address must be adapted each time it passes an intermediate node. It is sufficient to adapt the address offset. This is always done by the parent node: If a node receives a data packet from one of its subnets, the address offset is increased by the length of the address component of this subnet.

- If the new address offset is < 0, then the data packet must be passed further upward.
- If the address offset is >= 0, then the data packet must be passed on to the child node whose local address corresponds to the relative path, starting from the address offset. First of all, however, the address offset must be increased by the length of the local address of the child node, so that the child node sees the correct address.

A special situation results if the error mentioned above occurs during the determination of the common parent node. In this case the address offset of the actual parent node is negative, but this value is larger than the length of the partial address of the subnet from which the packet originated. So that the next node sees a correct relative address in this case, the node concerned must do the following: it must discover the error, calculate the local address of the child node on the basis of the address of the predecessor node and the length difference, and adapt the address offset accordingly. In this case, too, the address components as such remain unchanged; only the offset is changed.

Broadcast addresses

There are two types of broadcast - global and local. A global broadcast is sent to all the nodes in a network. The empty node address with a length of 0 is reserved for this purpose.

Local broadcasts are sent to all the devices in a network area. For this purpose, all the bits of the network address are set to 1. This is permissible both in relative and in absolute addresses.

A block driver must be able to process both kinds of broadcast addresses. This means: empty network addresses as well as network addresses whose bits were all set to 1 must be interpreted and sent to all devices concerned.

1.4.1.9.2 Symbol Configuration

Use the symbol configuration for preparing symbols with specific access rights for project variables. With these symbols, you can access the variables from outside, for example from an OPC server. When generating code, CODESYS also generates a symbol file (*.xml) that includes the description of the symbols.

The symbol file is stored in the project directory. The name of the symbol file is composed as follows: `<project name>.<device name>.<application name>.xml`

Example `proj_xy.PLC1.application.xml`

You can also generate the symbol file with the “Generate Code” command. This is very useful when downloading to the PLC is not possible.

The variables that you export as symbols can be bundled in the symbol configuration editor or defined in the variables declaration using the `{attribute 'symbol'}` pragma. Another option is using the element in the SFC editor, where you can define the implicitly generated element variables that should be exported to the symbol configuration.

The name of the symbol is generated in the symbol configuration in the following syntax: `<application name>.<POU name>.<variable name>`. When accessing the variable, you must always provide the complete symbol name in this syntax.

Example `MyApplication.PLC_PRG.a` or `MyApplication.SymFB.a`

As a rule, read-only access applies to symbols for input addresses and for variables that are mapped to input channels. Write access is possible for testing purposes in simulation mode only.

The symbol file is downloaded together with the application to the PLC. Depending on the device description, this file can be generated as an additional (child) application. This application is then listed on the *"Application"* tab of the device editor. Syntax: `<application name>._symbols`. The symbol application is regarded as a "normal" application with respect to the maximum number of applications on the PLC.

If your controller has a user management, then you can assign different access rights to a symbol to the individual user groups (clients). To do this, place the same symbol in different symbol sets and allow the individual user groups (clients) either to access a symbol set or not. An on-site operator or an operating data record, for example, receives more information and access to the same symbols as remote maintenance.

See also

- *Chapter 1.4.1.20.2.25 "Object 'Symbol Configuration'" on page 927*
- *Chapter 1.4.1.19.6.2.41 "Attribute 'symbol'" on page 728*
- *Chapter 1.4.1.19.6.2.44 "Effects of Pragmas on Symbols " on page 729*
- *Chapter 1.4.1.19.1.4.8.6 "SFC element properties" on page 493*
- *Chapter 1.4.1.20.2.8.4 "Tab 'Applications'" on page 845*
- *Chapter 1.4.1.20.3.6.23 "Command 'Simulation" on page 1044*

Creating a symbol configuration

Requirement: The project can be compiled without any errors.

1. Select the *"Application"* object in the device tree.
2. Click *"Project ➔ Add Object ➔ Symbol Configuration"*.
   ⇨ The *"Symbol Configuration"* object is added to the device tree and the objects editor opens.
3. Open the *"View"* menu of the editor and activate the categories of variables that should be provided in the configuration editor. Click *"Build"* in the symbol configuration editor.
   ⇨ All variables (according to the currently defined filter in the *"View"* menu) are displayed in a tree structure.
4. Select the check boxes of individual variables.

   Note: Pay attention to the current settings (see the *"Settings"* button in the menu bar of the editor).
   ⇨ In the field below the menu bar of the editor, information is provided about the current situation with accompanying instructions, as well as controls for corrective actions.

5. Follow the prompt in the field below the menu bar. In the following case, this should be only the information that the modified symbol configuration is transferred with the next download or online change.

   Click *"Build ➔ Generate Code"* on the CODESYS menu bar.

   ⇨ The `<project name>.<device name>.<application name>.xml` file is generated in the project directory.

CODESYS transmits the symbol configuration to the PLC for an application download or online change.

See also

- *Chapter 1.4.1.20.2.25 "Object 'Symbol Configuration'" on page 927*

Creating symbol sets with different access rights for different control clients

A symbol set is a defined set of symbols. If supported by the target device, you can combine different symbol sets from the symbols of the application in the symbol configuration editor. The information about the symbol sets is downloaded to the controller. Then you can define the user group that has access to each symbol set. Rights are assigned on the *"Symbol Rights"* tab of the device editor.

As a result, symbol sets allow different client-specific access rights to a symbol in the controller.

You can download changes to a symbol set definition to the controller in an online change. When the application is deleted on the controller, the symbol sets are also deleted. When building the application, you can create and save a symbol file in XML format for each symbol set.

In the following section, you will see an example of steps for creating symbol sets and the assignment of rights on the controller:

Requirements: The application has a defined symbol configuration in the project. The *"Enable symbol sets"* option is enabled in the settings of the symbol configuration. The controller has a user management. For the example here, there should be a user group that has the necessary rights for the servicing of the plant. By default, this type of user group, named "Service", is already created.

1. Define the connection to the controller in the *"Communication Settings"* of the device editor.
2. Click the button in the editor of the *"Symbol Configuration"* in order to create a new symbol set. Specify a name of the group ("Startup") in the *"Add New Symbol Set"* dialog.
3. Click the button (*"Build"*) in the toolbar of the dialog in order to display all symbols available in the project. Select the users who should belong to the group. Save the project.
4. Click the *"Configure Symbol Rights"* button.

   ⇨ The *"Symbol Rights"* tab of the device editor opens.
5. Click the button (*"Synchronization"*) to synchronize the display of the symbol sets with the device.

   ⇨ If you have not enabled user management on the controller yet, then you will see a dialog in the *"Users and Groups"* tab prompting you to do it.
6. Click *"OK"* in the dialog and click the *"Users and Groups"* tab.

   Click the button (*"Synchronization"*). Click *"Yes"* to confirm that user management should be enabled.

   ⇨ The *"Device User Logon"* dialog opens.
7. Sign in. If this is the first login, use "Administrator" as the user name and password, and then set a new password in the following dialog.

   ⇨ After the dialog is closed, the configurations of the device user management are displayed in the *"Users and Groups"* and *"Access Rights"* tabs.

8. Log in to the controller by clicking *“Online ➔ Login”*. Click *“Yes”* to the prompt of whether or not the application should be downloaded to the device.
9. After successful login, click the *“Symbol Rights”* tab. Click the *“Synchronization”* button.
   ⇨ In *“Symbol Sets”*, you see all sets that have currently been downloaded for the application (for this example, at least "startup"). In *“Rights”*, a table shows the user groups that are created in the user management of the controller. In the example, we assume that the default groups *“Administrator”* and *“Service”* have been created. When a symbol set is selected on the left, you see on the right the access rights of the individual user groups to this symbol set (: access granted; : access not granted). The possible type of access is already defined for each symbol in the symbol configuration (read, write, execute).
10. On the left, select the *“Startup”* symbol set and double-click the preset minus sign for "Administrator" as well as for "Service".
    ⇨ The symbol changes into a plus sign. The "Administrator" and "Service" now have access to the symbols in the *“Startup”* symbol set.

See also

- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

1.4.1.9.3 Network Variables

The values of network variables can be exchanged between different PLCs in a network. The variables must be defined in strict, identical lists on both the sender device and receiver device, and only one device application defines the network variables. The lists can be in one or more projects.

The network variable list in the sender is a global variable list where specific log and transfer parameters are defined in their object properties. By adding these properties, you create a "network variable list (sender)" from an ordinary *“GVL”*. You can also insert a *“Network Variable List (Sender)”* object directly into the device tree when this object already has these parameters set.

The network variable list in the receiver is of type *“Network Variable List (Receiver)”*. When creating one, select the respective network variable list of the server. As an alternative, you can read this variable list from an export file that was generated from the sender list. An export file is required anyway for defining the sender list in another project.

The network variables are transmitted as broadcasting in one direction only: sender to receiver. However, it is also possible for a device to contain both sender and receiver lists.

For the `NetVarUdp` library version 3.5.7.0 and later, a receiver channel is no longer assigned when confirmed transfer is not selected. In this way, network variable exchange is also possible between two controllers on one hardware device .

NOTICE!

- If the exchanging devices should be senders and receivers, then the variable list identifiers must be unique in order to prevent abnormal operation. The variable list identifiers are defined in the *"Properties"* dialog of an object GVL.
- Data exchange via network variables is not possible when:
  - The device (target system) does not support it.
  - A firewall blocks the communication.
  - Another client or application is using the UDP port that is set in the properties of the network variable list.
  - More than one application per sender device and receiver device use network variable lists.
- Only arrays that have limits defined with a literal or constant are transmitted to the receiver application. Constant expressions are not permitted for this purpose.

  Example: "`arrVar : ARRAY[0..g_iArraySize-1] OF INT ;`" is not transmitted, but "`arrVar : ARRAY[0..10] OF INT ;`" is transmitted.
- The maximum size of a network variable is 255 bytes. The possible number of network variables is unlimited.
- If the size of the GVL exceeds the maximum length of the network telegram, then the data is split into multiple telegrams. Depending on the configuration, this can result in data inconsistencies.

Communication by means of network variables is also possible when the PLCs operated with applications from different versions of the development system (V2.3, V3). However, in this case, you cannot use the export/import mechanism for matching the variable lists exactly in the sender and receiver projects. The reason is that an variable export file (`*.exp`) that is generated from V2.3 does not include the required amount of information necessary for creating a receiving NVL in V3. There is no respective network parameter configuration as a GVL file, which you exported from the sender previously. To get this file, you must recreate the V2.3 NVL in V3 first. Then you can generate an export file and create a receiving NVL in V3 based on this.

An alternate to data exchange between PLCs is the use of data sources. As opposed to the broadcasting method for exchanging network variables, defined point-to-point connections are created between one application and a remote data source.

See also

- *Chapter 1.4.1.20.4.10.11 "Dialog 'Properties' - 'Network Variables'" on page 1163*
- *Chapter 1.4.1.20.2.10 "Object 'GVL' - Global Variable List" on page 871*

Configuring a Network Variable Exchange

The following steps are necessary for exchanging network variables between the sender device and receiver device.

1. Creating a network variable list in the sender device and generating an export file

Requirements: An application is inserted in the device tree of the PLC that has been employed as the sender device.

1. Select the application and insert a *“Network Variable List (Sender)”* object. Make the following settings in the *“Add Network Variable List (Sender)”* dialog: network type: `UDP`, example: "`NVL_Sender`".
2. Double-click the NVL object to open the respective editor and type the declarations of the network variables. Example:

```
VAR_GLOBAL
iglobvar:INT;
bglobvar:BOOL;
strglobvar:STRING;
END_VAR
```

3. Right-click the NVL object in the device tree to open the *“Properties”*. In the *“Properties”* dialog, open the *“Network Variables”* tab. This shows the following settings: Network type: UDP; List identifier: 1; Pack variables; Cyclic transmission: every 50 ms.
4. Note: You can also convert an existing GVL into a network variable list by configuring its network variable properties.
5. Click the *“Link to File”* tab.in the *“Properties”* dialog of the *“NVL_Sender”*. Define a file name `<export>.gvl` and a location in the file system for the export file of the GVL. Select the *“Export before compile”* option.
6. Click *“Build ➔ Generate Code”* to compile the application.

The export file for the network variable list is now located in the defined folder.

See also

- *Chapter 1.4.1.20.4.10.11 “Dialog 'Properties' - 'Network Variables'” on page 1163*

2. Creating an associated network variable list in the receiver device

Requirements: A sender device and a receiver device exist in the device tree. An application with a task configuration is inserted below the device. An NVL or a GVL is created below the sender device as network variable list to be sent.

1. Select the application of the receiver in the device tree and click *“Add Object ➔ Network Variable List (Receiver)”*.

   ⇨ The *“Add Network Variable List (Receiver)”* dialog opens.
2. In the dialog, select the previously created NVL of the sender device and type a name (for example, "NVL_Receiver"). CODESYS populates this receiver list automatically with the variable declarations from the sender list.

   Note: As an alternative, you can select the *“Import from file”* option and load the export file that was generated previously from the sender list.

3. Testing the network variable exchange

Requirements: A network variable list (sender) exists in the sender device, a network variable list (receiver) exists in the receiver device, and both lists have identical variable declarations.

1. Below the application in the sender device, create a program that increments a network variable. Example: `iglobvar:=iglobvar+1;`.
2. Configure the application task so that this program calls it.

3. Below the application in the receiver device, create a program that writes the value of this network variable to a local variable. Example: `ilocalvar:=iglobvar;`.
4. Configure the application task so that this program calls it.
5. Download both applications to the controllers and start them. (Set the application as active, login, download, and start.)
6. In the online views of the editors of both programs, check whether the values of `iglobvar` match in the receiver and the sender.

See also

- *Chapter 1.4.1.20.2.16 “Object 'Network Variable List (Sender)'” on page 880*
- *Chapter 1.4.1.20.2.17 “Object 'Network Variable List (Receiver)'” on page 880*

1.4.1.9.4 Data Link with Data Sources

In order to have read/write access to the remote devices and their running applications, you can add a data source manager to your application with one or more data sources.

The functionality of the data source manager allows for establishing connections and communication to remote devices, and it makes its data available through data source variables. At this time, the partners communicate by means of a point-to-point connection. Depending on the network where the controllers are located, a connection is established via the CODESYS data source types or CODESYS ApplicationV3.

Data source type CODESYS Symbolic

The data source type CODESYS Symbolic is available **only** together with a CODESYS HMI device. However, then it is advantageous to use this type.

Below a CODESYS HMI device, you can configure the data link either with data source type CODESYS Symbolic or with data source type CODESYS ApplicationV3. We recommend that you select the data source type CODESYS ApplicationV3 only when there are no resources for the symbol configuration available on the remote device. For example, this is the case with embedded or mini PLCs whose applications often do not contain a symbol configuration.

The requirement for a connection setup is that symbols have been configured in the remote device and as a result a symbol file exists. The application in the remote device has a symbol configuration. Then the data link can take place via symbolic monitoring.

In the case of symbolic monitoring, the symbol file on the remote device is read and the stored variable information is used for the data source variables and the data transfer. The advantage is that the application does not have to be updated in the local device when someone modifies the remote application without updating its symbol configuration. If the symbol file is also located on your development system (either a file or a symbol configuration object as part of your project), then the local symbol file can also be read. Then you can work offline during the development phase.

During the development phase, you can create a variable list offline by means of local symbol configuration files. In this way, you can develop a local application offline based on the symbol information without a connection to a data source.

The following connection types are possible:

- *“CODESYS V2”*:
  The devices exist in the same network. The V2 runtime on the remote PLC provides a communication interface.
- *“CODESYS V2 (Via gateway)”*
  The devices do not exist in the same network. They are connected via a V2 gateway.
  Note: For this connection, a *“CoDeSys V2.3 Gateway Server”* (V2 gateway) has to be installed on the development computer where CODESYS V3 is running.

- *“CODESYS V3”*
  The devices exist in the same network. The V3 runtime on the remote PLC provides a communication interface.
- *“CODESYS V3 (Via gateway)”*:
  The devices do not exist in the same network. They are connected via a V3 gateway.

Data source type CODESYS ApplicationV3

This data source type is available below all device types.

The data link with CODESYS ApplicationV3 data source type in done by means of address monitoring. This requires that the address information between the remote PLC and the local device match. The runtime system of the local application needs valid communication parameters in order to establish the connection.

The network scan function can support you when configuring the data source.

Disadvantage: If you modify the remote application, then you also have to update the local application afterwards (for example, the HMI application.

The advantage is that a symbol configuration is not required in the remote application.

Data transmission

At runtime of the local application, the data source variables that appear in the data source editor of the *“Variables”* tab are updated in configurable time intervals. The remote application is also executed at this time. Variables that are configured in the visualization, in the trend, as alarms, or for recipes are transferred and stored automatically. When a variable is accessed in IEC code only, the variable is not updated automatically. In this case, you have to select the *“Update always”* option in the data source editor of the *“Variables”* tab.

The data source types support the (read or write) data access to variables of the source PLC for the following data types:

- Scalar value at top level
  Example: `PLC_PRG.hugo`
- Property to a program or GVL by means of a call when it is marked with `{attribute monitoring := 'call'}`.
  Example: `PLC_PRG.PropertyCall`
- Variable which is mapped to bit addresses
  Example in `PLC_PRG`: `x AT %MX0.5 : BOOL;`
- Variable (type `BIT`) in a function block
  Example: Declaration in `DUT`: `x, y : BIT`, bit access: `PLC_PRG.dutInst.y`
- Structured obtainable variable
  Example: `PLC_PRG.outerInst.innerInst.dwVar`
- Property to POU instance when it is marked with `{attribute monitoring := 'call'}`.
  Example: `PLC_PRG.instance.PropertyCall`
- Property at top level and to an instance when it is marked with `{attribute monitoring := 'variable'}`.
  Note: This cannot be written by monitoring or by the data sources.
- Array access with literal index
  Example: `PLC_PRG.arrOfInts[3]`, `PLC_PRG.inst.arrOfBool[1]`
- Nested access (for example, array of structures or structure of arrays)

CAUTION!
Bit access used in visualizations that are transferred via a data source connection function only if they contain literal offset specifications. A visualization cannot process an offset specification by defined constants.

See also

- *Chapter 1.4.1.20.2.4 “Object 'Data Source Manager'” on page 821*
- *“Dialog 'Add Data Source'” on page 822*
- *Chapter 1.4.1.20.2.5 “Object 'Data Source'” on page 823*

Initially Adding a Data Source

For data exchange between your local device and a remote device, add a *“Data Source Manager”* object and then a *“Data Source”* below your application in the device tree. A wizard guides you through the configuration of the data source. Afterwards, you can change the settings at any time in the editor of the object. However, it is not possible to modify the data source type later.

Use the “CODESYS Symbolic” data source type unless there are no resources available in the remote PLC for a symbol configuration. As long as the symbol configuration in the remote device is not impacted by an application change, you have the advantage that the application in the local device does not have to be updated.

When adding a data source, select a data source type. Then specify the connection settings of the point-to-point connection to the remote device. Ideally, the remote device is running during this time and CODESYS can establish the connection to it immediately. Then all available data source variables from the remote PLC are displayed. Select the variables that should be transferred. You can also select all variables. Then the data source is initialized automatically, the data source variables are created below the *“DataSources_Objects”* folder, and another data source is added below the data source manager.

If the data is transferred using symbolic monitoring and the symbol file is stored on your development system, then you can read the variable information from the symbol file and work offline. The symbol file is stored either as a file on your development system or as a symbol configuration object as part of your project (in CODESYS).

The initial settings can be changed at any time in the data source editor.

See also

- *Chapter 1.4.1.20.2.4 “Object 'Data Source Manager'” on page 821*

Initially connecting devices symbolically via 'CODESYS V3'

A *“CODESYS Control Win V3”* is running on the remote device. Moreover, an application is running with a symbol configuration.

1. Below your application in the device tree, insert a *“Data Source Manager”* object.
2. Select the *“Data Source Manager”* object and click *“Add Object ➔ Data Source”*.
   ⇨ The *“Add Data Source”* dialog opens.
3. In the *“Name”* field, specify the data source name.
   ⇨ Example: `ds_Remote_Device`

4. As the data source type, select *“CODESYS Symbolic”*.
   ⇨ The data transfer is done via symbolic monitoring. The *“Initialize Data Source Wizard - Provider settings”* dialog opens.
5. As the connection type, select *“CODESYS V3”*.
6. For *“Type of name or address”*, select the *“Node name”* option.
7. In the *“Connection Settings”* group, specify the connection parameters for configuring the remote device. Example: *“[03A7)”*
   ⇨ The connection to the remote device is established and the application is read. The *“Initialize Data Source Wizard - Browse data items”* dialog also opens. The read remote control variables are displayed in the tree view on the *“Variables”* entry. The top node is the application, which is displayed with its remote application name.
8. In the tree view, select which control variables should be transferred. Then click *“Finish”*.
   ⇨ The data source is initially configured. The `ds_Remote_Device` object is added below the *“Data Source Manager”* node. The object is open, and on the *“Variables”* tab, the data source variables to be generated are displayed in the tree view. The GVL `ds_Remote_Device`, where the data source variables are declared, is located below the *“DataSource_Objects”* folder.

Initially connecting devices symbolically via 'CODESYS V3 (Via gateway)'

A *“CODESYS Control Win V3”* is running on the remote device. Moreover, an application is running with a symbol configuration. The remote device exists in another network so that the communication has to be routed via a gateway.

1. Below your application in the device tree, insert a *“Data Source Manager”* object.
2. Select the *“Data Source Manager”* object and click *“Add Object ➔ Data Source”*.
   ⇨ The *“Add Data Source”* dialog opens.
3. In the *“Name”* field, specify the data source name.
   ⇨ Example: `ds_Remote_Device`
4. As the data source type, select *“CODESYS Symbolic”*.
   ⇨ The data transfer is done via symbolic monitoring. The *“Initialize Data Source Wizard - Provider settings”* dialog opens.
5. As the connection type, select *“CODESYS V3 (Via gateway)”*.
   ⇨ You can also specify the communication parameters for the gateway.
6. For *“Type of name or address”*, select the *“Node name”* option.
7. In the *“Connection Settings”* group, specify the connection parameters for configuring the remote device. Example: *“[03A7)”*
   ⇨ The connection to the remote device is established and the application is read. The *“Initialize Data Source Wizard - Browse data items”* dialog also opens. The read remote control variables are displayed in the tree view on the *“Variables”* entry. The top node is the application, which is displayed with its remote application name.
8. In the tree view, select which control variables should be transferred. Then click *“Finish”*.
   ⇨ The data source is initially configured. The `ds_Remote_Device` object is added below the *“Data Source Manager”* node. The object is open, and on the *“Variables”* tab, the data source variables to be generated are displayed in the tree view. The GVL `ds_Remote_Device`, where the data source variables are declared, is located below the *“DataSource_Objects”* folder.

Initially connecting devices symbolically via 'CODESYS V2'

A *"CoDeSys V2.3 SP PLCWinNT V2.4"* is running on the remote device. Moreover, an application is running with a symbol configuration.

1. Below your application in the device tree, insert a *"Data Source Manager"* object.
2. Select the *"Data Source Manager"* object and click *"Add Object ➔ Data Source"*.
   ⇨ The *"Add Data Source"* dialog opens.
3. In the *"Name"* field, specify the data source name.
   ⇨ Example: `ds_Remote_Device`
4. As the data source type, select *"CODESYS Symbolic"*.
   ⇨ The data transfer is done via symbolic monitoring. The *"Initialize Data Source Wizard - Provider settings"* dialog opens.
5. As the connection type, select *"CODESYS V2"*.
6. In the *"Connection Settings"* group, specify the connection parameters for configuring the remote device.
   Example: driver type `TCP/IP (Level 2 Route)`, address `localhost`, port `1200`
   ⇨ The connection to the remote device is established and the application is read. The *"Initialize Data Source Wizard - Browse data items"* dialog also opens. The read remote control variables are displayed in the tree view on the *"Variables"* entry.
7. In the tree view, select which control variables should be transferred. Then click *"Finish"*.
   ⇨ The data source is initially configured. The `ds_Remote_Device` object is added below the *"Data Source Manager"* node. The object is open, and on the *"Variables"* tab, the data source variables to be generated are displayed in the tree view. The GVL `ds_Remote_Device`, where the data source variables are declared, is located below the *"DataSource_Objects"* folder.

Initially connecting devices symbolically via 'CODESYS V2 (Via gateway)'

A *"CoDeSys V2.3 SP PLCWinNT V2.4"* is running on the remote device. Moreover, an application is running with a symbol configuration. The remote device exists in another network so that the communication has to be routed via a gateway.

1. Below your application in the device tree, insert a *"Data Source Manager"* object.
2. Select the *"Data Source Manager"* object and click *"Add Object ➔ Data Source"*.
   ⇨ The *"Add Data Source"* dialog opens.
3. In the *"Name"* field, specify the data source name.
   ⇨ Example: `ds_Remote_Device`
4. As the data source type, select *"CODESYS Symbolic"*.
   ⇨ The data transfer is done via symbolic monitoring. The *"Initialize Data Source Wizard - Provider settings"* dialog opens.
5. As the connection type, select *"CODESYS V2 (Via gateway)"*.
   ⇨ You can also specify the communication parameters for the gateway.
6. In the *"Connection Settings"* group, specify the connection parameters for both the gateway and the device configuring the remote device.
   Example: driver type `TCP/IP (Level 2 Route)`, address `localhost`, port `1200`
   ⇨ The connection to the remote device is established and the application is read. The *"Initialize Data Source Wizard - Browse data items"* dialog opens. The remote control variables are displayed in the tree view on the *"Variables"* entry.

7. In the *“Connection Settings”* group, specify the connection parameters for configuring the remote device. Example: *“[03A7)”*

   ⇨ The connection to the remote device is established and the application is read. The *“Initialize Data Source Wizard - Browse data items”* dialog opens. The read remote control variables are displayed in the tree view on the *“Variables”* entry. The top node is the application, which is displayed with its remote application name.

8. In the tree view, select which control variables should be transferred. Then click *“Finish”*.

   ⇨ The data source is initially configured. The `ds_Remote_Device` object is added below the *“Data Source Manager”* node. The object is open, and on the *“Variables”* tab, the data source variables to be generated are displayed in the tree view. The GVL `ds_Remote_Device`, where the data source variables are declared, is located below the *“DataSource_Objects”* folder.

Initially adding data source variables from a symbol file

Ideally, the same symbol file on the remote device is saved on your development system.

1. Below your application in the device tree, insert a *“Data Source Manager”* object.
2. Select the *“Data Source Manager”* object and click *“Add Object ➔ Data Source”* in the context menu.

   ⇨ The *“Add Data Source”* dialog opens.

3. In the *“Name”* field, specify the data source name.

   ⇨ Example: `ds_Symbols`

4. As the data source type, select *“CODESYS Symbolic”*.

   ⇨ The data transfer is done via symbolic monitoring. The *“Initialize Data Source Wizard - Provider settings”* dialog opens.

5. In *“Variable information”*, click the *“From symbol file”* entry.
6. In *“Select symbol file”*, specify the location and the file name of the symbol file. When the code is generated, an XML symbol file is created in the project directory by default.

   ⇨ Example: `D:\Projects\V3.5 SP11\Project_A.Device.Application.xml`

   Hint: When a symbol file is specified, no additional connection settings have to be configured. A connection is not established. You are working offline. You have to configure the connection settings only when you need current data from the controller which is transferred online. In the *“Variable information”* settings, select the *“From connection settings”* option.

7. Click the *“Next”* button.

   ⇨ The *“Initialize Data Source Wizard - Browse data items”* dialog opens. The read symbols are displayed in the tree view on the *“Variables”* entry.

8. In the tree view, select the symbols to be transferred. Then click *“Finish”*.

   ⇨ The data source is initially configured. The `ds_Symbols` object is added below the *“Data Source Manager”* node. The object is open, and on the *“Variables”* tab, the data source variables that were generated based on the symbol file are displayed in the tree view. The GVL `ds_Symbols`, where the data source variables are declared, is located below the *“DataSource_Objects”* folder.

Initially adding data source variables from a symbol configuration

Your active project contains the control application for the remove device. The control application includes a symbol configuration with symbols that are added to your local application as data source variables.

1. Below your local application in the device tree, insert the *"Data Source Manager"* object.
2. Select the *"Data Source Manager"* object and click *"Add Object ➔ Data Source"*.
   ⇨ The *"Add Data Source"* dialog opens.
3. In the *"Name"* field, specify the data source name.
   ⇨ Example: `ds_Symbols`
4. As the data source type, select *"CODESYS Symbolic"*.
   ⇨ The data transfer is done via symbolic monitoring. The *"Initialize Data Source Wizard - Provider settings"* dialog opens.
5. In *"Variable information"*, select the *"<remote device>.<application>.symbol configuration"* entry.
   ⇨ Example: `Device.Application.Symbol Configuration`

   Hint: When a symbol file is specified, no additional connection settings have to be configured. A connection is not established. You are working offline.
6. Click the *"Next"* button.
   ⇨ The *"Initialize Data Source Wizard - Browse data items"* dialog opens. The read symbols are displayed in the tree view on the *"Variables"* entry.
7. In the tree view, select the symbols to be transferred. Click *"Finish"*.
   ⇨ The data source is initially configured. The `ds_Symbols` object is added below the *"Data Source Manager"* node. The object is open, and on the *"Variables"* tab, the data source variables that were generated based on the symbol configuration are displayed in the tree view. The GVL `ds_Symbols`, where the data source variables are declared, is located below the *"DataSource_Objects"* folder.

Initially connecting devices with address monitoring

A *"CODESYS Control Win V3"* is running on the remote device. The project of the remote device is located on your development computer. The engineered application there does not contain a symbol configuration.

Use this communication link only if there are no resources available in the remote PLC for a symbol configuration.

1. Below your application in the device tree, insert a *"Data Source Manager"* object.
2. Select the *"Data Source Manager"* object and click *"Add Object ➔ Data Source"*.
   ⇨ The *"Add Data Source"* dialog opens.
3. In the *"Name"* field, specify the data source name.
   ⇨ Example: `ds_Remote_Device`
4. As the data source type, select *"CODESYS ApplicationV3"*.
   ⇨ The data transfer is done by means of address monitoring. The *"Initialize Data Source Wizard - Provider settings"* dialog opens.
5. For *"Select the project type"*, select the *"Other Project"* option.
6. For *"Choose file"*, specify the file and location of the project on the remote device. Example: `C:\Data\Projects\PLC_A.project`.
   ⇨ The remote device is displayed in the tree view of the window below, and as a result the connection was established.
7. Click the *"From device"* link.
   ⇨ The connection parameters to the remote device are read and displayed in the dialog. The connection is configured.

8. Click *“Next>”*.

   ⇨ The *“Initialize Data Source Wizard - Browse data items”* dialog opens. The remote control variables are displayed in the tree view on the *“Variables”* entry.

9. In the tree view, select which control variables should be transferred. Then click *“Finish”*.

   ⇨ The data source is configured. A connection is established. The settings are stored in the object and can be modified in the editor of the object.

   The data source is initially configured. The `ds_Remote_Device` object is added below the *“Data Source Manager”* node. The object is open, and on the *“Variables”* tab, the data source variables to be generated are displayed in the tree view. The GVL `ds_Remote_Device`, where the data source variables are declared, is located below the *“DataSource_Objects”* folder.

See also

- *Chapter 1.4.1.20.2.5.3 “Tab 'Communication' via CODESYS Symbolic” on page 826*
- *Chapter 1.4.1.20.2.5.4 “Tab 'Communication' via CODESYS ApplicationV3” on page 831*

Editing data source variables

In runtime mode, the remote data is saved to the data source variables. The data source variables and their mapping to the remote variables are displayed in the data source editor below of the *“Variables”* tab. If the local and remote variables have the same names and the same data types, then the data is mapped 1:1. The variables and the data types are created automatically. That is the regular procedure.

You can also map to existing variables. This is necessary, for example, if a visualization includes a data type in an interface. Then the same data must be passed to this visualization. In this case, the declared local variable and the remote variable have the same data type, for example from one library. Moreover, you can map a local variable with a conforming data type to a remote variable. The data type can be created in the *“Type Mappings”* tab.

The specifically created variables and data types are declared in the *“DataSources_Objects”* folder. For each data source, a global variable list of the same name as the data source is declared there. Moreover, the data source variables usually have the identical or conforming data type as the remote control variable and they are declared as user-defined data types (DUT objects). Considering all data sources, multiple declaration of the same data types is avoided.

Do not edit the data interface in the *“DataSources_Objects”* folder manually. It is created initially when adding a data source. Changes can be made later in the editor of the data source.

See also

- *Chapter 1.4.1.9 “Working with Controller Networks” on page 352*
- *Chapter 1.4.1.9.4.4 “ Updating data interfaces” on page 373*
- *Chapter 1.4.1.20.2.5.1 “Tab 'Variables'” on page 824*

Selecting variables for data transfer

You can edit the selection of the data source variables.

☑ Requirement: The remote device and its application are running. A data source manager is already inserted below the local application with a data source.

1. Open the editor of the data source.
2. Select the *“Variables”* tab.
3. Click *“Update Variables”*.

   ⇨ The *“Browse Variables”* dialog opens.

4. Activate the variables that should be transferred and click *“OK”* to close the dialog.

   ⇨ The data source variables are modified according to the selection. The declaration of variables and data types is also modified.

   The *“Variables”* tab shows the modified selection. Moreover, the mapped remote variable is listed in the *“Remote variable”* column.

See also

- *Chapter 1.4.1.9 “Working with Controller Networks” on page 352*
- *Chapter 1.4.1.9.4.4 “ Updating data interfaces” on page 373*
- *Chapter 1.4.1.20.2.5.1 “Tab 'Variables'” on page 824*

Mapping remote variables to a new variables

You need to map a remote variable to a global implicit variable that is created new. That is the regular procedure for transposing data source to 1:1.

☑ Requirement: A project is open. A data source manager and a data source below it are located in the device tree of the local application.

1. Open the editor of the data source.
2. Select the *“Variables”* tab.

   ⇨ The data source variables are listed.
3. Select a variable and click the symbol in the *“Create or map”* column.
4. Specify a name in *“Local variable”*.

   ⇨ A variable is declared automatically and it contains the same value as the mapped remote variable.

Mapping remote variables to a existing variables

You need to map a remote variable to an existing variable.

☑ Requirement: A data source manager and a data source below it are located in the device tree of the local application. The remote data that should be transferred is displayed in the editor of the data source in the *“Variable”* tab

1. Open the editor of the data source.
2. Select the *“Variables”* tab.
3. Select a variable and click the symbol in the *“Create or map”* column.

   ⇨ A variable contains the same value as the mapped remote variable.

Mapping remote variables to local variables with a conforming data type

First, create a conforming data type and then use it for a variable.

☑ Requirement: A data source manager and a data source below it are located in the device tree of the local application. The remote data that should be transferred is displayed in the editor of the data source in the *“Variable”* tab

1. Open the editor of the data source.
2. Select the *“Type Mappings”* tab.

3. Select the data type in the list that you want to edit.
   ⇨ The elements of the data type are listed in the window below the data type list
4. Specify a name for the data type. Example: `DataType_A`. Select the name for the remote data types to which the local type should conform. Example: `Library1.DataType_A`.
5. Modify it in the window below the data type list and remove the elements that are not necessary for the data transfer.
6. Select the symbol for this data type in the *“Create or map”* column.
   ⇨ The data type `DataType_A` is declared in the *“DataSources_Objects”* folder.
7. Select the *“Variables”* tab.
8. Specify a name in the *“Local variable”* column. Example: `Var_A`
9. Select the symbol in the *“Create or map”* column.
10. Specify the data type `DataType_A` in the *“Mapping type”* column.
11. Select the remote variable with the data that should be transferred. Example: `appPLC_A.Data_A`. Use the input assistance for this.
    ⇨ A variable `Var_A` is declared automatically with the user-defined data type `DataType_A`. During data transfer, it receives the data of the mapped remote variables.

Example

Library `SnakeUtil`

The example demonstrates how variables of the data source are created. At this time, new variables are created, data is mapped to existing data types and their variables, and new data types are created with type-conforming mapping.

The remote PLC uses POU instances from the `SnakeUtil` library and the HMI device visualizes these POU instances. This is why the HMI application requires a variable in the operating interface that has a data type appropriate for a visualization template. As a result, the `SnakeUtil` library is linked integrated into the HMI application and the HMI variables instantiate the `SnakeUtil.SnakeVisu` visualization function block.

The following library function blocks from the `SnakeUtil` library are used in the remote PLC.

- Function block `SnakeUtil.Snake`: Equipped with much logic and calling from external functions.
- DUT `SnakeUtil.PositionInfo`: Two values (of the variables x and y)
- DUT: `SnakeUtil.DrawingInfo`: Image ID
- The `SnakeUtil.SnakeVisu` visualization function block with transfer parameter `SnakeUtil.Snake` visualizes the `Snake` function block.

The following settings are entered in the editor of the *“Type Mappings”* tab:

SnakeUtil.PositionInfo		PositionInfo	SnakeUtil.PositionInfo
GeneratedType_2		GeneratedType_2	GeneratedType_2
GeneratedType_3		GeneratedType_3	GeneratedType_3
Snake		Snake	SnakeUtil.Snake
SnakeUtil.DrawingInfo		DrawingInfo	SnakeUtil.DrawingInfo

In the visualization, a frame is inserted with a reference to `SnakeUtil.SnakeVisu`. This expects to have the type `SnakeUtil.Snake`.

The data types `SnakeUtil.PositionInfo` and `SnakeUtil.DrawingInfo` are mapped to existing data types (symbol in the *“Create or map”* column). The data types are small and contain data only.

The `SnakeUtil.Snake` function block is very complex and calls external functions that are not available in the HMI visualization. The function block with code is not required in the visualization. You need a less extensive but compatible and conforming type in the HMI visualization. Therefore, do not create the original data type directly. Instead, first modify the original data type and remove the unnecessary elements. Then create the new data type `Snake` by selecting the symbol in the *“Create or map”* column.

Editing Communication

You have added a *“Data Source Manager”* object and below it a *“Data Source”* object below your application in the device tree. The connection parameters are displayed in the data source editor of the *“Communication”* tab. You can modify it there.

The data source type and the current connection type are listed in the status bar. It is not possible to modify the data source type later.

See also

- *Chapter 1.4.1.9.4.1 “Initially Adding a Data Source” on page 365*
- *Chapter 1.4.1.20.2.5.3 “Tab 'Communication' via CODESYS Symbolic” on page 826*
- *Chapter 1.4.1.20.2.5.4 “Tab 'Communication' via CODESYS ApplicationV3” on page 831*

Updating data interfaces

The data source variables are updated cyclically in runtime mode. Only the data is updated that either is used in the current visualization or has the property *“Update always”*.

You can define the time interval. Moreover, you can define variables whose data is transferred in each update interval, and therefore they are always update. To update variables that are not used in the application code, you can implement an update programmatically with the help of interface functions from the data source manager.

> **NOTICE!**
> If data traffic between the remote and local device is too high, then the update rate is reduced automatically. This can lead to an incomplete transfer.

See also

- *Chapter 1.4.1.20.2.5.1 “Tab 'Variables'” on page 824*

Setting the update rate

1. Open the editor of the data source.
2. Click the *“General and Diagnosis”* tab.
3. Specify a value in the *“Update rate”* field.

   Example: `100`

   ⇨ The data from the remote device to the local device is transferred every 100 ms.

See also

- *Chapter 1.4.1.20.2.5.6 “Tab 'General and Diagnosis'” on page 834*

Selecting the variable for 'Update always'

> **NOTICE!**
> Avoid updating too many variables always. Each update produces additional data traffic at the connection between the remote and local devices. When data traffic is too high, the update rate is reduced automatically. This can lead to an incomplete transfer.

1. Open the editor of the data source.
2. Activate the option *“Update always”* for a variable.

   ⇨ The data of the variables is transferred at each update cycle, even when the data has not changed.

See also

- *Further information on page 824*

Updating data programmatically

The data source manager provides interface functions in the `Datasources` library. If a data source manager is integrated in the application code, then the global variable `g_Datasources` is instantiated automatically. This provides access to the interface functions.

Then you can update individual variables that are not called in the active visualization.

Example

The variable `ivar` is activated and deactivated by means of methods from the `Datasources` library so that its value is transferred. Furthermore, you can configure that the variable is updated only over a defined duration in order to save transfer capacity.

```
//Synchronize with DatasourcesTask and block until access is
possible
//Regard the feedback in ERR_OK or in ERR_DE_MULTITASKING_LOCKED
g_Datasources.BeginDataConfiguration(TRUE);
// Activate variable
g_DataServer.UseData(ADR('RemoteDevice.Application.PLC_PRG.iVar'));
// Deactivate variable
//
g_DataServer.ReleaseData(ADR('RemoteDevice.Application.PLC_PRG.iVar'
));
g_DataServer.EndDataConfiguration();
```

The data configuration is started with `BeginDataConfiguration(TRUE)`, thus initializing the synchronization of the task `DatasourceTask` with the application task. The value `TRUE` blocks the processing until the access to the variable is possible; `FALSE` repeats access attempts without blocking. The return values `ERR_OK` and `ERR_DS_MULTITASKING_LOCKED` provide feedback about the access attempts.

When synchronization is successful, the variable is activated by means of the `UseData` method. Then the data configuration is completed with the `EndDataConfiguration` method and the synchronization triggered again with the task `DatasourceTask`.

The `ReleaseData` method is used in the same way for deactivating the variable again at the desired processing time.

Using remote data

The variables that are listed in the data source editor of the *"Variables"* tab (and declared in the *"DataSources_Objects"* folder) can be used in your application like IEC variables. For example, you can visualize the variables.

If multiple data sources are available and therefore conflicts occur regarding unique variable names, then you must specify the data source name as the prefix. If no conflicts occur, then this is not necessary and you can map the variables without a data source prefix.

```
<data source name>.<function block name>.<variable name>
```

Displaying variable values from the remote device

You need to show the variable value `iTemp` of a remote device in a visualization element of a visualization in the local application (with the data source manager).

Initial situation: A data source `dsRemotePLC` is below the local data source manager where the connection to the remote device is configured. In addition, the variable `iTemp` is selected in the data source editor of the *"Variables"* tab.

1. Select the visualization element in the editor view. Select the properties *"Text variables"* - *"Text variable"* in the *"Properties"* view.
2. Select the `iTemp` variable.
   ⇨ The variable mapping is qualified. Example: `dsRemotePLC.PLC_PRG.iTemp`.
3. Select the *"Text"* property of the visualization element and type in the following: `Temperature: %s`
   ⇨ The value of the `iTemp` variable from the remote device `RemoteDevice` is displayed.
4. Download and start the remote application.
5. Download and start the local application.
   ⇨ The visualization starts and displays the actual value of `iTemp`.

NOTICE!
The visualization integrated in CODESYS does not display actual values of variables that are transferred by means of a data source connection. The integrated visualization displays only the initialization values or the last otherwise delivered values because they do not establish a connection to the data sources.

NOTICE!
If variables are used that are not called in the visualization code, then the variables must be updated in the application code by means of functions from the data source interface.

Establishing an Encrypted Connection of a Data Source OPC UA Client to an OPC UA Server

Requirement:

- An OPC UA Server is available. For a description of the OPC UA Server which is included in the standard installation CODESYS, see the chapter "OPC UA Server".
- You have installed the CODESYS Security Agent add-on in CODESYS.
- CODESYS is open.
- The *"Allow anonymous login"* option is selected for your controller in the *"Change Communication Policy"* dialog of the device editor (*"Communication Settings"* tab, *"Change Communication Policy"* command, *"Device"* menu). Or the user management has been explicitly disabled (for example, by switching to *"Optional user management"* in the *"Change Communication Policy"* dialog) and then *"Reset Origin"*.

1. Start the OPC UA Server.
2. Create a new CODESYS project.
3. Add a *"Data Source Manager"* object to the application.
4. Add a *"Data Source" "OPCUAClient"* to the *"Data Source Manager"*.
   ⇨ The *"Datasource"* dialog opens.
5. In the *"Datasource"* dialog, specify the URI of the started OPC UA Server and select the *"Information Model Source"*. When you select *"Online"* as the *"Information Model Source"*, the OPC UA Client connects to the OPC UA Server and reads the information about which variables and types exist. When you select *"Offline"*, the client reads the same information from an installed information model and does not require a running OPC UA Server to do this.
6. For *"Message Security Mode"*, select *"Sign and Encrypt"*.
   Note: You should use *"Message Security Mode"* = *"None"* only in closed networks.
7. Click *"Next"*.Now the client scans the OPC UA Server to find the variables and types of the OPC UA Server. The OPC UA Server has to be online to do this.
8. Now select one or more variables.
   ⇨ These variables can be exchanged later via encrypted communication between the OPC UA Client and the OPC UA Server. For the variables, components are created in the *"Devices"* view, in the *"DataSources_Objects"* folder. The variables can be used in the application.
9. In the next steps, you create a certificate for the encrypted communication from the OPC UA Client to the OPC UA Server.
10. Click *"View* ➔ *Security Screen"*.
11. Select the *"Devices"* tab.

12. Select the controller in the left view.
    ⇨ In the right view, all services of the controller are displayed which require a certificate.
13. Select the service *"CmpOPCUAClient"*.
14. Create a new certificate for the device. Click the  icon.
    ⇨ The *"Certificate Settings"* dialog opens.
15. Define the certificate parameters and click *"OK"* to close the dialog.
    ⇨ The certificate is created on the controller.
16. Click the  button and save the certificate to the local file directory of the OPC UA Server, in the folder `certs`.
    ⇨ Now when you restart the OPC UA Server, it will recognize the client certificate. The the server sends its certificate to the client. In the following steps, this certificate will be made "trusted" to the client.
17. To do this, in the *"Security Screen"* view, on the *"Devices"* tab, click the *"Certificates in Quarantine"* folder in the left area.
    ⇨ The certificate is displayed in the right area.
18. Drag this certificate to the *"Trusted Certificates"* folder.
    ⇨ Now the server certificate is "trusted" by the client.
19. Now when you connect to the controller and the application starts, the data source variables of the OPC UA Client can be exchanged with the OPC UA Server via the encrypted connection.

See also

- *Chapter 1.4.1.20.2.5.5 "Tab 'Communication' via OPC UA Server" on page 834*

Establishing an Encrypted Connection of a Data Source OPC UA Client to an OPC UA Server

Requirement:

- An OPC UA Server is available. For a description of the OPC UA Server which is included in the standard installation CODESYS, see the chapter "OPC UA Server".
- You have installed the CODESYS Security Agent add-on in CODESYS.
- CODESYS is open.
- The *"Allow anonymous login"* option is selected for your controller in the *"Change Communication Policy"* dialog of the device editor (*"Communication Settings"* tab, *"Change Communication Policy"* command, *"Device"* menu). Or the user management has been explicitly disabled (for example, by switching to *"Optional user management"* in the *"Change Communication Policy"* dialog) and then *"Reset Origin"*.

1. Start the OPC UA Server.
2. Create a new CODESYS project.
3. Add a *"Data Source Manager"* object to the application.
4. Add a *"Data Source"* *"OPCUAClient"* to the *"Data Source Manager"*.
   ⇨ The *"Datasource"* dialog opens.
5. In the *"Datasource"* dialog, specify the URI of the started OPC UA Server and select the *"Information Model Source"*. When you select *"Online"* as the *"Information Model Source"*, the OPC UA Client connects to the OPC UA Server and reads the information about which variables and types exist. When you select *"Offline"*, the client reads the same information from an installed information model and does not require a running OPC UA Server to do this.

6. For *“Message Security Mode”*, select *“Sign and Encrypt”*.

   Note: You should use *“Message Security Mode”* = *“None”* only in closed networks.

7. Click *“Next”*.Now the client scans the OPC UA Server to find the variables and types of the OPC UA Server. The OPC UA Server has to be online to do this.

8. Now select one or more variables.

   ⇨ These variables can be exchanged later via encrypted communication between the OPC UA Client and the OPC UA Server. For the variables, components are created in the *“Devices”* view, in the *“DataSources_Objects”* folder. The variables can be used in the application.

9. In the next steps, you create a certificate for the encrypted communication from the OPC UA Client to the OPC UA Server.

10. Click *“View ➔ Security Screen”*.

11. Select the *“Devices”* tab.

12. Select the controller in the left view.

    ⇨ In the right view, all services of the controller are displayed which require a certificate.

13. Select the service *“CmpOPCUAClient”*.

14. Create a new certificate for the device. Click the  icon.

    ⇨ The *“Certificate Settings”* dialog opens.

15. Define the certificate parameters and click *“OK”* to close the dialog.

    ⇨ The certificate is created on the controller.

16. Click the  button and save the certificate to the local file directory of the OPC UA Server, in the folder `certs`.

    ⇨ Now when you restart the OPC UA Server, it will recognize the client certificate. The the server sends its certificate to the client. In the following steps, this certificate will be made "trusted" to the client.

17. To do this, in the *“Security Screen”* view, on the *“Devices”* tab, click the *“Certificates in Quarantine”* folder in the left area.

    ⇨ The certificate is displayed in the right area.

18. Drag this certificate to the *“Trusted Certificates”* folder.

    ⇨ Now the server certificate is "trusted" by the client.

19. Now when you connect to the controller and the application starts, the data source variables of the OPC UA Client can be exchanged with the OPC UA Server via the encrypted connection.

See also

- *Chapter 1.4.1.20.2.5.5 “Tab 'Communication' via OPC UA Server” on page 834*

1.4.1.9.5 Subordinate safety controller

If a safety controller is below the standard controller, then the communication with the development system and the data exchange run via the standard controller. The communication links of the safety controller can interrupted the execution of commands that affect the standard controller. You find a notice about this for each these command.

Possible interruptions

- Temporary interruption: During the execution of the command (for example: download), the connections with the safety controller are interrupted first and then are automatically available again afterwards. If the interruption time is too long, then safety-oriented reactions can occur in the output devices and connected network variable receiver safety controllers. Then in the safety controller, the corresponding communication errors must be acknowledged (if not done automatically) in order to end the safety-oriented reactions. This affects the connection to their field devices and network variable receiver connections to other sender safety controllers. In the case of a connected safety controller with network variable senders, the communication errors must be acknowledged in the other safety controllers.
- Permanent interruption: The execution of commands (for example: delete) leads to an interruption that is ended again by another action (for example: download). As a result of the interruption, safety-oriented reactions can occur in the output devices and connected network variable receiver safety controllers. After ending the interruption, the corresponding communication errors must be acknowledged in the safety controller (if not done automatically) in order to end the safety-oriented reactions.

For a subordinate safety controller, the routing runs via *“<Name of SafetyApp>_Mapping”*. In some cases, it can happen that the user can see this application in the device tree.

CAUTION!
No commands may be executed in the application *“<Name of SafetyApp>_Mapping”*.

- *Chapter 1.4.1.9 “Working with Controller Networks” on page 352*

1.4.1.10 Downloading an Application to the PLC

In order to transfer your application to the PLC, the program has to be compiled without any errors and the connection settings for the PLC have to be set.

If the communication with the controller is encrypted and/or restricted to specific users, then you need the respective certificates and permissions. See here:

- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*
- *Chapter 1.4.1.8.17 “Encrypting an application” on page 294*

You can edit the basic security policy for communication with the device in a dialog on the “Communication Settings” tab of the device editor. See here:

- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*

When these requirements are fulfilled, the application is downloaded to the PLC at login.

1.4.1.10.1 Configuring the Connection to the PLC

The connection to the controller is established by means of a gateway. This gateway can be your development computer or another network computer connected to the controller. The *“Communication Settings”* dialog is available for configuring the connection path. This dialog opens automatically when you attempt to log in, but the communication settings have not been configured yet.

> *If the communication with the controller is encrypted and secured by means of user management, then you need a corresponding certificate and credentials to establish the connection to the controller. In this case, see the relevant instructions on the "Encrypting communication and Changing Security Settings" help page.*

Requirement: The project can be compiled without any errors. A programmable logic controller (PLC) is inserted in the device tree. The use of a user management is required for the device, but it is not enabled.

1. In the device tree, select the PLC and click *“Project ➔ Edit Object”*.
   ⇨ The PLC opens in the editor.
2. Click the *“Communication Settings”* tab.
3. On the menu bar, click *“Scan Network”*.
   ⇨ The *“Select Device”* dialog opens. All available devices in the network are shown on the left side.
4. Select the desired device and click *“OK”*.
   ⇨ A dialog prompt is displayed with the notice that a user management is required for the device, but it is not enabled yet. You are prompted to enable the user management if you want. The notice is displayed that in this case you have to create a new administrator account and then log in as this user.
5. Click *“Yes”* to close the dialog prompt.
   ⇨ The *“Add Device User”* dialog opens to create an initial device administrator.
6. Define the credentials (*“Name”* and *“Password”*) for the device administrator. Select the *“Password can be changed by the user”* option.

   > **NOTICE!**
   > Remember the seriousness of the password: From within the development system, there is no way to access the controller again if you forget the password.

   Click *“OK”* to close the dialog.
   ⇨ The *“Device User Logon”* dialog opens.
7. Enter the credentials for the device administrator which you defined in the previous step.
   ⇨ The connection path for the PLC is set.

> *You can reset the communication settings view to the original view in the CODESYS options of the device editor.*

See also

- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*
- *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*
- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*

1.4.1.10.2 Encrypting Communication, Changing Security Settings

NOTICE!

Recommendations for data protection

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running. Whenever possible, avoid exposing the PLC and control networks to open networks and the Internet. Use additional data link layers for protection, such as a VPN for remote access. Install firewall mechanisms. Restrict access to authorized persons. Use high-strength passwords. Change any default passwords regularly before and after commissioning.

Use the security features supported by CODESYS and the respective controller, such as encryption of communication with the controller and intentionally restricted user access.

Communication with the device can be protected by means of encryption and user management on the device. You can change the current security preset on the *“Communication Settings”* tab of the device editor.

Establishing a connection to the controller, logging in, installing a trusted certificate for encrypted communication

☑ Requirement: Encrypted communication with the controller and user management are enforced on the controller. However, an individual password does not exist yet. A certificate has not been installed on your computer and the connection to the controller has not been configured yet.

1. In the device tree, double-click the controller.

   ⇨ The device editor opens.

2. Click the *“Communication Settings”* tab.
3. Click *“Scan Network”*.

4. Select a controller.

   ⇨ A dialog opens, informing you that the certificate of the device does not have a trusted signature for communication. You are prompted whether or not to install this certificate as trusted in the local "Controller Certificates" store on your computer, or accept a session only for this one.

   NOTICE!
   A controller certificate installed in this way is valid for only 30 days. This gives you time for the following long-term solutions:

   – Creation of an additional self-signed certificate with a longer term (for example, 365 days). You can do this on the security screen if you have installed the CODESYS Security Agent, even if a certificate already exists. Using the PLC shell of the device editor is not a convenient workaround.
     See below: "Configuring encrypted communication with a controller certificate with a more long-term validity period"
   – Importing a CA-signed certificate. This is currently only possible via the PLC shell commands of the runtime. Therefore, we recommend to use self-signed certificates first.

5. If you want to install the certificate, then select the first option and click *"OK"* to confirm the dialog prompt.

   ⇨ The certificate is listed as trusted. After accepting the self-signed certificate for the first time, you can establish an encrypted connection with the controller again and again without further prompts.

   A dialog prompt is displayed with the notice that a user management is required for the device, but it is not enabled yet. You are prompted to enable the user management if you want. The notice is displayed that in this case you have to create a new administrator account and then log in as this user.

6. Click *"Yes"* to close the dialog prompt.

   ⇨ The *"Add Device User"* dialog opens to create an initial device administrator.

7. Create a device user in order to edit the user management as this user. In this case, only the *"Administrator"* group is available. Specify a *"Name"* and *"Password"* for the user. The password strength is displayed. Note also the set options regarding a password change. By default, the password can be changed by the user at any time. Click *"OK"* to confirm.

   ⇨ The *"Device User Logon"* dialog opens.

8. Enter the credentials for the device administrator which you defined in the previous step.

   ⇨ You are logged in on the controller. On the *"Users and Groups"* tab, you can use the button to switch to synchronized mode. The device user management is displayed there and you can edit it.

   After you click *"OK"* to confirm, the device user management is displayed in the editor view. It contains the user of the *"Administrator"* group who you just defined. The name of this user is also displayed in the taskbar of the window as *"Device User"*.

9. All saved controller certificates (from Step 5) are stored in the local Windows Certificate Store on your computer. You can access this memory by means of the *"Execute"*, `certmgr.msc` command.

   ⇨ All registered certificates for encrypted communication with controllers are listed here in *"Controller Certificates"*.

Configuring a controller certificate with a more long-term validity period

Requirement: The CODESYS Security Agent add-on product is installed. You want to replace the temporary certificate (as described above) acquired the first time you connected to the protected controller with a certificate with a longer validity period.

for encrypted communication by means of CODESYS Security Agent (recommended)

In this case, the *"Security Screen"* view provides an additional tab: *"Devices"*. This allows for the simple configuration of certificates for the encrypted communication with controllers. For operation, see the help for CODESYS Security Agent: "Encrypted Communication with Devices via Controller Certificates".

Installing a controller certificate for encrypted communication via the PLC shell of the device editor

Choose this less convenient method when the CODESYS Security Agent is unavailable to you. In this case, you can set up a certificate with a more long-term validity period for communication encryption on the *"PLC Shell"* tab of the device editor.

☑ Requirement: You are connected to the controller.

1. At first, you check if a qualified certificate is already on the controller. If no certificate is available, then you create a new certificate.

   Open the device editor by double-clicking the controller in the device tree, and select the *"PLC Shell"* tab.

   ⇨ The tab appears with a blank display window. Below that is a command line.

2. Type the following command in the command line: `cert-getapplist`.

   ⇨ All used certificates are listed. The list includes information about the runtime component and whether or not the certificate is available.

3. If a certificate still does not exist for the component `CmpSecureChannel`, then type the following command in the input line:

   `cert-genselfsigned <number of the component in the applist>`

4. Click the *"Log"* tab and then the refresh button (🔄).

   ⇨ The display shows whether or not the certificate was generated successfully.

5. Change back again to the *"PLC Shell"* tab and type the command `cert-getapplist`.

   ⇨ The new certificate for the component `CmpSecureChannel` is displayed.

6. In the next two steps, activate encrypted communication in the security screen of CODESYS.

7. Open the *"Security Screen"* by double-clicking 🛡 in the status bar.

8. On the *"User"* tab, select the *"Enforce encrypted communication"* option in the *"Security Level"* group.

   ⇨ The communication to all controllers is encrypted. If there is not a certificate on a controller, then you cannot log in to it.

   The connecting line between the development system, the gateway, and the controller is displayed in yellow on the *"Communication Settings"* tab of the device editor of the controller.

9. As an alternative to the *"Enforce encrypted communication"* option which applies to all controllers, you can also define encrypted communication for specific controllers only. To do this, select the *"Communication Settings"* tab in the editor of the respective controller. Then click *"Encrypted Communication"* in the *"Device"* list box.

   ⇨ The communication with this controller is encrypted. If there is not a certificate on the controller, then you cannot log in to it.

   The connecting line between the development system, the gateway, and the controller is displayed in yellow on the *"Communication Settings"* tab of the device editor of the controller.

10. When you log in to the controller for the first time, a dialog opens with information that the certificate of the controller is not signed by a trustworthy authority. In addition, the dialog displays information about the certificate and prompts for you to install it as a trustworthy certificate in the local store in the *"Controller Certificates"* folder.

    When you confirm the dialog, the certificate is installed in the local store and you are logged in to the controller.

    In the future, communication with the controller will be encrypted automatically with this control certificate.

11. To increase security for key exchange for controllers < V3.5 13.0, you can generate Diffie-Hellman parameters on the controller. To do this, type the command `cert-gendhparams` in the input line.

    This is no longer required for controllers >= V3.5.13.0.

> **NOTICE!**
> Caution: Generating the Diffie-Hellman parameters can last for several minutes or even several hours. However, this process must be executed only one time for each controller. The Diffie-Hellman parameters increase security for key exchange and for future attacks against encrypted data recording.

Changing the communication policy (encryption, user management)

☑ Requirement: The connection to the device is established.

1. In the device tree, double-click the controller.

   ⇨ The device editor opens.

2. Click the *"Communication Settings"* tab.
3. Open the *"Device"* menu in the header of the editor. Click *"Change Communication Policy"*.

   ⇨ The *"Change Communication Policy"* dialog opens.

4. In the upper part of the dialog, you can toggle between the *"Optional encryption"*, *"Enforced encryption"*, and *"No encryption"* settings.
5. In the lower part of the dialog, you can toggle between the *"Optional user management"* and *"Enforced user management"* settings.

Enabling and disabling enforced encrypted communication

☑ Requirement: The device supports encrypted communication.

1. In the device tree, double-click the controller.

   ⇨ The device editor opens.

2. Click the *"Communication Settings"* tab.

3. Open the *"Device"* menu in the header of the editor. Click *"Encrypted Communication"*. The status toggles between enabled and disabled.

   ⇨ If the *"Encrypted communication"* option is selected, then the connection line between the development system, the gateway, and the device is highlighted in the editor in bold and in color in the graphical representation.

See also

- *Chapter 1.4.1.10.3 "Handling of Device User Management" on page 385*
- *Chapter 1.4.1.8.17 "Encrypting an application" on page 294*
- *"Encryption with certificates" on page 198*

1.4.1.10.3 Handling of Device User Management

NOTICE!

Recommendations for data protection

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running. Whenever possible, avoid exposing the PLC and control networks to open networks and the Internet. Use additional data link layers for protection, such as a VPN for remote access. Install firewall mechanisms. Restrict access to authorized persons. Use high-strength passwords. Change any default passwords regularly before and after commissioning.

Use the security features supported by CODESYS and the respective controller, such as encryption of communication with the controller and intentionally restricted user access.

For devices that support a device user management, the device editor includes the *"Users and Groups"* tab and the *"Access Rights"*. When offered by the device, you can view the user management for the device here as well as edit it in synchronization mode (not in online mode). Here, you can grant or deny specific permissions on the controller to the defined user groups.

The device user management can already be set up in the device description.

Note the commands in the "Online ➔ Security" menu. You can easily add, edit, or remove a user account on the controller where you are currently logged in.

In order for the "Access Rights" tab to be available in the device editor, the corresponding CODESYS option must be selected in the device editor and unlocked in the device description. If the device editor is not available, then contact the manufacturer of the controller.

In order to grant access rights to a user group, first the users and user groups have to be configured on the *"Users and Groups"* tab of the device editor. User management first has to be set up on the controller before access rights can be configured on it. In case the user management of a device is not enabled yet, it can be enabled in the following way: Either by switching to the synchronized mode on the *"Users and Groups"* tab, or by adding a new user by means of the command *"Online ➔ Security ➔ Add Device User"*.

General information about device user management

For the CODESYS Control devices, a user management is enforced by default.

Access rights can be granted to groups only, not individual users. Therefore every user has to be a member of a group.

Access rights can be granted for the following actions which are executed on the individual objects of the controller:

- Add/Remove
- Modify
- View
- Execute

An object on the controller is usually assigned to just one controller component.

Each object can use all of the listed actions, but usually only the permissions for the following actions are needed on an object:

- *"View"*
- *"Modify"*

The objects are organized in a tree structure. There are two root objects for the two kinds of objects:

- *"Runtime objects ➔ Device"*: In these objects, all objects are managed that have online access in the controller and therefore have to control the access rights.
- *"File system objects ➔ /"*: In these objects, the permissions can be granted to folders of the current execution directory of the controller.

The child objects inherit the access rights from the root object (also *"Device"* or "*"/"*"). If a permission of a user group is denied or explicitly granted to a parent object, then this affects all child objects.

A single permission can be explicitly granted or denied (green plus sign or red minus sign), or remain "neutral" (light gray character). Neutral means that the permission has been neither explicitly granted nor denied. In this case, the permission of the parent object applies.

If no permission has been explicitly granted or denied in the entire hierarchy of the object, then it is by definition denied. As a result, all permissions are initially denied (exception: the access right for the *"View"* action). Initially, this permission is explicitly granted for every user group both on the *"Device"* runtime object as well as on the "*"/"*" file system object. This allows read access to all objects, unless it is explicitly denied in child objects.

For an overview table for the objects, see the "Tab 'Access Rights'" chapter.

See the following instructions for handling the editor for the device user management:

First-time login on the controller in order to edit or view its user management

Requirement: The connection to the controller is configured. The controller supports a device user management, but one is not active yet.

1. Double-click the controller device object in the device tree.
   ⇨ The device editor opens.
2. Click the *"Users and Groups"* tab.
3. Click  .
   ⇨ A dialog opens prompting whether the device user management should be activated.
4. Click *"Yes"* to acknowledge the dialog prompt.
   ⇨ The *"Add Device User"* dialog opens.

5. Now create a device user in order to edit the user management as this user. In this case, only the *"Administrator"* group is available. Specify a *"Name"* and *"Password"* for the user. The password strength is displayed. Note also the set options regarding a password change. By default, the password can be changed by the user at any time. Click *"OK"* to confirm.

   ⇨ The *"Device User Logon"* dialog opens.

6. Specify a *"User name"* and *"Password"* for the user who you just defined.

   ⇨ After you click *"OK"* to confirm, the device user management is displayed in the editor view. It contains the user of the administrator group who you just defined. The name of this user is also displayed in the taskbar of the window as *"Device User"*.

Setting up a new user in the user management of the controller

Requirement: The controller has a device user management. You have the corresponding access data.

1. Double-click the controller device object in the device tree.

   ⇨ The device editor opens.

2. Click the *"Users and Groups"* tab.

3. Click (Synchronization) to load the user management configuration from the controller to the editor. If you are not logged in to the device yet, then the *"Device User Logon"* dialog opens for entering the user name and password.

   ⇨ The user management configuration of the device is shown in the editor.

4. In the *"Users"* view, click *"Add"*.

   ⇨ The *"Add User"* dialog opens.

5. Specify the name of the new user and assign the user to a group. This counts as the user's minimum required default group. The user can be assigned to other groups later. Define and confirm a *"Password"* for the user. Define whether the user can change the password and whether the user has to change the password at the first login. Click *"OK"* to confirm.

   ⇨ The new user appears in the *"Users"* view as a new node and in the *"Groups"* view as a new subentry of the selected default group.

Changing of access rights to controller objects in the user management of the controller

Requirement: The controller has a device user management. You have the corresponding access data.

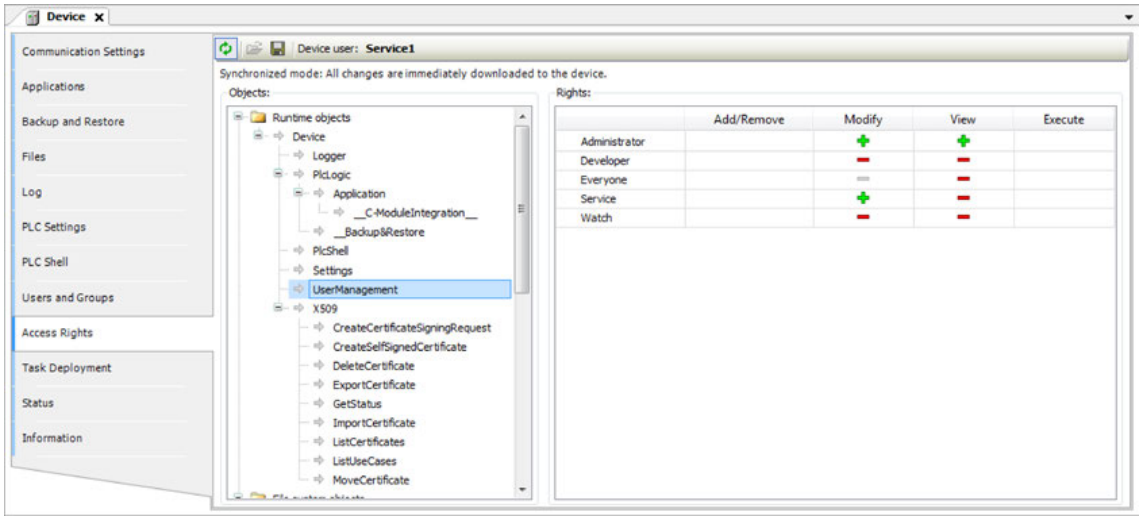
1. Double-click the controller device object in the device tree.

   ⇨ The device editor opens.

2. Click the *"Access Rights"* tab.

3. Click (Synchronization) to load the rights management configuration from the controller to the editor. If you are not logged in to the device yet, then the *"Device User Logon"* dialog opens for entering the access data.

   ⇨ The access rights management configuration of the device is shown in the editor.



4. Select the object whose access right you want to change to the left in the object tree.

   ⇨ In the *"Rights"* view, a table shows the access rights to this object for all configured user groups.

5. Double-click the right in the table that you want to change.

   ⇨ If the object has child objects, then a dialog prompts whether you want to modify the permissions for the child objects.

6. Click *"Yes"* or *"No"* to close the prompt.

   ⇨ The permissions are switched from "allowed" + to "not allowed" −, or the other way around. The symbol in the table cell changes accordingly. Explicitly set permissions appear in the table as green or red symbols. Rights that are inherited from a parent object appear as gray symbols.

Transferring and enabling a saved user management in offline mode from a DUM2 file to a controller

In V3.5 SP16 and higher, a file (.dum2) to be encrypted with a password is used for exporting a user management.*

1. Double-click the controller device object in the device tree.

   ⇨ The device editor opens.

2. Click the *"Users and Groups"* tab.

3. Click .

   ⇨ The dialog for selecting a file from the local file system opens.

4. Select the file (`<file name>.dum2`) with the desired user management from the local file system and click *"Open"* to confirm.

   ⇨ The *"Enter Password"* dialog opens.

5. Specify the password that was assigned when the user management file was exported (possible by means of the button).

   CAUTION: The import of a device user management by means of a `*.dum2` file completely overwrites the existing user management on the device. In order to log in to the device again afterwards, you need authentication data from the recently imported user management.

   ⇨ When the password is entered correctly, the configuration from the downloaded user management file is now displayed in the editor view.

6. Edit the configuration however you like. For example, change the user password or add a new user.

   ⇨ Every change is downloaded immediately to the device.

See also

- *Chapter 1.4.1.20.2.8.14 “Tab 'Access Rights'” on page 863*
- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*
- *Chapter 1.4.1.20.2.8.13 “Tab 'Users and Groups'” on page 860*
- *Chapter 1.4.1.7.1 “Configuring Devices and I/O Mapping” on page 213*
- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*

1.4.1.10.4 Generating Application Code

The application code is the machine code that a PLC executes when you start an application.

CODESYS automatically generates the application code from the source code that was written in the development system. This is done automatically before downloading the application to the PLC. Before the application code is generated, a test is performed for checking the allocations, the data types, and the availability of libraries. Moreover, the memory addresses are allocated when the application code is generated.

You can click *“Build ➔ Generate Code”* to execute this command explicitly. This is useful for detecting any errors in your source code, even when the PLC is not connected yet. The errors are output in the message view in the "Build" category.

NOTICE!
If you have encrypted the application, then consider the following information: If a (new) boot application is generated on request after an online change, then the boot application is formed in the RAM with the current code that is not encrypted.

Explicitly generating the application code

Requirement: The application can be compiled without any errors.

▷ Click *“Build ➔ Generate Code”*.

⇨ The application code is generated. Detailed information about memory allocation is output in the message view.

See also

- *Chapter 1.4.1.10.6 “Generating boot applications” on page 391*
- *Chapter 1.4.1.20.3.5.1 “Command 'Generate Code'” on page 1021*

Messages when generating the application code

When you generate the application code, CODESYS outputs information about memory allocation in the message view. Gaps form in the memory because reallocation is only for new and changed POUs and variables due to the incremental memory build. Online changes have the same effect. This fragmentation reduces the amount of available memory. However, you can completely reallocate the memory by clicking *“Clean”* and therefore increase the amount of free memory.

Syntax errors and bugs that CODESYS detects during the code generation and memory allocation are also output in the message view (*“Build”* category).

Output information about memory allocation:

- *“ Size of the generated code”* (in bytes): Sum of all code segments
- *“Size of global data”* (in bytes): Total memory used by the global variables. Inputs and outputs are not included unless inputs or outputs are mapped in the area of the global variables.
- *“Total allocated memory size for code and data”* (in bytes): The total allocated memory is composed of the already used memory areas plus the reserved, not yet used memory for incremental builds and online changes. After the first build, the already used memory is approximately equal to the highest used address (see below). The largest contiguous memory gap (see below) still corresponds approximately to the difference to the total allocated memory. However, as the number of incremental builds and online changes increases, the number of memory gaps also increases, and the largest contiguous memory gap becomes smaller.
- *“Memory area <n>”*: Contents of the individual reserved memory areas

  Background: It depends on the PLC which data and code is stored in which memory areas. For example, code and data are located in the same area on the CODESYS Control Win V3. For the addresses `%I`, `%M`, and `%Q`, memory is always reserved, even when a variable is not assigned to an address. After cleaning the application, the memory is reallocated completely. In this case, small gaps could result from the predefined alignment (normally 8). Larger gaps result from changing a date without cleaning, for example by increasing an array area. In this case, only the affected POUs are recompiled. Furthermore, in the case of an online change, the memory is used only for new variables and new code. Memory that was previously reserved by deleted variables and code is made available again. As a result, memory fragmentation can occur after many incremental builds and online changes. This creates many small gaps that might not be usable at all in some cases. To clarify how much memory is safely available, the "largest contiguous memory gap" of the memory area is output during code generation.
  - *“highest used address”* (Byte) : This is the highest reserved address in the entire allocated memory area. During the first build after a "Clean" operation, the memory addresses are output to variables in ascending order, taking into consideration the alignment (usually 8 bytes). As a result, the highest address used at this time corresponds approximately to the amount of memory used. The rest of the allocated memory area is still completely available for incremental builds and online changes.
  - *“Largest contiguous memory gap”* (in bytes): This is the memory size that is available for backup.

    Resulting gaps in the allocated memory are reused whenever possible for other changes. When, for example, a global variable of type `Byte` is added, it is placed in the first free byte of the memory. Even just a small gap is enough for this. However, an FB instance, a variable of the type structure or array, or the code for a POU has to be stored contiguously and therefore occupies more memory accordingly. As a result, they can be allocated only to the largest contiguous memory area. This is why during code generation the "largest contiguous memory gap" that is safely available is output (in bytes), as well as its percentage of the total memory.

Note the options for generating applications.

See also

- *Chapter 1.4.1.20.4.10.9 “Dialog 'Properties - Application Build Options'” on page 1162*

Encrypting the application code

See also

- *Chapter 1.4.1.8.17 “Encrypting an application” on page 294*

1.4.1.10.5 Downloading the application code, logging in, and starting the PLC

In order to download the source code of your application to the PLC, you must log in to the PLC with application. If there are several applications in the project, then you must switch explicitly to the correct application first.

When you download an application to the PLC, CODESYS performs the following checks:

The list of applications on the PLC is compared with the applications available in the project. If they do not match, then you are prompted to download the application that is not on the PLC yet or delete existing applications.

For "externally implemented" blocks in the application to be downloaded, CODESYS checks whether these are available on the PLC. If they are not available, then the message "unresolved reference(s)" is printed to a dialog prompt and to the message view. Then CODESYS compares the parameters (variables) of the blocks in the application to be downloaded and the parameters of the same-named blocks in the application that exists on the PLC (signature check). If there are any discrepancies, then the message "invalid signature(s)" is printed to a dialog prompt and to the message view.

If the "Download Application Info" check box is selected in the application properties, then additional information about the application contents are downloaded to the PLC.

if multiple applications exist for the same device, then notice that the *“I/O Mapping”* dialog contains the definition for which of the applications is used for for the I/O mapping of the device.

See also

- *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*

Transferring an application and starting the program

Requirement: The application contains no errors and the communication settings of the PLC are correct. The application does not exist yet on the PLC: The application and the communication with the controller are not encrypted.

1. Select the required application in the device tree. Skip to Step 3 if you have only one application.
2. Click *“Set Active Application”*.
   ⇨ The application name appears in bold typeface.
3. Click *“Online ➔ Login”*.
   ⇨ A dialog prompts you whether the application should be created on the PLC.
4. Click *“Yes”* to confirm.
   ⇨ The application is downloaded to the PLC.
5. Click *“Debug ➔ Start”* or press the *[F5]* key.
   ⇨ The application is running on the controller.

1.4.1.10.6 Generating boot applications

A boot application is the application that is started automatically when the controller is switched on or started. For this to happen, the application on the controller must exist as a file `<application name>.app`.

For each application that is running on the controller, a boot application can also be saved there.

By default, CODESYS generates the boot application automatically when an application is downloaded and transfers them to the PLC. The defaults for generating automatically are located in the *“Boot application”* category of the application *“Properties”*. When logging in with a changed application, you are still prompted whether or not to generate a new boot application.

In addition, you can create a (new) boot application at any time in online mode by clicking *“Online ➔ Create boot application”*.

You can create and save a local copy of a boot application in offline mode as well. Then, you can copy this application to the controller with external tools. In this way, you transfer an application to the controller, even when there is no connection to CODESYS.

Generating boot applications on the controller automatically and explicitly

Requirement: Offline mode; the application is compiled without errors. The connection to the controller is configured and the controller is running. The application is active. The following steps demonstrate the options:

1. Click *"Online ➔ Login"*.

   ⇨ The boot application file `<application name>.app` is created on the controller with the checksum of the boot application `<application name>.crc`.

2. Click *"Online ➔ Create Boot Application"* explicitly.

   ⇨ The files on the controller are replaced by new files.

3. Log out.
4. Change the application. Log back in to the controller.

   ⇨ You are prompted whether an online change should be performed. You see the *"Update boot application"* check box in the same dialog box. This is cleared by default, but this can be changed in the *"Boot Application"* category of the application *"Properties"*.

5. Keep the check box cleared and continue login.

   ⇨ A new boot application file is not created.

6. Log out. Close the project. Stop the controller. Restart the controller.

   ⇨ The boot application that was created above is running on the controller.

You can save the encrypted boot application on the controller. These settings are defined in the "Application Build Options" category of the application "Properties".

See also

- *Chapter 1.4.1.20.4.10.9 "Dialog 'Properties - Application Build Options'" on page 1162*
- *Chapter 1.4.1.20.4.10.2 "Dialog 'Properties' - 'Boot Application'" on page 1158*
- *Chapter 1.4.1.13.1 "Executing the online change" on page 439*
- *Chapter 1.4.1.20.3.6.4 "Command 'Create Boot Application'" on page 1032*

Creating boot applications in offline mode

Requirement: Offline mode; the application is compiled without errors. You want to generate a boot application for an application and save it in the file directory for copying it later to the controller by using external tools (without CODESYS).

1. Click *"Online ➔ Create Boot Application"*.

   ⇨ A dialog box opens for specifying a save location in the local file system.

2. Click a save path and then click *"Save"*.
3. If the application has changed since the last boot application was generated, then you are prompted to use a new code for the boot application. In this case, click *"Yes"*.

   ⇨ The *"Save as"* dialog box opens.

4. Select a directory and click *“Save”*.
   ⇨ The boot application file <application name>.app is created in the given path.

   You are prompted whether or not the build information for the boot application is saved.
5. Click *“Yes”*.
   ⇨ The build information is saved to the project directory as a file named `<application name>.compileinfo`. It is a requirement for a possible online change the next time the application is updated. Please note: Clicking *“Build ➔ Clean”* deletes this file.

See also

- *Chapter 1.4.1.20.3.6.4 “Command 'Create Boot Application'” on page 1032*

When using CODESYS Control Win V3

When using a CODESYS Control Win V3, the application name must also be included in the configuration file (`*.cfg`).

```
[CmpApp]
 Application.1=MyApplication
```

1.4.1.10.7 Downloading source code to and from the PLC

CODESYS provides the capability of loading project source code to a PLC as a project archive. You can then transfer this project archive back to the development system from the PLC as needed.

Requirement: The connection settings are configured for the affected controllers.

Downloading source code to the PLC

1. Choose the command *“File ➔ Source Download”*.
   ⇨ The dialog box *“Select Device”* opens.
2. Select the controller to receive the source code. Click *“OK”*.
   ⇨ CODESYS writes the archive file `Archiv.prj` to the controller.

By choosing the command “Online ➔ Source Download to Connected Device”, you can load the source code directly to the connected device.

See also

- *Chapter 1.4.1.20.3.1.11 “Command 'Source Download'” on page 963*
- *Chapter 1.4.1.20.3.6.7 “Command 'Source Download to Connected Device'” on page 1035*
- *Chapter 1.4.1.20.4.11.5 “Dialog 'Project Settings' – 'Source Download'” on page 1174*

Loading source code from the PLC

1. Choose the command *“File ➔ Source Upload”*.
   ⇨ The dialog box *“Select Device”* opens.
2. Select the controller to send the source code. Click *“OK”*.
   ⇨ The *“Extract Project”* dialog box opens.
3. Select the destination directory where you want to extract the project archive. Click *“Extract”*.
   ⇨ CODESYS extracts the project archive to the directory.
4. Then you are prompted to open the project archive. Click *“Yes”*.
   ⇨ The project opens.

See also

- *Chapter 1.4.1.20.3.1.10 “Command 'Source Upload'” on page 962*

1.4.1.11 Testing and Debugging

CODESYS provides various options for testing your application and detecting errors. You can start your application in simulation mode, even without connecting any hardware. Using breakpoints and stepping commands, you can examine specific parts of a program. By writing values to variables, you can influence the running program.

Commands are provided that reset your application in various different ways, from resetting only non-persistent variables to completely resetting the controller to factory settings.

See also

- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*
- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*
- *Chapter 1.4.1.11.5 “Resetting applications” on page 404*

1.4.1.11.1 Testing in simulation mode

Use simulation mode for testing and debugging your program when you do not have a physical target device. In this mode, the application is started on a simulated device.

The command is available only when you are logged out.

Requirement: You program contains no errors (compiler error messages or compile errors) and you are not logged in.

1. Activate simulation mode as follows:
   - Click *“Online ➔ Simulation”*, or
   - Right-click the controller in the device tree and click *“Simulation”*.
   ⇨ The name of the controller in the device tree is displayed in italics. In the status line, "Simulation" appears highlighted in red. The *“Simulation”* command is selected in the main menu.
2. Click *“Online ➔ Login”*.
3. When logging in with the active application, you will be prompted whether the application "Sim.<device name>.<application name>" should be created and loaded. Click *“Yes”* to confirm.
   ⇨ The application is logged onto the PLC.

4. Now you can check and correct the program flow with the commands provided in the main menu in *“Debug”*.
5. Log out from the controller and end the simulation mode.

See also

- *Chapter 1.4.1.20.3.6.23 “Command 'Simulation” on page 1044*

Limitations

- The focus of the simulation mode is testing and debugging your program of the PLC. That means the functionality of the simulated PLC is limited. Keep in mind that some POUs have no function. They are not creating any compile or download errors, they will simple not work.
- Without an extra available “Virtual Commissioning” license the “Online mode” of the simulated PLC is limited to 2 hours. After 2 hours starting from the “Login”, the “Online mode” is automatically terminated and the PLC is logged out.
- It is not possible to create a “Boot Application” in the simulated PLC. Every “Login” starts with an empty simulated PLC and a download of the application is required.
- When logging in to a simulated PLC the first time a “Windows Security Alert” is displayed. Depending on the application, e.g. if any network communication is implemented, it might be necessary to allow the “Virtual AC500” to communicate on one or multiple network types.

1.4.1.11.2 Using Breakpoints

Breakpoints are commonly used for debugging programs. CODESYS supports breakpoints in all IEC editors.

You can set breakpoints at specific positions in the program to force an execution stop and to monitor variable values. You can set special data breakpoints to halt program execution when the value of a specific variable changes.

The halt at a breakpoint or data breakpoint can be linked to additional conditions. You can also redefine breakpoints and data breakpoints as execution points where specific code is executed instead of stopping the program.

The “Breakpoints” view provides an overview of all defined breakpoints. It also includes additional commands for processing batch changes to multiple breakpoints.

In the editor, the following symbols identify the status of a breakpoint or execution point:

- The breakpoint is enabled.
- The breakpoint is disabled.
- Breakpoint is set in another instance of the POU open in the editor.
- The program is halted at the breakpoint.
- The conditional breakpoint is enabled.
- The conditional breakpoint is disabled.
- The execution point is enabled.
- The execution point is disabled.
- The conditional execution point is enabled.
- The conditional execution point is disabled.
- The data breakpoint is enabled.
- The data breakpoint is disabled.
- Halt at data breakpoint
- The data execution point is enabled.
- The data execution point is disabled.
- Halt at data execution point

- The conditional data execution point is enabled.
- The conditional data breakpoint is enabled.

See also

- *Chapter 1.4.1.20.3.3.12 “Command 'Breakpoints'” on page 989*

Data breakpoints

The function of data breakpoints depends on the target system. Currently, data breakpoints are possible only with the CODESYS Control Win V3.

Program execution stops at a data breakpoint when the value of a particular variable or memory address changes. As with ordinary breakpoints, the halt can be linked to an additional condition, or specific code can be processed instead of the halt (converted to a data execution point).

You set a data breakpoint either by means of the *“New Data Breakpoint”* command in the *“Debug”* menu or by means of the *“New”* button in the *“Breakpoints”* view. You specify a qualified variable name or a memory address directly which is to be monitored for changes in its value.

Example

In the following sample code, the memory of the variable `iNumber` is overwritten unintentionally. However, a data breakpoint at the variable `iNumber` will detect when its value changes. The processing then stops with a corresponding message at the array access, which overwrites the variable value: `Idx` = `7`. See also below: "Setting a data breakpoint".

```
PROGRAM PLC_PRG
VAR
  Idx : INT;
  Ary : ARRAY[0..3] OF BYTE;
  iNumber : INT := 55;
END_VAR
FOR idx := 0 TO 6 DO
  Ary[idx] := 0;
END_FOR
```

Breakpoints in applications with multiple tasks

Basically, debugging is not possible for multiple tasks at the same time. While you are working on a task with breakpoints or stepping, breakpoints are ignored in other tasks.

If a POU containing a breakpoint is used by multiple tasks, then only the debug task is halted because it reaches the breakpoint first. All other tasks continue. The *“Call Stack”* dialog shows which task is currently halted.

If you need a breakpoint to affect only one specific task, then you can define this in the breakpoint properties.

Breakpoints operate separately for each application so that a "HALT ON BP" does not affect any other applications. This applies also to parent/child applications, even if the breakpoint is set in a block that is used by several applications and whose code is located only once on the PLC.

NOTICE!
The I/Os that are called by the debug task are not updated at a halt in the breakpoint, even if you select the *“Refresh I/Os in Stop”* check box in the PLC settings.

If the application stops at a breakpoint on the PLC, then an online change or download causes all tasks to halt which means the PLC will stop. In this case, CODESYS prompts you whether or not to continue with the login.

See also

- ↳ *Chapter 1.4.1.20.3.3.15 “Command 'Call Stack'” on page 993*

Setting a single breakpoint (example in ST editor)

Requirement: The application is in online mode and running. The operating mode is *“Debug”*.

1. In the editor, open a POU programmed in structured text (ST).
2. Place the cursor in the line where a breakpoint will be set.
3. Click *“Debug ➔ Toggle Breakpoint”* or press *[F9]*.
   ⇨ The line is marked in red and identified by the "breakpoint enabled" symbol (). If the program is halted at the breakpoint, then the line is marked by the "stop at breakpoint" symbol (). The processing of the program is stopped. This is identified in the status line by the `HALT ON BP` status highlighted in red.
4. Click *“Debug ➔ Start”* or press *[F5]*.
   ⇨ The program continues.
5. Set more breakpoints and check the variable values at the break position.
6. Place the cursor in the line where a breakpoint should be removed.
7. Click *“Debug ➔ Toggle Breakpoint”* or press *[F9]*.
   ⇨ The marking disappears. The breakpoint is deleted.

See also

- ↳ *Chapter 1.4.1.20.3.7.9 “Command 'Toggle Breakpoint'” on page 1050*

Defining a breakpoint condition (example in ST editor)

1. In the editor, open a POU programmed in structured text (ST).
2. Place the cursor in the line where a breakpoint will be set.
3. Click *“Debug ➔ New Breakpoint”*.
   ⇨ The *“Breakpoint Properties”* dialog opens.
4. Click the *“Condition”* tab.
5. Click *“Break when the hit count is a multiple of”* in the *“Hit Count”* list box.
   Specify the value "5" in the field to the right.
6. In addition, you can define a Boolean condition for when the breakpoint should be active. Select the *“Break, when true”* check box. Specify a Boolean variable in the text field to the right.
7. Select the *“Enable breakpoint immediately”* check box.
8. Close the dialog.
   ⇨ The line is marked red and identified by the "conditional breakpoint enabled" symbol ()

Monitor the running program. As long as the Boolean variable for the condition is `FALSE`, the breakpoint condition is not fulfilled and the program continues to run. If you set the variable to `TRUE`, then the condition is fulfilled and the program halts at the breakpoint every 5th pass.

See also

- *Chapter 1.4.1.20.4.5 "Dialog 'Breakpoint Properties'" on page 1151*

Defining an execution point (example in ST editor)

1. In the editor, open a POU programmed in structured text (ST).
2. Place the cursor at the position for an execution point.
3. Click *"Debug ➔ New Breakpoint"*.
   ⇨ The *"Breakpoint Properties"* dialog opens.
4. Click the *"Execution Point Settings"* tab.
5. Select the *"Execution point"* option.
   In the *"Execute the following code"* field, type the following statement: `iCounter := iCounter + 1;`
   In the *"Print a message in the device log"* field, type the following text: `Execution point reached {iCounter}`
6. Close the dialog.

When the program reaches the execution point, it does not halt, but executes the code defined above. In addition. a message is issued to the device log.

See also

- *Chapter 1.4.1.20.4.5 "Dialog 'Breakpoint Properties'" on page 1151*

Setting a data breakpoint

Requirement: The application is in online mode and running.

1. Click *"View ➔ Breakpoints"*.
2. Click *"Debug ➔ New Data Breakpoint"*.
3. Click the … button in the *"New breakpoint"* dialog (*"Data"* tab.
4. In the *"Input assistant"* dialog (*"Watch Variables"* tab), select the variables for which the program should halt when changed.
   As an alternative, specify the qualified name of the variable on the *"Data"* tab directly in the input line. Example: `PLC_PRG.iNumber`. The exact number of bytes to be monitored is specified as the *"Size"*. A value that corresponds to the data type is set here automatically by default. You can also specify fewer bytes to be monitored.
5. In the *"Breakpoints"* view, select the line with the data breakpoint and click the button.
   ⇨ The line is marked and identified by the "Data breakpoint enabled" symbol (●). When the program reaches the data breakpoint (meaning when the value of the selected variables changes), the program processing halts. In the implementation part of the POU, the next line is identified by an arrow. This is identified in the status line by the `HALT ON BP` status highlighted in red.
6. Click *"Debug ➔ Start"* or press *[F5]*.
   ⇨ The program continues running and halts again when the value of the variables changes again.

See also

- *Chapter 1.4.1.20.4.5 "Dialog 'Breakpoint Properties'" on page 1151*

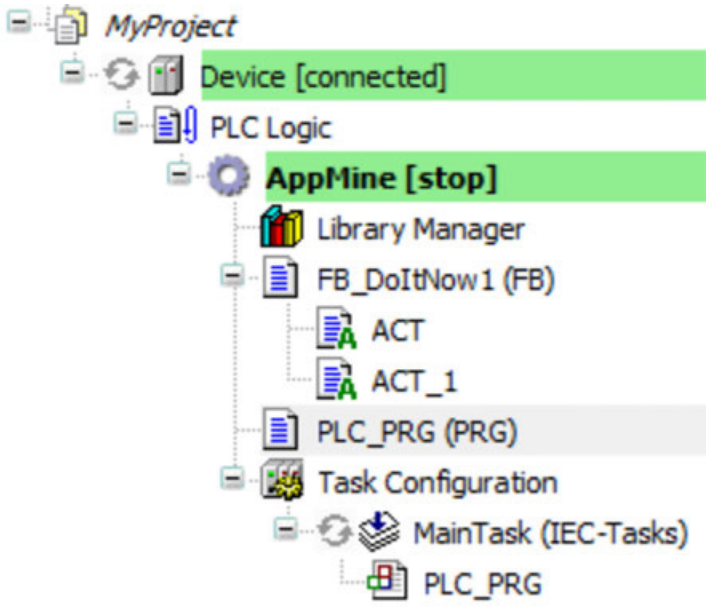
1.4.1.11.3 Stepping Through a Program

You can step through an application and navigate the code. This is useful to check the status of your code at runtime. You can examine the call process, track variable values, or locate errors.

Stepping commands are provided in the *"Debug"* menu for this purpose. The commands become available when you set breakpoints in online mode and then halt execution at a breakpoint: the application is in *"HALT ON BP"* state (debug mode). During debug mode, the current break position is highlighted in yellow and marked with the ➪ symbol in the text editors.

Switching to debug mode

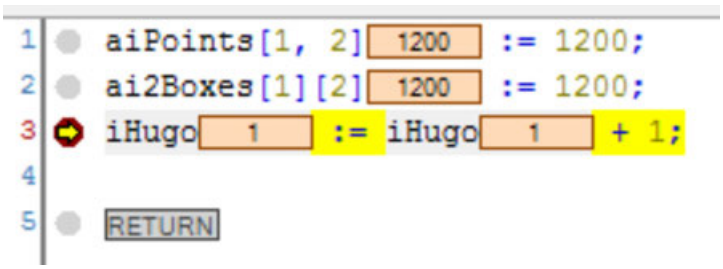
1. Download your application to a controller.
   ⇨ The application is highlighted in green. CODESYS and the editors of the POUs are in online mode.



2. In the POUs, set breakpoints at the locations in the code that you want to examine.
   ⇨ All breakpoints are listed in the *"Breakpoints"* view.
3. Start the application.
   ⇨ The application starts and the code is processed until the first breakpoint.

   Now the application is in debug mode. In the device tree, the application is labeled with *"[halt on breakpoint]"*. The status bar provides information about the operating state:

   The editor was opened at the current break position. The line of code with an active breakpoint where program execution was halted is highlighted in yellow and marked by the ➪ symbol. This statement highlighted in yellow has not been executed yet.



   Now you can select the various stepping commands or display the call tree.

Alternatively, you can first start the application and then set a breakpoint.

Behavior of the stepping command in the 'Debug' menu

- Command *"Step Over"*
  The statement at the breakpoint position is executed. Program execution halts before the next statement in the POU.
  If the statement contains a call (from a program, function block instance, function, method, or action), then the subordinate POU is processed completely in one step.
- Command *"Step Into"*
  The statement at the breakpoint position is executed. Program execution halts before the next statement.
  If the statement contains a call (from a program, function block instance, function, method, or action), then the program execution jumps to this subordinate POU. The first statement there is executed and the program execution halts before the next statement. The new current breakpoint position is then in the called POU.
- Command *"Step Out"*
  The command executes the POU from the current breakpoint position to the end of the POU and then jumps back to the calling POU. Program execution halts at the calling position (in the line with the call).
  If the current breakpoint position is in the main program, then the POU is run through to the end. Then the program execution jumps back to the beginning (to the program start at the first line of code in the POU) and halts there.
- Command *"Run to Cursor"*
  First set the cursor at any line of code and then execute the command. The program is executed from the current breakpoint position and halts at the current cursor position without executing the code of this line.
- Command *"Set Next Statement"*
  First set the cursor at any line of code (also before the current breakpoint position) and then execute the command. The statement marked with the cursor is executed next. All statements in between are ignored and skipped.
- Command *"Show Next Statement"*
  If you do not see the current breakpoint position, then execute the command. Then the window with the current breakpoint position comes into focus and the breakpoint position is visible.

Click *"View ➔ Call Stack"* to completely show the previous call tree for the breakpoint position currently reached in the program processing.

The "Call Stack" view shows the location of the block in the call structure of the program at all times, even before compiling the application.

See also

- *Chapter 1.4.1.20.3.7.11 "Command 'Step Into'" on page 1051*
- *Chapter 1.4.1.20.3.7.10 "Command 'Step Over'" on page 1050*
- *Chapter 1.4.1.20.3.7.12 "Command 'Step Out'" on page 1051*
- *Chapter 1.4.1.20.3.7.13 "Command 'Run to Cursor'" on page 1052*
- *Chapter 1.4.1.20.3.7.14 "Command 'Set Next Statement'" on page 1052*
- *Chapter 1.4.1.20.3.7.15 "Command 'Show Next Statement'" on page 1052*
- *Chapter 1.4.1.20.3.3.16 "Command 'Call tree'" on page 993*

1.4.1.11.4 Forcing and Writing of Variables

CAUTION!

Unusual changes to variable values in an application currently running on the controller can lead to undesired behavior of the controlled machinery.

Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled machinery, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

In CODESYS, variable values in the PLC can be changed in online mode. Here we make a distinction between forcing and writing a previously prepared value.

Writing is done with the *"Write Values"* command (*[Ctrl]+[F7]*) and sets the variable to the prepared value one time. In this way, the value can be overwritten again by the program at any time.

Forcing is done with the *"Force Values"* command (*[F7]*) and sets the prepared value permanently. For more information, see below.

The preparation of a value for forcing or writing is possible at different places:

- Declaration part: *"Prepared value"* field
- Implementation part of the FBD/LD/IL editor: inline monitoring field
- Watch view: *"Prepared value"* field

For instructions about this, see below. In the case that you want to prepare a value again for an already forced value, the *"Prepare Value"* dialog opens with options for handling the current force value.

Functionality of forcing

The prepared value is set to the respective variable at the beginning and end of a task cycle (or of a processing loop in the case of other task types).

The processing order in each cycle of a task is as follows:

1. Read the inputs

2. Force: Before the first program call, all prepared values are written to the variables by the runtime system, regardless of whether or not they are used by the task.

3. Process the IEC code

4. Force: After the last program call, all prepared values are written to the variables by the runtime system, regardless of whether or not they are used by the task.

5. Write the outputs

Note: It is possible that a forced variable temporarily gets a different value in the cycle while the code is being processed because the IEC code performs an assignment. Then the variable receives the forced value again only at the end of the cycle. The variable value can also be overwritten by the write access of a client to symbols of the application in mid-cycle. For this case, see the *"Access variables in sync with IEC tasks"* option in the *"Properties"* of the device object, or the *"Configure synchronization with IEC tasks"* setting in the symbol configuration. In this way, a PLC handler-supported synchronization of the write accesses by clients can be enabled with the task cycle.

NOTICE!

Forced values are marked with the F symbol. CODESYS does the forcing until the user lifts it explicitly by one of the following actions:

- Executing the *"Cancel forcing for all values"* command
- Lifting the force operation in the *"Prepare Value"* dialog
- Logging out of the application
  If forced variables still exist when logging out, then a dialog opens, prompting whether or not forcing should be lifted for all variables. If you respond by clicking *"No"*, then the forced values are applied again at the next login.

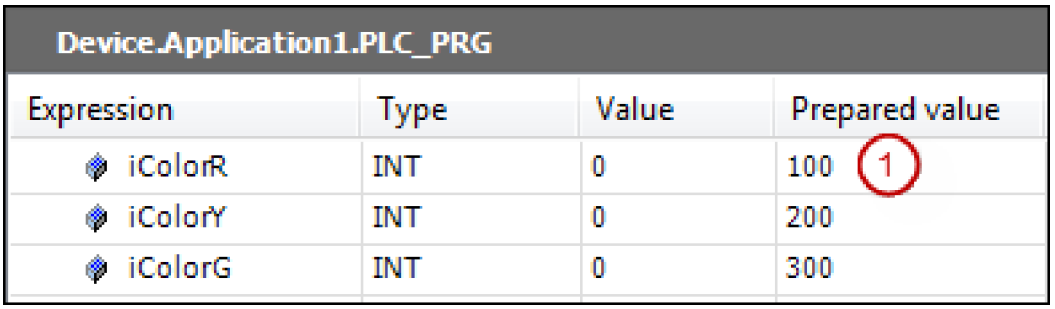
See also

- *Chapter 1.4.1.8.16 "Task Configuration" on page 292*
- *Chapter 1.4.1.20.4.10.19 "Dialog 'Properties' - 'Options'" on page 1169*
- *"Setting: Configure synchronization with IEC tasks" on page 932*

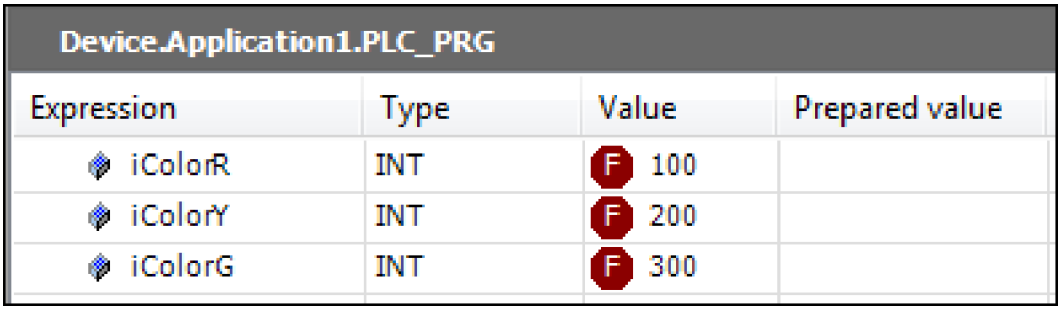
Forcing in the declaration part

Requirement: Your application includes a POU with declarations. The application is in online mode.

1. Open the POU in the editor by choosing the command *"Project ➔ Edit Object"*.
2. In the declaration part of the editor, double-click in column (1) *"Prepared Value"* of a variable.
   ⇨ The field can be edited and a value can be entered. When it is a Boolean value, you change the value by clicking in the field.



3. Perform Step 2 for other variables.
4. Click *"Debug ➔ Force Values"*.
   ⇨ The variable values are overwritten with the prepared values. The values are marked with the F symbol.



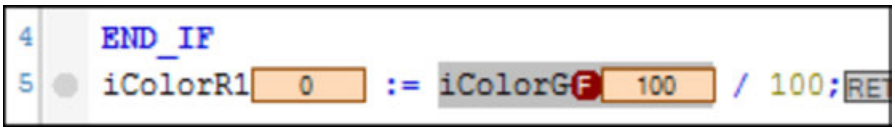
You can also force the variable values in the "Watch" view.

Forcing in the implementation part

Requirement: The application is in online mode.

1. Open the POU in the editor by choosing the command *"Project ➔ Edit Object"*.
2. In the implementation part of the editor, double-click an inline monitoring field (1).
   ⇨ The *"Prepare Value"* dialog opens.
3. Enter the new value in the field *"Prepare a new value for the next write or force operation"*.
   ⇨ The prepared value appears in the inline monitoring field.

4. Click *"Debug ➔ Force Values"*.
   ⇨ The value of the variables is overwritten with the prepared values. The values are marked with the F symbol.



Viewing and editing all forced variables one list

Requirement: The application is in online mode. Multiple variables are forced.

1. Click *"View ➔ Watch ➔ Watch all Forces"*.
   ⇨ The *"Watch all Forces"* view opens. It contains all currently forced variables of the application in the form of a watch list.
2. Select all lines in the list and click *"Unforce ➔ Unforce and Keep All Selected Values"* in the list box in the upper left part of the view.
   ⇨ The variables are unforced and they get the values that they had before forcing.

Forcing a function block input in CFC

Requirement: An application has a CFC POU that contains a function block, and the application is in offline mode.

NOTICE!
This kind of forcing uses a data breakpoint internally and is therefore different from forcing with the *"Force Values"* command or *[F7]*.

Values that were forced by the command *"Force FB Input"* do not respond to the commands *"Show All Forces"* or *"Unforce Values"*.

1. Open the editor of the CFC POU by double-clicking the object in the tree.
2. When using compiler version 3.5.11.x or 3.5.12.x, enable the "forceability" for the desired function block. Select the POU element in CFC and click *“CFC ➔ Prepare Box for Forcing”*.
3. Log in to the application on the target device. In CFC, select the input of the POU and click *“Force Function Block Input”* in the context menu.

   ⇨ The *“Force Value”* dialog opens.
4. Set a new value for the input. Example in the case of a `TON` POU: `FALSE` for the Boolean input `IN`, or `t#4s` for the `PT` input (TIME). Click *“OK”* to confirm.

   ⇨ The set value is forced immediately. A green circle is displayed at the upper left of the POU element and the name of the input in the element is highlighted in green. In the case of a Boolean value, a small monitoring view with the value also opens at the input. In the monitoring views, the forced value is displayed, for example in the *“Value”* column, as in the declaration part.
5. To remove the forced value, click *“Force Function Block Input”* again. In the *“Force Value”* dialog, select the *“Remove value”* option.

   ⇨ Forcing is canceled. The input gets the current value from the controller.

See also

- *Chapter 1.4.1.20.3.12.34 “Command 'Prepare Box for Forcing'” on page 1101*
- *Chapter 1.4.1.20.3.12.35 “Command 'Force Function Block Input'” on page 1101*
- *Chapter 1.4.1.20.4.7 “Dialog Box 'Prepare Value'” on page 1153*
- *Chapter 1.4.1.20.3.7.16 “Command 'Force Values'” on page 1053*
- *Chapter 1.4.1.20.3.7.18 “Command 'Unforce Values'” on page 1054*
- *Chapter 1.4.1.20.3.7.17 “Command 'Write Values'” on page 1053*
- *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*

1.4.1.11.5 Resetting applications

Resetting the application stops the program and resets the variables to their initialization values. Depending on the type of reset, retain variables and persistent variables are also reset.

- Reset warm: All variables are reset, except `RETAIN` and `PERSISTENT` variables.
- Reset cold: All variables are reset, except `PERSISTENT` variables.
- Reset origin: All variables are reset.
- Reset origin device: All variables are reset and all applications are deleted.

The following sample program and statements clarify the functionality of the various resets.

See also

- *“Lifespan of variables when calling online commands” on page 303*

Sample program

Example

Declaration

```
VAR
    iVar: INT := 0;
END_VAR
VAR RETAIN
    iVarRetain: INT :=0;
END_VAR
VAR PERSISTENT
    iVarPersistent : INT:= 0;
END_VAR
```

Implementation

```
iVar := 100;
iVarRetain := 200;
iVarPersistent :=300;
```

1. Insert the *"Persistent Variables"* object below the application and open it in the editor.
2. Click *"Build ➔ Build"*.
3. Click *"Declare ➔ Add All Instance Paths"*.
   ⇨ The instance path of the persistent variables is inserted.
4. Download the application to the controller.

Executing a "Reset warm", "Reset cold", and "Reset origin"

Requirement: The sample program runs on the controller.

1. Click *"Online ➔ Login"* to switch to online mode.
2. Monitor the variables `iVar`, `iVarRetain`, and `iVarPersistent`.
3. Click *"Online ➔ Reset Warm"*.
   ⇨ You are prompted whether you really want to execute the command.
4. Click *"Yes"* to confirm the dialog.
   ⇨ The application is reset. The `iVar` variable is set to the initialization value 0. Both of the other variables retain their values.
5. Click *"Online ➔ Reset Cold"*.
   ⇨ You are prompted whether you really want to execute the command.
6. Click *"Yes"* to confirm the dialog.
   ⇨ The application is reset. The `iVar` and `iVarRetain` variables are set to the initialization value 0. The `iVarPersistent` variable retains its value.
7. Click *"Online ➔ Reset Origin"*.
   ⇨ You are prompted whether you really want to execute the command.
8. Click *"Yes"* to confirm the dialog.
   ⇨ The application is reset. All variables are reset to their initialization values.

See also

- *Chapter 1.4.1.20.3.6.11 “Command 'Reset Warm'” on page 1038*
- *Chapter 1.4.1.20.3.6.10 “Command 'Reset Cold'” on page 1038*
- *Chapter 1.4.1.20.3.6.12 “Command 'Reset Origin'” on page 1039*
- *Chapter 1.4.1.20.3.6.13 “Command 'Reset Origin Device'” on page 1040*

1.4.1.11.6 Flow Control

With flow control, you can monitor the processing of the application program. Flow control is provided for the ST, FBD, LD, and CFC language editors.

With an activated flow control, CODESYS displays the variable values and results from function calls and operations at the respective processing location and time. In this way, the exact lines of code and networks that process the current cycle are marked in colors. Compare this to standard monitoring, in which CODESYS delivers only the value that a variable has between two processing cycles.

Flow control works in all parts of the editor view that are currently visible. *“Flow control enabled”* is then displayed in the status line as long as the function is active and flow control positions (processed parts of code) are visible in an editor view.

You can write values in the declaration part and implementation part. Forcing is not possible.

NOTICE!
Values are written at the end of the current cycle.

NOTICE!
When you enable flow control, the cycle time of the application is prolonged.

When *“Confirmed Online Mode”* is selected in the communication settings, a dialog prompt appears when switching on the flow control to cancel the operation.

When flow control is enabled, it is not possible to use breakpoints or step through the program.

See also

- *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*
- *Chapter 1.4.1.20.3.7.22 “Command 'Flow Control'” on page 1056*

Display of the flow control in different language editors

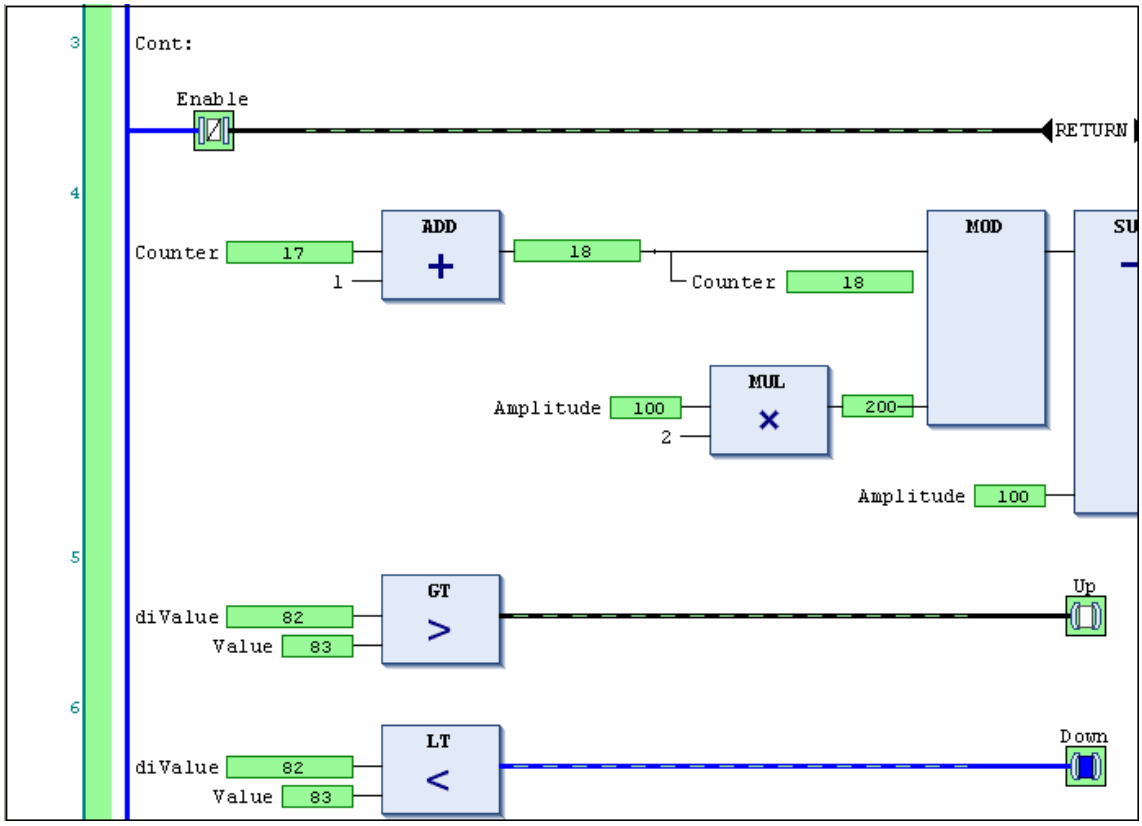
By default, CODESYS displays the flow control positions of the processed parts of code as green fields. Unprocessed parts of code are displayed in white.

Note that the displayed value of an unprocessed code position is an ordinary monitoring value. This is the value between two task cycles.

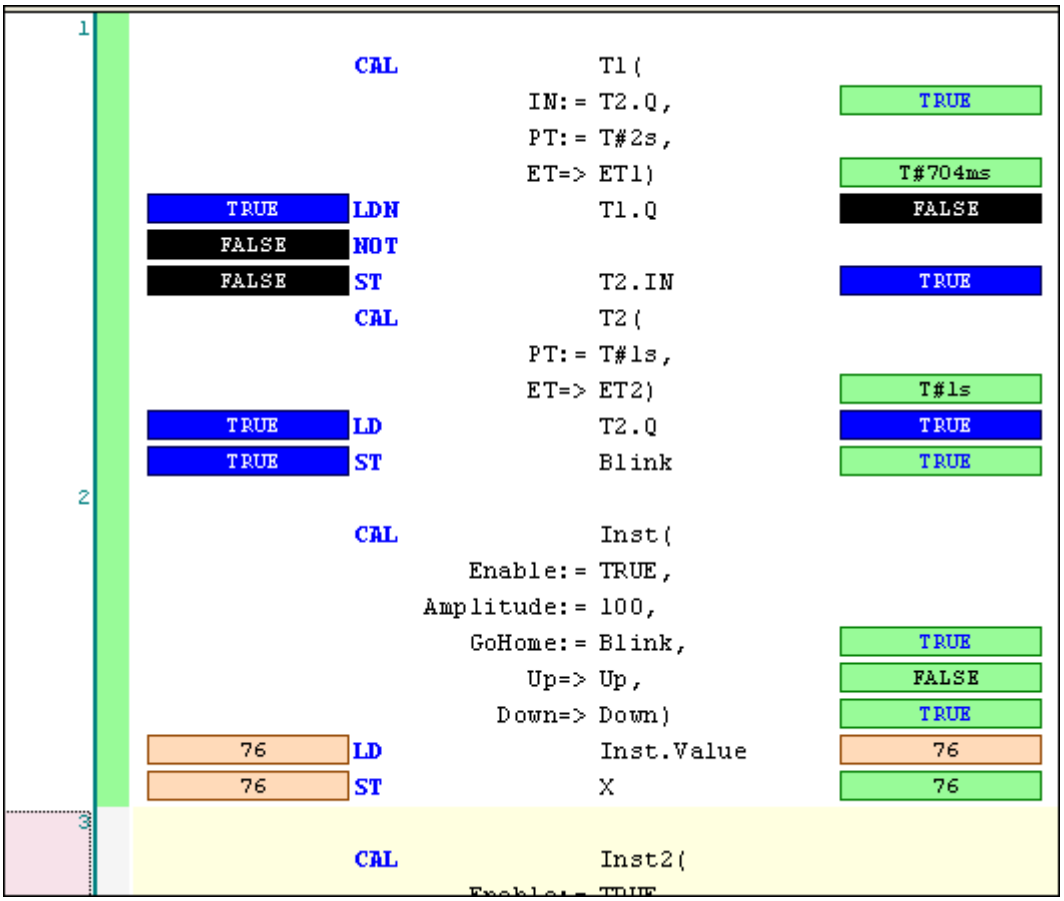
```
1  i 1619 := i 1619 + 1;
2  b 0 := NOT b 0;
3  IF str 'abcdefghij' = str1 '' THEN
4  fl2 1.5 := fl 1.23;
5  ELSE
6  fl2 1.5 :=1.5;
7  D 6.5E+04 :=B255*B255;
8  END_IF;
9  IF D 6.5E+04 <0.0 THEN
```

In network editors, CODESYS marks the processed networks with bars on the left edge in the flow control color.

In LD, CODESYS displays the currently processed connecting lines in green and all others in gray. The actual value of the connection is also displayed: `TRUE` by a bold blue line, `FALSE` by a bold black line, and unknown or analog values by thin black lines. Combinations of these lines are displayed as dashed lines.



In IL, for each statement CODESYS uses two fields for the display of the actual values. One is located to the left of the operator with the current accumulator value, and one is located to the right of the operand with the operand value.



1.4.1.11.7 Determining the current processing position with the call stack

You can use the call stack for determining the current position of the program flow. This function is very useful when stepping into programs.

Requirement: The application is in online mode. The program is halted at a breakpoint or you are stepping into it.

▷ Open the call stack by clicking *"View ➔ Call Stack"*.

⇨ The call stack opens. The list shows the current location with the complete call path.

The call stack is also available in offline mode and normal online mode (without using debugging functions). In this case, it receives the last displayed location during a stepped execution, but it is displayed in gray.

See also

- ↳ *Chapter 1.4.1.20.3.3.15 "Command 'Call Stack'" on page 993*
- ↳ *Chapter 1.4.1.11.3 "Stepping Through a Program" on page 399*

1.4.1.11.8 Checking the Task Deployment

The Tab *"Task Deployment"* of the device editor indicates in an overview the tasks that process the individual inputs and outputs of the I/O mapping of your application and the priority with which they do so. You can check here whether an unintentional overwriting of values is caused, which can lead to undefined values.

1. Generate code for the application: to do this select, for example, the command *“Build ➔ Generate Code”*.
2. Open the device editor by double-clicking on the device object in the device tree. Select the *“Task Deployment”* tab.
   ⇨ You obtain a display of the inputs and outputs of your application and the assignment of the tasks and their priorities. See the description of the *“Task Deployment”* tab for details.

See also

- *Chapter 1.4.1.20.2.8.17 “Tab 'Task deployment'” on page 869*
- *“General information about I/O mapping” on page 214*

1.4.1.12 Application at Runtime

When the application is running on the PLC, in the CODESYS Development System there are some features for monitoring and changing the values of the variables as well as for recording and storing the value charts.

Furthermore, you can poll some information from the PLC, you can have a look into the PLC-log, display a core dump and monitor the time behavior of the tasks.

Regard also the possibility to restrict the access on the running application in critical states of the machine via online commands provided by CODESYS Development System. For this purpose some system variables are available in a module of the `ComponentManager` library.

1.4.1.12.1 Monitoring of Values

In runtime mode, you can monitor the current variable values of a programming object at different places in a project. The following is what we refer to as monitoring:

- Online view of the programming editor of an object: inline monitoring
- Online view of the declaration editor of an object
- Object-independent, configurable watchlists

When you set the {attribute 'monitoring'...} pragma, you can monitor the results from function calls and the current variable values in property-type objects.

More options for recording current variable values:

– *Read and save recipes*
– *Record values on a timeline for displaying the history immediately or later: trace and trend features*

See also

- ↳ *Chapter 1.4.1.19.6.2.25 “Attribute 'monitoring'” on page 709*
- ↳ *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*
- ↳ *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- ↳ *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*
- ↳ *Chapter 1.4.1.12.4 “Data Recording with Trend” on page 430*

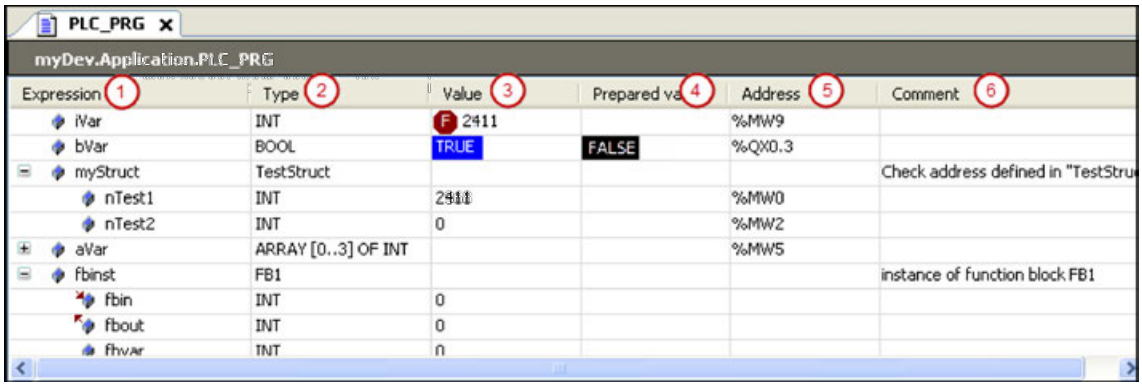
Calling of monitoring in programming objects

When an application is running on the controller, the actual values of variables are displayed in the editors of the POUs. This is how the values of variables are monitored.

☑ Requirement: The *“Enable inline monitoring”* option is activated in *“Tools* ➔ *Options”* in the *“Text Editor”* category on the *“Monitoring”* tab.

1. Download an application to the controller and start it.
2. Click *“Debug* ➔ *Display Mode* ➔ *Decimal”*.
   ⇨ The display format of the actual values is set.
3. Click a programming object in the *“Devices”* view or *“POUs”* view.
   ⇨ The respective editor opens. Actual values of the variables are refreshed continually for both the declaration and implementation.

Monitoring in the declaration editor



The actual value of an expression (1) is displayed in the *“Value”* column (3).

You can write and force a value in the *“Prepared Value”* (4) column. During the forcing, the actual value is decorated with a red symbol (F).

The expression of an interface reference can be expanded. If the interface points to a global instance, then this global instance is displayed as the first entry below the reference. Afterwards, if the interface reference changes, then the displayed reference is collapsed.

Monitoring in the implementation (inline monitoring)

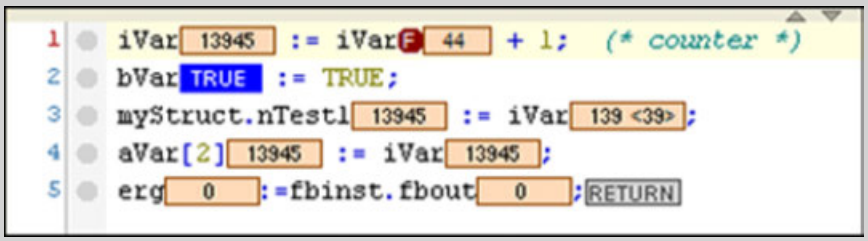
Inline monitoring is the display of the current variable value in the implementation.

Depending on the implementation language, the following displays are possible in the implementation part:

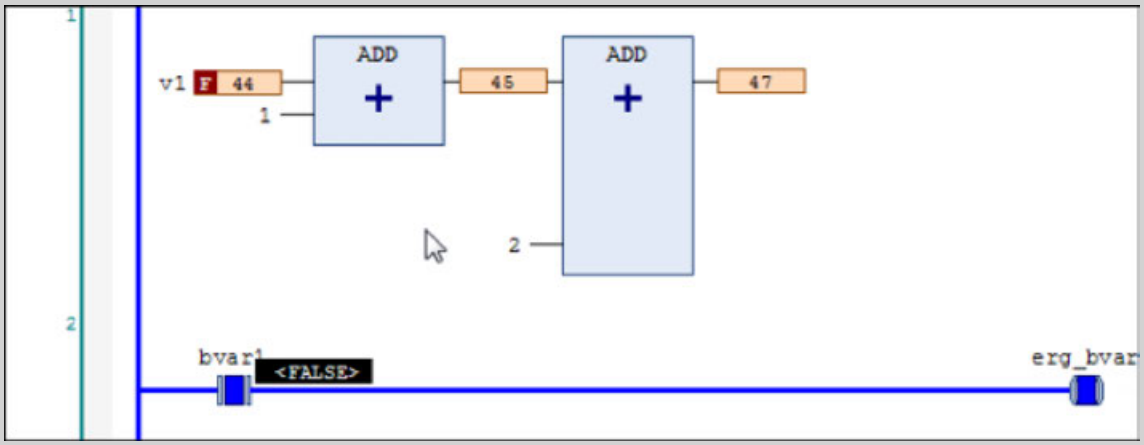
- Variables have a window with the current value displayed after their name: `nResult2` 17
  If you have prepared values for variables for forcing or writing, then they are displayed in angle brackets in the inline monitoring view after the current value.
  After forcing, the respective values are identified by the F symbol.
- Network editors and the CFC editor:
  Connecting lines are displayed in color according to their actual Boolean value (blue means `TRUE`, black means `FALSE`).

- LD editor:
  The contact and coil elements are also marked.
  For contacts and coils, a prepared value (`TRUE` or `FALSE`) is shown in a small view next to the element.
- SFC editor:
  Transitions with the value `TRUE` are displayed in color according to their actual Boolean value (blue means `TRUE`, black means `FALSE`).
  Active steps are displayed in blue.
  Forced transition values are displayed in red in the implementation.
- IL tabular editor:
  Current values are displayed in a separate column.

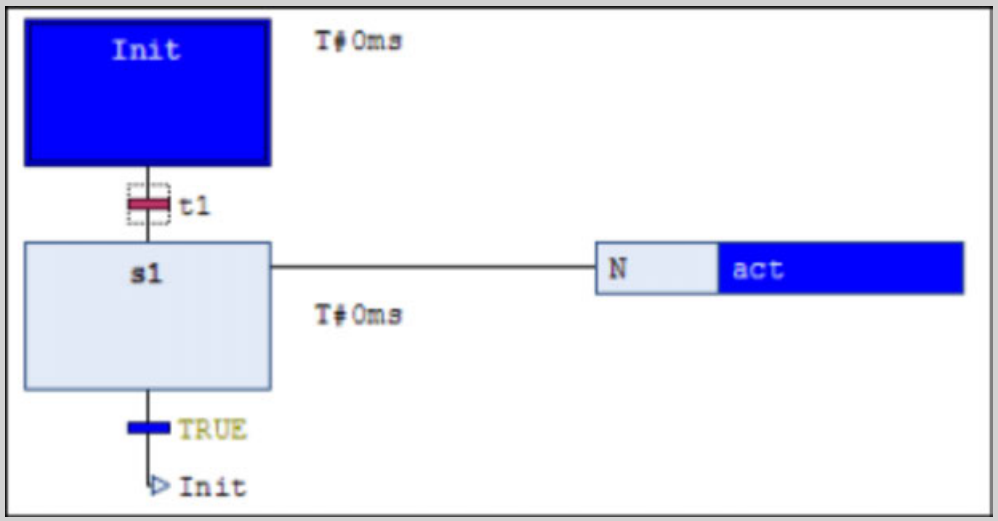
Monitoring in the ST editor



Monitoring in the LD editor



Monitoring in the SFC editor



> *You can deactivate the inline monitoring function in "Tools ➔ Options", in the "Text Editor" category on the "Monitoring" tab.*

See also

- *Chapter 1.4.1.19.1.3.1 "ST Editor" on page 463*
- *Chapter 1.4.1.19.1.3.2 "ST editor in online mode" on page 463*
- *Chapter 1.4.1.19.1.5.2 "FBD/LD/IL editor in online mode" on page 499*
- *Chapter 1.4.1.19.1.4.2 "SFC Editor in Online Mode" on page 476*
- *Chapter 1.4.1.19.1.6.4 "CFC Editor in Online Mode" on page 516*

Partial monitoring of an array

An expanded array shows the actual values for up to 1000 elements. However, this can be confusing. In addition, an array can contain more than 1000 elements. Then it is helpful to limit the range of displayed elements. You can do this in online mode in the following way.

Limiting the monitoring range

☑ Requirement: An application is running. It contains a multidimensional array variable with more than 1000 elements. Example: `arrBig : ARRAY [0..100, -9..10, -19..20] OF INT;`

1. Click in the field of the *"Data Type"* column for the `arrBig` variable.
   ⇨ The *"Monitoring Range"* dialog opens.
2. Specify the value `[1, -9, -19]` for *"Start"*.
3. Specify the value `[1, 10, 20]` for *"End"*.
   ⇨ The actual values of 800 array elements are displayed in the declaration editor. The range is limited to the elements of the index `[1, <i>, <j>]` with i from -9 to 10 and j from -19 to 20.

See also

- *Chapter 1.4.1.19.1.1 "Declaration Editor" on page 461*
- *Chapter 1.4.1.11.4 "Forcing and Writing of Variables" on page 401*
- *Chapter 1.4.1.20.4.9 "Dialog 'Monitoring Range'" on page 1156*

Monitoring a function block

When you double-click the editor view of a function block in online mode, a dialog opens where you can choose between viewing the basic implementation or a specific instance.

If you select the basic implementation, then the code is displayed in the editor without current values. Now set a breakpoint in the basic implementation. If the execution halts there, then the current values of the instance that is processed first in the program flow are displayed. Now you can step successively through all instances.

If you select one of the instances, then the editor opens with the code of the function block instance. The current values are displayed in the declaration and, if applicable, in the implementation, and are updated continuously.

See also

- *Chapter 1.4.1.20.2.18.2 "Object 'Function Block'" on page 883*
- *Chapter 1.4.1.11.2 "Using Breakpoints" on page 395*

Monitoring a property

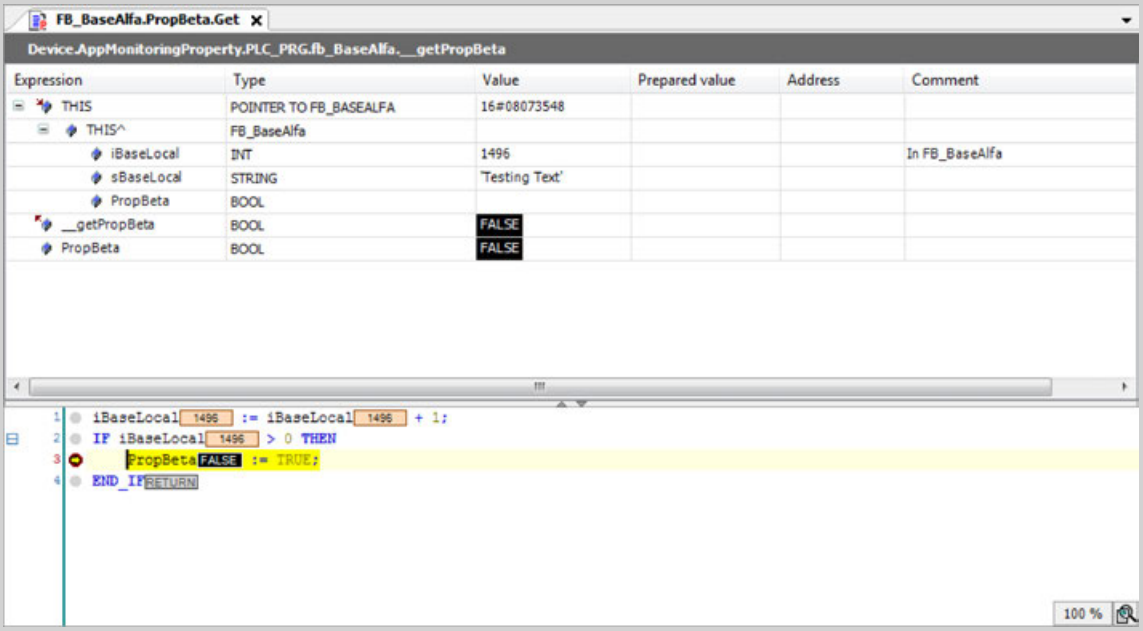
You can monitor variables in a property object by setting a breakpoint in the function during online mode. When halted there, the current values are displayed.

In addition to your own values, the values of the variables of the superordinate instance are displayed automatically. In the declaration part of the property, the `THIS` pointer, which points to the superordinate instance, appears in the first line with the current data type information and values.

Example

Code

```
FUNCTION_BLOCK FB_BaseAlfa
VAR
    iBaseLocal : INT;
    sBaseLocal : STRING;
END_VAR
iBaseLocal := iBaseLocal + 1;
sBaseLocal := 'Testing Text';

FB_BaseAlfa.PorpBeta.Get
iBaseLocal := iBaseLocal + 1;
IF iBaseLocal > 0 THEN
    PropBeta := TRUE;
END_IF

FB_BaseAlfa.PorpBeta.Set
IF PropBeta = TRUE THEN
    iBaseLocal := 0;
    sBaseLocal := 'Tested IF';
END_IF


PROGRAM PLC_PRG
VAR
    fb_BaseAlfa : FB_BaseAlfa;
END_VAR

fb_BaseAlfa();

IF fb_BaseAlfa.PropBeta = TRUE THEN
    xResult := TRUE;
END_IF
IF xReset THEN
    fb_BaseAlfa.PropBeta := TRUE;
    xReset := FALSE;
END_IF
```



See also

- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*

Monitoring of property access in the superordinate programming object

You can monitor the values of subordinate properties in a function block or program in addition to the variable values.

To do this, add either the pragma `{attribute 'monitoring' = 'variable'}` or `{attribute 'monitoring' = 'call'}` to the subordinate property object in the declaration. If you open the superordinate program instance or function block instance at runtime, then the current property values are displayed in the editor in addition to the current variable values.

See also

- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.19.6.2.25 “Attribute 'monitoring'” on page 709*

Monitoring a method

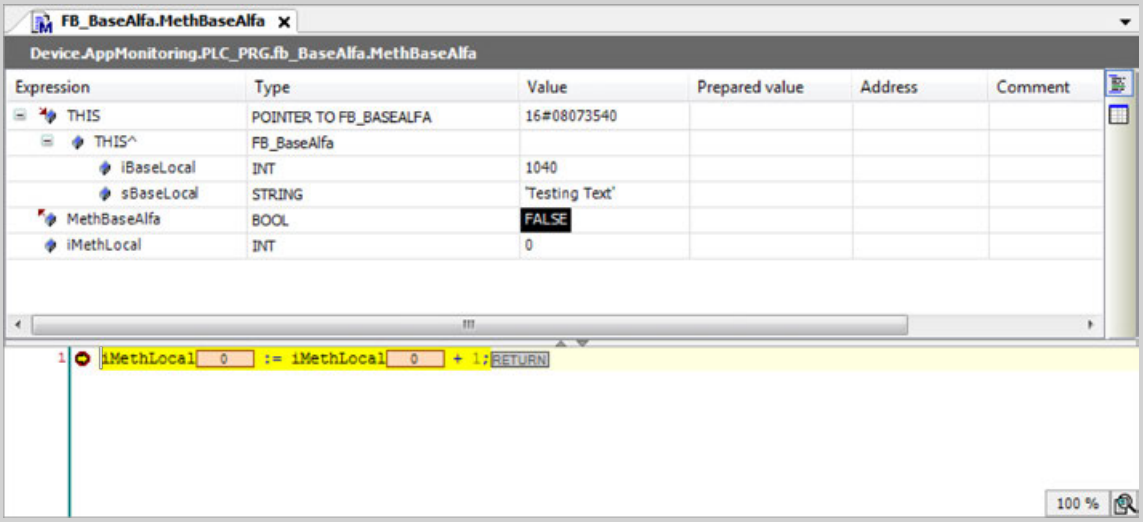
You can monitor variables in a method object by setting a breakpoint in the method during online mode. When halted there, the current values are displayed.

In addition to your own values, the values of the variables of the superordinate instance are displayed automatically. In the declaration part of the method, the `THIS` pointer, which points to the superordinate instance, appears in the first line with the current data type information and values.

Example

Code

```
FUNCTION_BLOCK FB_BaseAlfa
VAR
    iBaseLocal : INT;
    sBaseLocal : STRING;
END_VAR
iBaseLocal := iBaseLocal + 1;
sBaseLocal := 'Testing Text';

METHOD MethBaseAlfa : BOOL // Method of FB_BaseAlfa
VAR_INPUT
END_VAR
VAR
    iMethLocal : INT;
END_VAR
iMethLocal := iMethLocal + 1;


PROGRAM PLC_PRG
VAR
    fb_BaseAlfa : FB_BaseAlfa;
END_VAR
fb_BaseAlfa();
fb_BaseAlfa.MethBaseAlfa();
```



See also

- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*

Monitoring a function

You can monitor variables in a function object by setting a breakpoint in the function during online mode. When halted there, the current values are displayed.

Monitoring the return value of a function call

In the ST editor of a POU, the current return value is displayed as inline monitoring at the position of the POU where a function is called.

The following conditions must be fulfilled:

- The value can be interpreted as a 4-byte numeric value. Example: `INT`, `SINT`, or `LINT`.
- The pragma `{attribute 'monitoring' := 'call'}` is inserted into the function.

See also

- *Chapter 1.4.1.20.2.18.3 “Object 'Function'” on page 886*
- *Chapter 1.4.1.19.6.2.25 “Attribute 'monitoring'” on page 709*

Using watch lists

What is a watch list?

A watch list is a user-defined list of project variables that are collected in one view for the purpose of monitoring their values. In online mode, you can write and force variable values in a watch list. Monitoring, writing, and forcing are handled the same way as the declaration editor in online mode. You can customize the format of the representation of floating-point values in the options for monitoring.

There are four, ready-to-use watch lists (`Watch <n>`) available in a project. Click *“View ➔ Watch”*.

> *If the expression is an interface reference, then it can be expanded. If the interface points to a global instance, then this global instance is displayed as the first entry below the reference. If the interface reference changes, then the displayed reference is collapsed.*

See also

- *Chapter 1.4.1.19.1.1 “Declaration Editor” on page 461*
- *Chapter 1.4.1.20.4.13.18 “Dialog 'Options' - 'Monitoring'” on page 1197*

Creating and editing a watch list (offline or online mode)

Requirement: The project is in either online or offline mode. It includes an application with declared variables that you want added to one of the four possible watch lists.

1. Click *“View ➔ Watch ➔ Watch <n>”*.

   ⇨ The `Watch <n>` view opens. It contains a blank table row.

2. Double-click the field in the *“Expression”* column and type a variable to monitor, either manually or with the input assistant.

   Syntax: <device name>.<application name>.<object name>.<variable name>

   Example: "`Dev1.App1.PLC_PRG.ivar`"

   If you type the name of a structured variable, then the individual components are displayed automatically in other lines in online mode.

3. Define all successive variables that will be monitored with this list. You can change the order by using drag and drop operations.

   ⇨ The *“Execution point”*, *“Type”*, *“Address”*, *“Comment”* fields are filled in automatically according to the variables declaration. The symbol before the expression indicates the type of variable: input variable (), output variable (), or ordinary variable ( ).

> *In online mode, you can also create or edit watch lists by right-clicking and choosing the “Add Watch” command.*

See also

- *Chapter 1.4.1.20.3.3.8 “Command 'Watch' - 'Watch <n>'” on page 987*

Adding variables by choosing the 'Add Watch' command (online mode)

Requirement: A project is open and running. It includes an application with declared variables that you want added to a possible watch list.

1. Click *“View ➔ Watch <n>”* to open the watch list.
2. Place the cursor on a variable in the declaration or implementation part of a POU and right-click to choose the *“Add Watch”* command.
   ⇨ This adds an entry to the list for the selected variable.
3. You can add other variables in this way or by typing directly into the list in the *“Expression”* field as described above.
   ⇨ The watch lists are updated immediately.

If a watch list is not open when you click “Add Watch” for a variable, then it is added automatically to the “Watch 1” list.

Writing and forcing variable values is also possible in the watch lists. In online mode, the “Prepared value” column is also available.

See also

- *Chapter 1.4.1.20.3.22.1 “Command 'Add Watch'” on page 1147*
- *Chapter 1.4.1.12.1.1 “Calling of monitoring in programming objects ” on page 410*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*

1.4.1.12.2 Changing Values with Recipes

Use recipes to change or read recipes values for a specific set of variables (recipe definition) on the controller at the same time.

You define the basic settings for recipes, such as location and format, in the *“Recipe Manager”* object. Insert one or more recipe definitions below this object. A recipe definition is composed of one or more recipes for the contained variable. The recipe consists of specific variable values.

You can save a recipe to a file or write directly from files to the PLC.

Recipes can be loaded via the CODESYS development interface, the visualization element, or the application program.

Using recipes on remote devices

The variable values from recipes are transferred automatically to and from another controller when they are data source variables and a data source exchange is configured. Reading and writing occurs synchronously. Therefore, CODESYS updates all variables in a recipe at the same time. After reading or writing, you can use the call `g_RecipeManager.LastError` *to check whether or not the transfer was successful (*`g_RecipeManager.LastError = 0`*).*

See also

- *Chapter 1.4.1.20.2.22 “Object 'Recipe Manager'” on page 923*
- *Chapter 1.4.1.20.2.23 “Object 'Recipe Definition'” on page 926*
- *Chapter 1.4.1.9.4 “Data Link with Data Sources” on page 363*

Handling of recipes in the CODESYS user interface

The development interface of CODESYS provides commands for generating recipes as well as reading/writing in online mode.

See also

- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Using recipes in applications

At runtime, you can use recipes in the user program and visualization elements.

In the user program, you use the methods for the function block `RecipeManCommands` from the library `RecipeManagement`. In the visualization, you use recipes via the input configuration (internal command of visualization elements.

During the initialization process, the recipe management reads the values of the variables that are defined in the recipe definition. This operation takes place at the end of the initialization phase of the application. At this point, all initial values of the application variables are set. This is performed to initialize missing values from recipe files correctly.

See also

- *RecipeManCommands*
- *Input Configuration*

Creating a recipe

1. Select the *“Application”* object in the device tree.
2. Click *“Project ➔ Add Object ➔ Recipe Manager”*.
   ⇨ CODESYS adds the Recipe Manager to the device tree.
3. Select the *“Recipe Manager”* object in the device tree.
4. Click *“Project ➔ Add Object ➔ Recipe Definition”*.
   ⇨ CODESYS adds the recipe definition below the Recipe Manager.
5. Open the editor of the recipe definition by double-clicking the object.
6. Double-click the blank field below *“Variable”*. Specify the name of a variable that you will define a recipe. The Input Assistant can be used for this: [...] button.
7. Click *“Recipes ➔ Add New Recipe”* and specify a name for the new recipe.
   ⇨ A column with the new recipe name appears in the editor.
8. Enter the variable value for this recipe in this field.
9. Insert additional fields as needed.
10. Select a variable value for the recipe and click *“Recipes ➔ Save Recipe”*. Select a location and file name.
    ⇨ CODESYS saves the recipe in the format as defined in the Recipe Manager.

See also

- *Chapter 1.4.1.20.2.22 “Object 'Recipe Manager'” on page 923*
- *Chapter 1.4.1.20.2.23 “Object 'Recipe Definition'” on page 926*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Loading a recipe from a file

Requirement: A Recipe Manager is available in the application. In a recipe definition, there is a *“myRec”* recipe with variable values. A `myRec.txt` recipe file is located on the file system and contains the entries for this recipe.

Example of the recipe file:

```
PLC_PRG.bVar:=0
PLC_PRG.iVar:=2
PLC_PRG.dwVar:=35232
PLC_PRG.stVar:='first'
PLC_PRG.wstVar:='123443245'
```

1. Double-click the *“Recipe Definition”* object in the device tree to open the tabular editor for the definition of the individual recipes.
   ⇨ You see the `myRec` column with the current values for this recipe.
2. Edit the `myRec.txt` file in an external text editor and replace the variable values with other values that you want to load into the recipe definition in CODESYS. Save the file.
3. Click the *“myRec”* column in the recipe definition and click *“Load Recipe”* in the context menu.
   ⇨ A dialog prompt notifies you about the possibly needing to perform an online change when logging in again. An online change is necessary when you change the current values of the recipe variables by loading the recipe.
4. Click *“Yes”* to close the dialog and continue. Select the `myRec.txt` file from the file explorer for loading.
   ⇨ The recipe values in the recipe definition are updated according to the values read in the file.

If you want to overwrite only individual recipe variables with new values, then remove the values for the other variables before loading to the recipe file. Entries without value definitions are not read, and therefore updating leaves these variables unchanged on the PLC and in the project.

For values of the data type REAL/LREAL, the hexadecimal value is also written to the recipe file in some cases. This is necessary so that the exact identical value is restored when converting back. In this case, change the decimal value and delete the hexadecimal value.

See also

- *Chapter 1.4.1.20.3.19.4 “Command ‘Load Recipe'” on page 1128*
- *Chapter 1.4.1.20.3.19.8 “Command 'Load and Write Recipe'” on page 1129*

Recipe management on the controller; memory usage

When you clear the *“Recipe management in the PLC”* option, the Recipe Manager and recipe definitions will not use any memory on the PLC.

If you select this option, then code is generated for the Recipe Manager and all recipe definitions, and this code is stored on the PLC. The size of the used memory primarily depends on the number pf recipes and their variables, as well as the data type of the variables. Whether or not the fields of the recipe definition are filled also has an effect. The memory usage of recipes cannot be calculated. It has to be determined by experimentation at the time it is needed. The following table merely provides some guiding principles.

	Code Size (bytes)	Data Size (bytes)	Total (bytes)
Recipe definition with 100 INT variables	194406	79400	267352
Recipe definition with 200 INT variables	238318	121284	459344
Recipe definition with 300 INT variables	282230	163084	543856
Recipe definition with 100 BOOL variables	192742	69884	343168
Recipe definition with 200 BOOL variables	235446	101568	436872
Recipe definition with 300 BOOL variables	278146	133284	510072
Recipe definition with 100 string variables	203278	870084	1154000
Recipe definition with 200 string variables	255570	1709784	2973296
Recipe definition with 300 string variables	307886	2549484	2964112

Loading recipe values from the controller

You can apply recipe values on the controller to recipe definitions in the project, even if these definitions have been modified in the project.

Requirement: The *“Recipe management in the PLC”* is option is selected in the Recipe Manager.

1. Create a recipe definition `RecDef1` in the project, containing the variables `PLC_PRG.ivar` and `PLC_PRG.bvar`. Insert a recipe *“R1”*: value for `PLC_PRG.ivar`: `33`; value for `PLC_PRG.bvar`: `TRUE`.
2. Log in to the controller and download the application.
   ⇨ The recipe file `R1.RecDef1.txtrecipe` is saved to the default directory of the controller ($PlcLogic$).
3. Logout and add another variable `PLC_PRG.dwvar` to the recipe definition in the project.
4. Edit the recipe definition file `R1.RecDef1.txtrecipe` on the device by changing the value for `PLC_PRG.ivar` from `33` to `34`.

   Moreover, add another recipe *“R2”* on the device. To do this, copy the `R1.RecDef1.txtrecipe` and rename it to `R2.RecDef1.txtrecipe`. Then edit this file and change the recipe values: `PLC_PRG.ivar`: `1`, `PLC_PRG.bvar`: `FALSE`.

   ⇨ Now two recipes *“R1”* and *“R2”* are available on the device. In the project, there is only *“R1”*, and it also contains other values than *“R1”* on the device.
5. Log in to the controller by means of an online change.
6. Click *“Load Recipes from Device”* from the context menu.
   ⇨ A dialog prompt notifies you that executing the command at the next login may trigger an online change, and that the recipes on the runtime system will overwrite the recipes of the current recipe definition.
7. Confirm that you want to continue.
   ⇨ A dialog prompt notifies you that the recipe for `PLC_PRG.dwvar` loaded on the device cannot yield a value from the controller.

8. Confirm that you want to continue.

   ⇨ The value of `PLC_PRG.ivar` in recipe *"R1"* of the recipe definition in the project changes to `34`. The recipe *"R2"* with the values `1` and `FALSE` is also listed in the recipe definition now. `PLC_PRG.dwvar` remains in the recipe definition.

1.4.1.12.3 Data Recording with Trace

You can use a *"Trace"* to follow the value history of variables on the controller in a similar way as a digital sampling oscilloscope. When the application is in runtime mode with trace, all statements are executed first within the task cycle. Then, data recording starts with value storage including time stamps. These time stamps are relative and refer to the start time of the data recording. The data yields a discrete time signal and CODESYS displays its course in the trace editor.

A sample (data record) is composed of the value and the time stamp. The runtime system writes the samples to a buffer with a definable size. CODESYS requests the data, saves it in the trace editor buffer, and displays it in the trace diagram as a function of time. You can monitor the value history of the configured variables continually because CODESYS displays the latest data.

You can trigger the data recording. When this happens, the application saves the data from the time of the trigger and CODESYS displays the data at the time of the trigger.

The configuration and the display of a trace are possible in the CODESYS project by means of trace objects in the trace editor. There are the following two object types:

- *"Trace"*: Inserted below the IEC application in the device tree. This kind of object always contains a purely application-specific trace configuration. You can download this trace configuration to the controller and run it with the application.
- *"DeviceTrace"*: Inserted below the device object in the device tree. If the PLC supports a trace manager, then you can use one or more *"DeviceTrace"* objects to access one or more traces that are running on the controller. These can be both application-specific or controller-specific traces. For example, a controller can support traces for recording the processor load. Menu commands allow for access from the CODESYS project to the trace manager in the device.

Access to the trace manager from IEC code is possible by means of the functions from the library `CmpTraceMgr.library`. For more information, refer to the library documentation.

NOTICE!
A running data recording with trace can lead to a significant increase in the cycle time of the IEC task.

NOTICE!
Data recording with trace also continues running after logging out of the device.

Runtime system component `CmpTraceMgr`, "Trace manager"

The device description of a runtime system with trace manager includes the `tracemanager` entry in the `TargetSettings` section.

In this case, CODESYS transfers only the trace configuration when downloading the application to the PLC. When you start the trace, the application interprets the configuration on the RTS by means of the trace manager, executes the data recording, and buffers the data sets on the PLC.

The `CmpTraceMgr` runtime system component provides extended functionality, as compared to data recording with IEC code.

Data recording is therefore possible as follows:

- Parameters on the PLC (for example, the processor load (cpuload, plcload), or the temperature curve of a CPU or a battery). The measurement of the processor load per CPU core (cpuload) is interesting for multicore controllers.
- Device signals (for example, the current path of a drive)
- System variables of another runtime system component

You can configure parameters like IEC variables in the *“Trace Configuration”* dialog of the *“Variable Settings”*.

The display of traces that run on the controller is possible in the trace editor of a DeviceTrace object.

See also

- *Chapter 1.4.1.12.3.4 “Accessing All Traces on the Controller” on page 428*
- *Chapter 1.4.1.20.2.29 “Object 'DeviceTrace'” on page 948*
- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*

Data recording after triggering

To monitor data that depends on an event or a condition, you can free the data recording that depends on a trigger. At runtime, the application checks whether the event has occurred or the the condition is fulfilled, and then it buffers the data accordingly.

The trace configuration enables triggering by:

- a trigger variable that maps the event
- a condition as expression
- a combination of trigger variable and condition

Saving samples to a file

You can save samples from the development system to a file. The file can also include the trace configuration.

Table 28: Possible file formats

File Extension	File type	Description
`*.trace`:	*“Trace file”*	Contains the samples and the trace configuration in XML format. You can execute the *“Load Trace”* command to load the file to the trace editor when offline and analyze the samples without a controller.
`*.txt`	*“Text File”*	Contains the samples in ASCII format. You can edit the file with an external tool.

File Extension	File type	Description
`*.trace.csv`	*"Trace dump"*	File in CSV format includes the trace configuration and optional samples. You can create the file by clicking *"Export Symbolic Trace Config"*. You can transfer the file to the controller and load it to the application. Then you can execute the *"Load Trace"* command in CODESYS to display this in the trace editor. You can also click *"Trace ➔ Save Trace"* and select the `*.trace.csv` file format. You can transfer the file to the controller and load it with an HMI for analysis.
`*.traceconfig`	*"Symbolic trace configuration"*	Contains the trace configuration CSV format. You can create the file by clicking *"Export Symbolic Trace Config"*. The `CmpTraceMgr` runtime system component can read the file.

See also

- ↳ *Chapter 1.4.1.20.3.21.15 "Command 'Save Trace'" on page 1145*
- ↳ *Chapter 1.4.1.20.3.21.8 "Command 'Load Trace'" on page 1141*
- ↳ *Chapter 1.4.1.20.3.21.7 "Command 'Export Symbolic Trace Config'" on page 1139*
- ↳ *Chapter 1.4.1.20.2.25 "Object 'Symbol Configuration'" on page 927*

See also

- ↳ *Chapter 1.4.1.20.2.28 "Object 'Trace'" on page 945*
- ↳ *Chapter 1.4.1.20.2.29 "Object 'DeviceTrace'" on page 948*

Getting started

Program

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    iVar : INT;
    rSin : REAL;
    rVar : REAL;
END_VAR

iVar := iVar + 1;
iVar := iVar MOD 33;

rVar := rVar + 0.1;
rSin := 30 * SIN(rVar);
```

☑ Requirement: The application is running the `PLC_PRG` program on the controller.

1. In the device tree, select the application and add a new trace object by clicking *"Project ➔ Add Object"*.

   ⇨ The respective trace editor opens with the commands available in the *"Trace"* menu.

2. Click *"Trace ➔ Configuration"*.

   ⇨ The *"Trace Configuration"* dialog box opens.

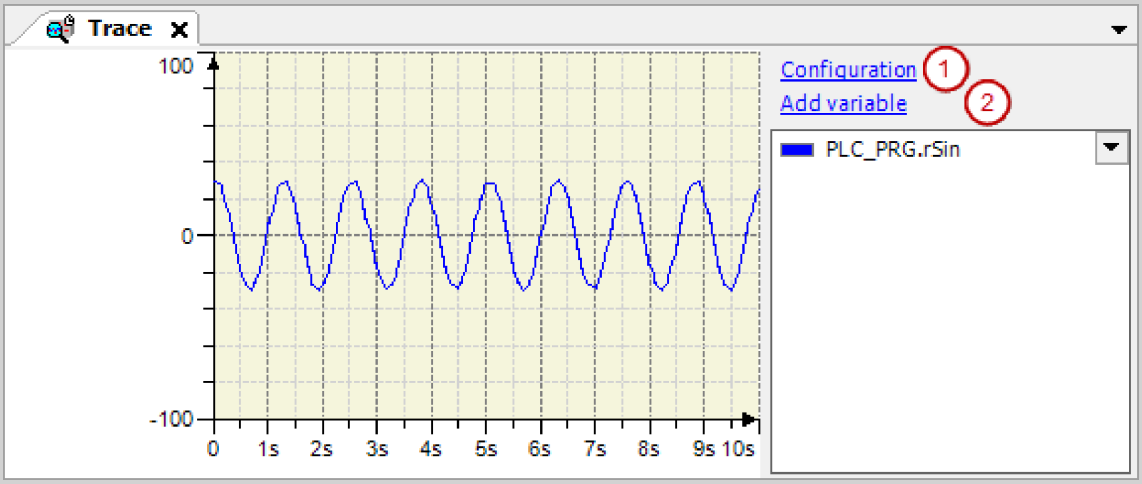
3. Select a task for running the trace feature. Normally this is the same task that is running in PLC_PRG.

4. Click *“Add Variable”* to add an entry to the tree view of the trace configuration and assign an IEC variable (for example, `PLC_PRG.rSin`).
5. Click *“Trace ➔ Download Trace”*.
   ⇨ CODESYS loads the trace configuration to the controller. The application starts recording data and transmits the data to CODESYS, where it is displayed in the trace diagram as a graph. Commands are provided for navigating through the samples and controlling the data recording.

Example

Trace the sine-shaped data of the IEC variable `PLC_PRG.rSin`

The `PLC_PRG` program is running on the controller. When you follow the instructions for "Getting Started", CODESYS displays the following trace diagram.



- (1) : *“Configuration”*
- (2) : *“Add Variable”*

See also

- *Chapter 1.4.1.20.2.28 “Object 'Trace'” on page 945*

Creating trace configuration

For a complete trace configuration, specify at least one task and one variable. In order to trigger the data recording, activate the trigger option and select a trigger variable or specify a recording condition.

See also

- *Chapter 1.4.1.20.4.15.1 “Dialog 'Advanced Trace Settings'” on page 1208*
- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Chapter 1.4.1.20.4.17 “Dialog Box 'Advanced Trend Settings'” on page 1214*
- *Chapter 1.4.1.12.4.2 “Configuring trend recording” on page 432*

Assigning a task

In this task, the data recording is executed in runtime mode. Usually the same task is selected where the variables are written.

1. Double-click the trace object.
   ⇨ The trace editor opens with the commands available in the *“Trace”* menu.

2. Click *“Trace ➔ Configuration”*.
   ⇨ The *“Trace Configuration”* dialog opens. In the tree view *“Trace Record”*, the top item is selected and the subdialog *“Record Settings”* is shown on the right.
3. Click the "arrow down" symbol (▼) in the *“Task”* drop-down list.
   ⇨ The drop-down list opens with all tasks that are available throughout the application.
4. Select a task for the trace.

See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*

Configuring a trace variable

1. Double-click the trace object.
   ⇨ The trace editor opens. The commands of the *“Trace”* menu are available.
2. Click *“Add Variable”*.
   ⇨ The *“Trace Configuration”* dialog opens. The subdialog *“Variable Settings”* is displayed on the right.
3. Click … in the input field of the *“Variable”* setting and select a trace variable in the *“Input Assistant”* dialog.
   ⇨ The variable is configured for data recording. The trace record tree and the display tree were extended by the variable.
4. Click the *“Add Variable”* link.
   ⇨ The trace record tree and the display tree receive a new variable. The settings of the variables are available on the right.
5. Select a trace variable.
6. Click *“OK”* to close the dialog.
   ⇨ The variables are trace variables and are displayed in the trace variable list.

Deleting a trace variable

1. Double-click the trace object.
2. Click a variable in the trace record tree.
3. Click the *“Delete Variable”* command or press *[Del]*.
4. Click *“OK”* to close the dialog.
   ⇨ The variable is removed from the trace variable list.

Tracing a parameter

1. Double-click the trace object.
2. Click *“Add Variable”*.
   ⇨ The *“Trace Configuration”* dialog opens. The subdialog *“Variable Settings”* is displayed on the right.
3. Click ▼ (right of the *“Variable”* setting, left of the input field).
4. Select the *“Parameter”* option in the drop-down list.
5. Click … and select a parameter from the *“Input Assistant”* dialog.

6. Configure how the parameter is displayed.
7. Click *“OK”* to close the dialog.
   ⇨ The parameter will be traced and displayed in the trace variable list.

Configuring a trigger

1. Double-click the trace object.
   ⇨ The trace editor opens with the commands available in the *“Trace”* menu.
2. Click *“Trace ➔ Configuration”*.
   ⇨ The *“Trace Configuration”* dialog opens. The subdialog *“Record Settings”* is displayed on the right.
3. Select the *“Enable trigger”* check box.
4. Select the task in which the trend record is to be executed.
5. Select a variable from the *“Trigger Variable”* field.
6. Click *“OK”* to close the dialog.
   ⇨ The data recording will be triggered.

The trigger time is displayed as a black line in the diagram in runtime mode.

1. Download the application and start it.
2. Click *“Trace ➔ Download Trace”*.
   ⇨ The trace configuration is loaded. After triggering, the runtime system saves the value graph of the trace variables. The data is displayed in the trace editor. The trigger time is displayed as a black line in the diagram.

Configuring the display of the time axis

1. Double-click a trace object.
2. Click the *“Configuration”* link above the configuration tree.
   ⇨ The *“Trace Configuration”* dialog opens.
3. Select *“Time axis”* in the display tree (below *“Presentation (Diagrams)”*).
   ⇨ The display settings of the time axis are shown on the right.
4. Edit the presets and click the *“Preview”* link.
   ⇨ The changes are seen in the coordinate system preview.
5. Click *“Y-axis”* in the display tree. The *“Y-axis”* item is below every configured diagram. Therefore, the display of the value axis is set for each diagram.
   ⇨ The subdialog *“Display Settings”* of the selected axis is displayed on the right.
6. Change the preset value.
   ⇨ The changes are applied in the coordinate system preview.
7. Click OK to close the *“Trace Configuration”* dialog.
   ⇨ The display changes are visible in the affected diagrams.

Configuring the display of the trace variable

1. Double-click a trace object.
2. Click the *“Configuration”* link.
   ⇨ The *“Trace Configuration”* dialog opens.
3. Select a variable below *“Trace Record”*.
   ⇨ The subdialog *“Variable Settings”* of the selected variable is displayed on the right.
4. Change a setting, for example the *“Line type”*.
5. Click *“OK”* to close the dialog.
   ⇨ The display changes are visible in the affected diagrams.

Configuring the buffer for data on the runtime system

1. Double-click a trace object.
2. Click *“Trace ➔ Configuration”*.
   ⇨ The *“Trace Configuration”* dialog opens. The subdialog *“Record Settings”* is displayed on the right.
3. Click *“Advanced”*.
   ⇨ The *“Advanced Trace Settings”* dialog opens.
4. Change the setting *“Measure in every n-th cycle”* or *“Recommended runtime buffer size (samples)”*.
5. Click *“OK”* to close the dialog.
   ⇨ The buffer settings are reconfigured. It is applied after the trace configuration is loaded to the RTS the next time.

Editing the trace configuration in runtime mode

Requirement: The application is running on the controller and a trace configuration is loaded.

1. Double-click a variable in the trace record tree.
   ⇨ The *“Trace Configuration”* dialog opens.
2. Change the color, for example.
   ⇨ The variable is displayed in the new color in the affected diagrams without interrupting the execution of the application.

If you change essential settings, for example a trace variable, then you must download the trace configuration to the controller again.

Operating the data recording

Use menu commands for controlling how data is recorded.

Requirement: The application is loaded on the runtime system and a trace is configured.

Menu commands

- *“Trace ➔ Download Trace”*
- *“Trace ➔ Start Trace”*
- *“Trace ➔ Stop Trace”*
- *“Trace ➔ Reset Trigger”*

See also

- *Chapter 1.4.1.20.3.21.6 “Command 'Download Trace'” on page 1138*
- *Chapter 1.4.1.20.3.21.16 “Command 'Start Trace'” on page 1145*
- *Chapter 1.4.1.20.3.21.17 “Command 'Stop Trace'” on page 1145*
- *Chapter 1.4.1.20.3.21.13 “Command 'Reset Trigger'” on page 1144*

Accessing All Traces on the Controller

If the controller supports the runtime system component `CmpTraceMgr` (Trace Manager), then you can access all traces from a CODESYS project which are running on the controller. In addition to application-related traces that capture the values of IEC variables, these can also be entirely controller-specific traces (for example, for recording device signal values or the CPU load).

For each trace running on the controller that you want to present in your project, you have to insert an individual *“DeviceTrace”* object in the device tree.

In order to show a trace from the device in this object, the connection to the PLC has to be configured correctly (*“Communication Settings”*). Then use one of the following menu commands:

- *“Trace ➔ Upload Trace”*: Establishes the connection to the PLC and opens the *“Online List”* dialog for selecting a trace from the controller.
- *“Trace ➔ Online List”*: Available in online mode only: Also opens the *“Online List”* dialog.

Now the trace uploaded from the controller can be started and traced in the editor of the DeviceTrace object. The configuration of the presentation (colors, labels, etc.) is the same as with traces for application variables configured in the project.

> **NOTICE!**
>
> **Closing the DeviceTrace editor terminates the connection to the controller.**
>
> Please note that the connections to the controller is also terminated when the last open *“DeviceTrace”* editor is closed. In order for device traces to be displayed again in the project, you have to reload them into the *“DeviceTrace”* objects.
>
> At this time, closing the editor is also the recommended procedure for deliberately terminating the connection to the controller. Logging out is not enough for this.

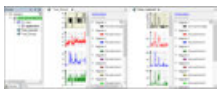
See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Chapter 1.4.1.20.2.29 “Object 'DeviceTrace'” on page 948*
- *Chapter 1.4.1.20.3.21.19 “ Command 'Upload Trace'” on page 1146*
- *Chapter 1.4.1.20.3.21.12 “Command 'Online List'” on page 1143*

Displaying the CPU load with DeviceTrace objects in the CODESYS project (example)

Requirement: The PLC device supports the Trace Manager. For the example described here, this is CODESYS Control Win V3. The device provides traces of the individual CPU loads (`CpuLoad`), as well as traces of the CPU load caused by the runtime system (`PlcLoad`). The possible display of the CPU load in the project can be helpful when using multicore functionality.

1. In the project, define the *"Communication Settings"* for the controller.
2. Select the PLC entry in the device tree and add a *"DeviceTrace"* object.
3. Rename *"DeviceTrace"* to "Trace_PlcLoad" (*"Properties"*).
4. Set the focus in the trace editor and click *"Trace ➔ Upload Trace"*.
   ⇨ The connection to the controller is established and the *"Online List"* dialog opens.
5. Select the *"PlcLoad"* entry in the dialog and click *"Upload"*. Click OK to close the dialog.
   ⇨ Multiple trace views open in the trace editor to show the CPU load in the runtime system. There are the traces for the particular CPUs and one trace for the average value. The following text appears for each: "No samples have been recorded."
6. Click *"Trace ➔ Start Trace"*.
   ⇨ The trace recording for the four parameters is displayed.
7. If you also want to display the traces for the `CpuLoad` per CPU with their average value in the project, then insert another *"DeviceTrace"* object into the device tree. Name it "Trace_CpuLoad" for example. Load and start the traces for *"CpuLoad"* in the editor as described above.
   ⇨ Now you can monitor all traces in the project:



8. If you want to change the appearance of the presentation, then click *"Configuration"* in the respective trace editor window to access the configuration dialogs. You can use these dialogs (except variable assignments) in the same way as for an IEC variable trace created in a project.
9. To disconnect from the controller, close all open DeviceTrace editor windows. If you are logged in to the device, then logging out is enough to terminate the connection.

See also

- ↳ *Chapter 1.4.1.20.2.8.2 "Tab 'Communication Settings'" on page 840*

Navigating into trace data

Use menu commands to navigate the data in the trace diagram.

Requirement: The application is in online mode.

Menu commands

- *"Trace ➔ Cursor"*
- *"Trace ➔ Mouse Zooming"*
- *"Trace ➔ Reset View"*
- *"Trace ➔ AutoFit"*
- *"Trace ➔ Compress"*
- *"Trace ➔ Stretch"*
- *"Trace ➔ Convert to Single-Channel"*
- *"Trace ➔ Convert to Multi-Channel"*

See also

- *Chapter 1.4.1.20.3.21.5 “Command 'Cursor'” on page 1137*
- *Chapter 1.4.1.20.3.21.9 “Command 'Mouse Zooming'” on page 1141*
- *Chapter 1.4.1.20.3.21.14 “Command 'Reset View'” on page 1144*
- *Chapter 1.4.1.20.3.21.2 “Command 'AutoFit'” on page 1137*
- *Chapter 1.4.1.20.3.21.3 “Command 'Compress'” on page 1137*
- *Chapter 1.4.1.20.3.21.18 “Command 'Stretch'” on page 1146*
- *Chapter 1.4.1.20.3.21.10 “Command 'Convert to Multi-Channel'” on page 1141*
- *Chapter 1.4.1.20.3.21.11 “Command 'Convert to Single-Channel'” on page 1142*

Managing trace

Use menu commands to load and save traces in various formats.

Menu commands

- *“Trace ➔ Save Trace”*
- *“Trace ➔ Load Trace”*
- *“Trace ➔ Export Symbolic Trace Config”*

See also

- *Chapter 1.4.1.20.3.21.15 “Command 'Save Trace'” on page 1145*
- *Chapter 1.4.1.20.3.21.8 “Command 'Load Trace'” on page 1141*
- *Chapter 1.4.1.20.3.21.7 “Command 'Export Symbolic Trace Config'” on page 1139*

Showing statistics

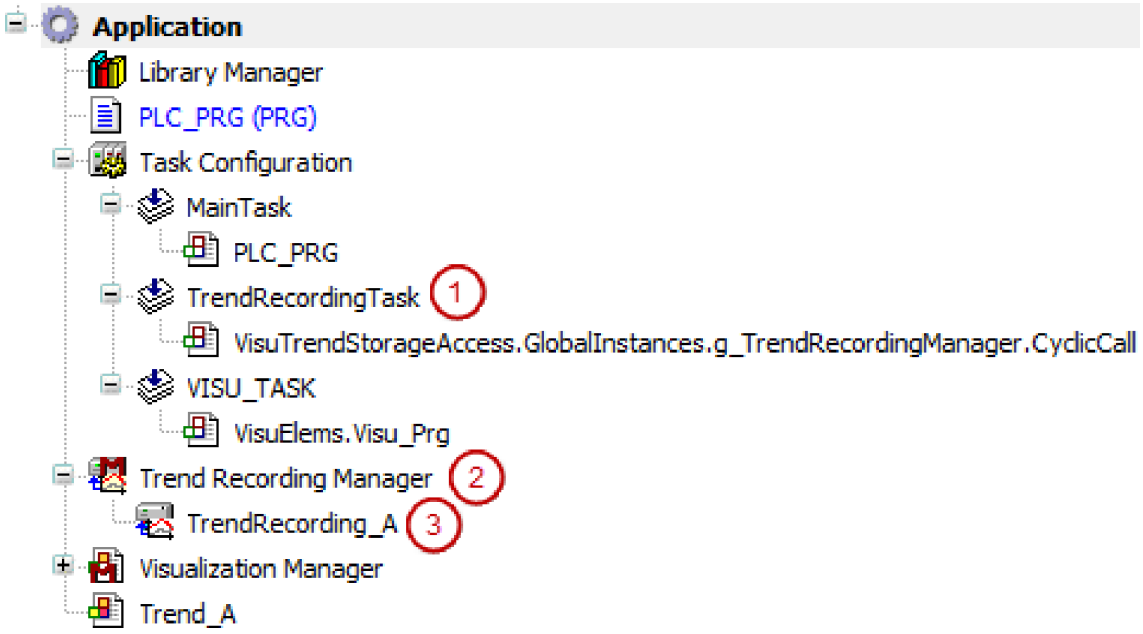
CODESYS evaluates and displays the recorded data with an option of saving the data to the clipboard. Click *“Trace ➔ Statistics”*.

See also

- *Chapter 1.4.1.20.3.21.20 “Command 'Statistics'” on page 1146*

1.4.1.12.4 Data Recording with Trend

When you want to monitor the development of data over a long period of time for the purpose of reading a trend, you can save the data with *“Trend Recording”*. You can configure any number of variables or parameters to save their values in a persistent database. This database is located on the PLC and is populated continually at runtime.



Trend recording comprises the following objects:

- (1):*"Trend recording task"* of type *"Task"*
- (2): Object of type *"Trend Recording Manager"*
- (3): Object of type *"Trend Recording"*

NOTICE!

Timeout for trend recording

During a trend recording, it can happen that the application task triggers a timeout that is caught with an exception when transitioning from *"Running"* to *"Stop"*. Causes can be that file operations with the SQLite database are taking too long or that too many variables are being recorded. This usually happens on a target device with weak performance.

You can avoid the occurrence of an exception:

– Configure the trend recording with less memory demand so that the amount of data that is stored is adapted to the target system.
– Reduce the number of variables.

To display the collected data, you design a visualization with a "Trend" element. This kind of visualization accesses the database for visualizing the data.

See also

- *Chapter 1.4.1.20.2.30 "Object 'Trend Recording Manager'" on page 949*
- *Chapter 1.4.1.20.2.31 "Object 'Trend Recording'" on page 949*
- *Chapter 1.4.1.20.2.32 "Object 'Trend Recording Task'" on page 952*

Getting started with trend recording

To execute trend recording on a runtime system, you need an application with a *"Trend Recording Manager"* object that contains at least one *"Trend Recording"* object. Then you can configure a database on the runtime system and the data buffering.

1. Add a *"Trend Recording Manager"* object below your application.
2. Select the *"Trend Recording Manager"* object and click *"Add Object ➔ Trend Recording"*. Type a name in the *"Add Trend Recording"* dialog box.
   ⇨ CODESYS creates the object. The editor opens.
3. Type a task in *"Record Settings"*.
4. Click *"Add Variable"*.
   ⇨ CODESYS adds another variables. The blank settings open in the *"Variable Settings"* to the right of the tree view.
5. Select a valid IEC variable from the *"Variable"* field.
   ⇨ The IEC variable is configured for trend recording.
6. Build the application.
7. Download the application to the controller and click *"Start"*.
   ⇨ The application records data in runtime mode and saves it to a database.

See also

- *Chapter 1.4.1.20.2.31 "Object 'Trend Recording'" on page 949*
- *Chapter 1.4.5.11.1 "Getting Started with Trend Visualization" on page 1309*

Configuring trend recording

You can configure a database on the runtime system and the data buffering.

NOTICE!
Timeout for trend recording
During a trend recording, it can happen that the application task triggers a timeout that is caught with an exception when transitioning from *"Running"* to *"Stop"*. Causes can be that file operations with the SQLite database are taking too long or that too many variables are being recorded. This usually happens on a target device with weak performance.

You can avoid the occurrence of an exception:

– Configure the trend recording with less memory demand so that the amount of data that is stored is adapted to the target system.
– Reduce the number of variables.

Assigning tasks

In this task, the runtime system records the trend.

In general, trend recording runs in the same task as the main program (for example, `PLC_PRG`).

1. Double-click a *"Trend Recording"* object in the device tree.
   ⇨ The respective editor opens. In the tree view of the trend configuration, the top entry is selected, and on the right you see the current configuration in *"Record Settings"*.
2. Click the "arrow down" symbol (▾) in the *"Task"* drop-down list.
   ⇨ The drop-down list opens with all tasks that are available throughout the application.
3. Select a task for trend recording.

See also

- ↳ *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*

Adding IEC variables

> **NOTICE!**
> The number of variables is limited for trend recording. You can change this number in the *“Trend storage”* dialog.

1. Double-click a *“Trend Recording”* object in the device tree.
   ⇨ The respective editor opens. In the tree view of the trend configuration, the top entry is selected, and on the right you see the current configuration in *“Record Settings”*.
2. Right-click an entry in the tree view.
3. Click *“Add Variable”*.
   ⇨ CODESYS adds another variables. The blank settings open in the *“Variable Settings”* to the right of the tree view.
4. Select a valid IEC variable from the *“Variable”* field.
   ⇨ The IEC variable is configured for trend recording.
5. Configure how the variable is displayed in the trend diagram.
6. Configure how the alert color is displayed in the trend diagram.

See also

- ↳ *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*
- ↳ *Chapter 1.4.1.20.4.16 “Dialog Box 'Trend storage'” on page 1214*

Removing variables from the configuration

1. Double-click a *“Trend Recording”* object in the device tree.
2. Click a variable in the tree view of the configuration.
3. Click *“Delete Variable”* or press *[Del]*.

Starting conditional trend recording

You can configure conditional trend recording for execution. Configuration is not possible when depending on triggering. For that you need a *“Trace”* object.

1. Double-click a *“Trend Recording”* object in the device tree.
2. Click the top node in the tree view of the trend configuration.
   ⇨ The name of the trend configuration is selected and on the right you see the current configuration in *“Record Settings”*.
3. Assign a Boolean variable, an access to a bit, or a property to the *“Record condition”* field.
   ⇨ When the application is in runtime mode, data is recorded only if the value is `TRUE`.

See also

- ↳ *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*

Adding parameter

1. Double-click a *“Trend Recording”* object in the device tree.
   ⇨ The respective editor opens. In the tree view of the trend configuration, the top entry is selected, and on the right you see the current configuration in *“Record Settings”*.
2. Right-click an entry in the tree view.
3. Click *“Add Variable”*.
   ⇨ CODESYS inserts a new variable. The blank settings open in the *“Variable Settings”* to the right of the tree view.
4. Click the "down" symbol (▼) to the right of the *“Variable”* label.
5. Select *“Parameter”* from the drop-down list.
6. Click ... and select a parameter from the *“Input Assistant”* dialog.
7. Configure how the parameter is displayed in the trend diagram.
8. Configure how the alert color is displayed in the trend diagram.

See also

- *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*

Configuring data buffering on the RTS

1. Double-click a *“Trend Recording”* object in the device tree.
   ⇨ The respective editor opens. In the tree view of the trend configuration, the top entry is selected, and on the right you see the current configuration in *“Record Settings”*.
2. Click *“Trend Storage”*.
   ⇨ The *“Trend Storage”* dialog opens.
3. Now you can change the settings.

See also

- *Chapter 1.4.1.20.4.16 “Dialog Box 'Trend storage'” on page 1214*

Configuring additional buffering

1. Double-click a *“Trend Recording”* object in the device tree.
   ⇨ The respective editor opens. In the tree view of the trend configuration, the top entry is selected, and on the right you see the current configuration in *“Record Settings”*.
2. Click *“Advanced”*.
   ⇨ The *“Advanced Trace Settings”* dialog opens.
3. Now you can change the settings.

See also

- *Chapter 1.4.1.20.4.17 “Dialog Box 'Advanced Trend Settings'” on page 1214*

See also

- *Chapter 1.4.1.12.3.2 “Creating trace configuration” on page 424*

1.4.1.12.5 Monitoring tasks

In online mode, you can display some statistical values of the tasks in the runtime system. This information is very useful for testing clock cycles or solving problems in the runtime performance.

1. Switch to online mode.
2. Select the *“Task Configuration”* object in the device tree.

   Click *“Project ➔ Edit Object”*.

   ⇨ The task configuration opens in the editor.
3. Click the *“Monitor”* tab.

See also

- *Chapter 1.4.1.20.2.26.3 “Tab 'Monitor'” on page 940*

1.4.1.12.6 Reading the PLC log

CODESYS provides the capability to display the events and error messages logged in the controller.

See also

- *Chapter 1.4.1.20.2.8.8 “Tab 'Log'” on page 848*

Reading the log

Requirement: The controller is running.

1. Select the controller in the device tree.
2. Choose the command *“Project ➔ Edit Object”*.

   ⇨ The device editor opens.
3. Choose the tab *“Log”*.
4. Click on to update the view.

   ⇨ A connection to the controller is established. The controller in the device tree is highlighted in green.

   All controller log information are displayed.
5. Click on to delete the current list.
6. Filter the view by clicking on the desired category (for example "Information").
7. Save the log entries. Click on and choose a file name.

See also

- *Chapter 1.4.1.20.2.8.8 “Tab 'Log'” on page 848*

1.4.1.12.7 Using PLC shell for requesting information

The "PLC shell" in CODESYS is a text-based control monitor (terminal) on a tab of the device editor. There you can enter commands for the request of specific information from the controller, as well as execute actions like starting, stopping or downloading applications. Also a description on the meaning and syntax of the possible commands you can get directly via the PLC shell.

Requesting information about the application on the controller

Requirement: Your project is connected with a controller; Example: CODESYS Control Win V3, on which an application `App1` is running.

1. Open the device editor double-clicking on the object CODESYS Control Win V3 in the device tree, and activate tab *"PLC Shell"*.
   ⇨ The tab shows an empty output data window. Below there is an entry field for a command.
2. Click button [...].
   ⇨ The *"Insert Standard Command"* dialog appears with a list of commands.
3. Choose command *"?"* and click button *"Execute"*.
   ⇨ The dialog closes and in the output data window you see a list of the supported commands and their possible parameters. Each the syntax for how to enter the command is displayed.
4. Click again [...] and choose command *"pid"*. In the input assistant supplement the command as follows: `pid App1`. Press the Enter key.
   ⇨ In the output data window the following gets displayed (the GUIDs are just exmples):

```
pid App1
Project Identification
Application: App1
Code GUID:0x08a893c0
Data GUID:0x762d0e90
```

5. Click button ▼ in the command line.
   ⇨ Command `pid App1` is added to the history of already entered commands.

See also

- *Chapter 1.4.1.20.2.8.10 "Tab 'PLC Shell'" on page 852*

1.4.1.12.8 PLC operation control via system variables

CAUTION!
You are responsible for runtime system services being enabled under safe application conditions and disabled only under critical conditions.

At runtime, the state of an application or facility can become sensitive and disruptive actions can endanger the entire machine or facility. However, in this state you can suppress certain commands and prevent dangerous actions. The *"PlcOperationControl"* function block and *"Component Manager"* library are provided for this purpose.

Examples of CODESYS commands that can suppress operations when executed:

- *“Online Change”*, *“Download”*
- *“Enable Breakpoint”*
- *“Reset Application”*, *“Stop Application”*
- *“Transmit Data”*
- *“Force Values”*, *“Write Values”*

In order that a backup solution is always in place, you are not permitted to suppress the *“Reset origin”* and *“Delete”* commands.

CODESYS will notify you if a currently disabled runtime system service is required when the application is in runtime mode. Then, you can respond with an appropriate countermeasure.

Function block `PlcOperationControl` for operation control

This function block is used for enabling and disabling operations.

Table 29: Property (`PROPERTY`)

Name	Data Type	Initial value	Description
`xDisableApplicationOnlineChange`	`BOOL`	`FALSE`	`TRUE`: Online change is suppressed.
`xDisableApplicationDownload`	`BOOL`	`FALSE`	`TRUE`: Download is suppressed.
`xDisableApplicationStop`	`BOOL`	`FALSE`	`TRUE`: Application stop is suppressed.
`xDisableApplicationBP`	`BOOL`	`FALSE`	`TRUE`: Setting breakpoints is suppressed.
`xDisableApplicationWrite`	`BOOL`	`FALSE`	`TRUE`: Writing variables is suppressed. This can also be suppressed via PLCHandler/IecVarAccess.
`xDisableApplicationForce`	`BOOL`	`FALSE`	`TRUE`: Forcing variables is suppressed.
`xDisableApplicationReset`	`BOOL`	`FALSE`	`TRUE`: Resetting the application (not "Reset origin") is suppressed.
`xDisableAll`	`BOOL`	`FALSE`	`TRUE`: All operations are suppressed.

Implementing operation control

Requirement

- Compiler version >= `3.4.3.0`
- In the device description, the PLC operation control is enabled by system variables.

1. Declare an instance of the `PlcOperationControl` function block (for example, `PlcOpCtrl_Inst`).

   ⇨ `PlcOpCtrl_Inst : PlcOperationControl;`

2. Suppress a command by assigning the respective `TRUE` property (for example, "Stop Application".

   ⇨ `PlcOpCtrl_Inst.xDIsableApplicationStop := TRUE;`

See also

- *Chapter 1.4.1.20.3.6.6 “Command 'Online Change'” on page 1033*
- *Chapter 1.4.1.20.3.7.16 “Command 'Force Values'” on page 1053*
  Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401

1.4.1.12.9 Backup and restore

CODESYS and the CODESYS standard runtime systems (with version 3.5.8.0 and later) support backing up application-specific files on the PLC. You can execute the required actions in the *“Backup and Restore”* tab of the generic device editor.

A backup consists of creating and saving a file in zip archive that contains the application-related files and an information file `meta.info`. This backup file has the extension `TBF` (="Target Backup File") and can be saved in the local file system or on the PLC.

The following applies when restoring the software status from the backup file:

- A dialog opens with a list of affected files on the PLC, and you can deactivate optional components.
- We highly recommend to set the application to STOP mode for backup or restore. A dialog prompt will open to warn you about this.
- The user interface is blocked when restoring to the PLC.
- Existing files are overwritten without warning.
- Existing boot applications are deactivated as soon as at least one new boot application is part of the restore.

See also

- *Chapter 1.4.1.20.2.8.5 “Tab 'Backup and Restore'” on page 846*

Creating backup files

Requirement: A project is open with an application that is running on the required device. In addition, for this example an external file `myExternalFile.txt` is inserted as an object below the application. This file is downloaded to the PLC implicitly when downloading the application.

1. Open the device editor by double-clicking the device entry in the device tree. Click the *“Backup and Restore”* tab.

   ⇨ The tab opens with a menu bar including the *“Backup”* and *“Restore”* menus.

2. In the *“Backup”* menu, select the *“Read Backup Information from Device”* item.

   ⇨ If the PLC is not connected at the moment, then CODESYS opens a temporary connection to the device and reads the relevant files from the `$PlcLogic$` directory of the PLC into a table in the lower part of the tabbed page. In this example, at least the following files will be listed: `$PlcLogic$/Application/Application.app`, `$PlcLogic$/Application/Application.crc`, and `$PlcLogic$/Application/myExternalFile.txt`. In addition, other external, project-dependent files are listed, which have been inserted below the application in the device tree. Furthermore, the source code archive file `$PlcLogic$/Archive.prj` is listed if you have set the project setting for this (*“Implicitly at program download and online change”*) as the loading time.

3. In the table, clear the check box for the `$PlcLogic$/Application//myExternalFile.txt` file in the *“Active”* column.

4. Select *“Save Backup File to Device”* in the *“Backup”* menu.

   ⇨ The *“Save as”* dialog opens. The file type is predefine as *“Backup files (*.tbf)”*.

5. Select a location for the backup file and click *“Save”*.

See also

- *Chapter 1.4.1.20.4.11.5 “Dialog 'Project Settings' – 'Source Download'” on page 1174*

Restoring from backup files

Requirement: A project is open with a PLC device connected. A backup file is saved to the local file system as described above.

1. Open the device editor by double-clicking the device entry in the device tree. Click the *"Backup and Restore"* tab.

   Click *"Restore ➔ Load Backup File from Disc"*.

   ⇨ The default prompt opens for selecting a backup file `tbf` in the local file system.

2. Select the backup file and click *"Open"*.

   ⇨ The files are read from the backup file and shown in the table of the dialog below. The file `$PlcLogic$/Application/myExternalFile.txt` that was excluded in the backup is missing.

3. Click *"Restore ➔ Restore Backup to Device"*.

   ⇨ A dialog prompt opens with information about the actions when restoring.

4. Click *"OK"* to start restoring the files to the PLC file system.

   ⇨ When restore is complete, you are prompted to restart the PLC in order to activate the loaded application.

1.4.1.13 Updating an Application on the PLC

CODESYS basically provides two options to transfer a modified application to the controller: download and online change.

A download results in a recompilation of the application. In that time, a syntax check is performed and application code is also created and downloaded to the controller. This leads to the running program being stopped. A download is the recommended method of data transfer because a defined starting state is always created due to the program stop and the reinitialization.

In the case of an online change, only the modified parts are downloaded again to the controller. A running program is not stopped for this. You should perform an online change only in the case of minor changes to the application. For extensive changes, the behavior of a program cannot be safely predicted. For more information, read the notes in the description of the *"Online Change"* command.

See also

- *Chapter 1.4.1.13.1 "Executing the online change" on page 439*
- *Chapter 1.4.1.13.2 "Execution of a download" on page 440*
- *Chapter 1.4.1.20.3.6.5 "Command 'Load'" on page 1032*
- *Chapter 1.4.1.20.3.6.6 "Command 'Online Change'" on page 1033*

1.4.1.13.1 Executing the online change

CODESYS automatically offers you an online change if you log in with an application that is already present on the controller, but has been changed since the last download in the programming system. With this procedure only the modified parts are reloaded to the controller. A running program on the controller is not stopped during the online change.

In the view *"Memory reserve for online change"*, you can configure memory reserves for the online change for function blocks of a project. In this way, instance variables do not have to be moved to the memory after changes are made to a function block for an online change.

NOTICE!

When carrying out the online change, pay attention to the notes in the description of the *"Online Change"* command.

See also

- ↳ *Chapter 1.4.1.20.3.6.6 “Command 'Online Change'” on page 1033*
- ↳ *Chapter 1.4.1.20.3.6.2 “Command 'Login'” on page 1028*
- ↳ *Chapter 1.4.1.10.4 “Generating Application Code” on page 389*

Executing the online change when logging in

Requirement: The connection settings of the controller are correctly set. The applications in the project and on the controller are identical. The project on the controller is running. The application is logged out.

1. Change your application.
2. Click *“Online ➔ Login”*.
   ⇨ A dialog appears with the information that the application has been changed since the last download.
3. Click the *“Details…”* button
4. Check the details in the *“Application information”* tab.
   If you have not generated any code since the last change, the command *“Application is not up to date. Generate code now?”* appears at the bottom edge of the dialog. In this case click this command.
   ⇨ You are shown a comparison view of the objects (objects marked red are different).
5. Close the dialog.
6. Select the option *“Login with Online Change”* and click *“OK”*.
   ⇨ The change is loaded to the controller. The running program on the controller is not stopped while doing this. The application is logged in.

See also

- ↳ *“View 'Project Comparison' - 'Differences'” on page 1011*

Execute online change in the logged-in state (online operation)

Requirement: The connection settings of the controller are correctly set. The applications in the project and on the controller are identical. The project on the controller is running. The application is logged in.

1. Select an object in the device tree. It is best to select a POU or a GVL here.
2. Click *“Project ➔ Edit Object (Offline)”*.
   ⇨ The object opens in the editor.
3. Change the object. For example, you can declare a new variable or change a value assignment here.
4. Click *“Online ➔ Online Change”*.
   ⇨ A query will appear, asking whether you really want to execute the online change.
5. Click *“Yes”* to confirm the dialog.
   ⇨ The change is loaded to the controller.

1.4.1.13.2 Execution of a download

A download of the application causes a compilation of the active application. In the process, a syntax check is performed and application code is also created and loaded to the controller. A program running on the controller is stopped during the download.

> **NOTICE!**
> During the download, pay attention to the notes in the description of the *"Download"* command.

See also

- ↳ *Chapter 1.4.1.20.3.6.5 "Command 'Load'" on page 1032*
- ↳ *Chapter 1.4.1.20.3.6.2 "Command 'Login'" on page 1028*
- ↳ *Chapter 1.4.1.10.4 "Generating Application Code" on page 389*

Downloading when logging in

Requirement: the connection settings of the controller are correctly set. The applications in the project and on the controller are identical. The project on the controller is running. The application is logged out.

1. Change your application.
2. Select the command *"Online ➔ Login"*
   ⇨ A dialog box appears with the information that the application has been changed since the last download.
3. Select the option *"Login with download"* and click on *"OK"*.
   ⇨ The running program on the controller is stopped and the change is loaded to the controller. The application is logged in.

Downloading in the logged-in state (online mode)

Requirement: the connection settings of the controller are correctly set. The applications in the project and on the controller are identical. The project on the controller is running. The application is logged in.

1. Select an object in the device tree. It is best to select a POU or a GVL here.
2. Select the command *"Project ➔ Edit Object (Offline)"*
   ⇨ The object opens in the editor.
3. Change the object. For example, you can declare a new variable or change a value assignment here.
4. Select the command *"Online ➔ Download"*.
   ⇨ A query will appear, asking whether you really want to execute the download.
5. Confirm the dialog box with *"Yes"*.
   ⇨ The running program on the controller is stopped and the change is loaded to the controller.

1.4.1.14 Copying files to/from PLC

In the generic *"Files"* tab of the device editor, you can copy files to and from the local file system and the controller.

Requirement: The vendor has unlocked the tab. In the device tree, the connection to the controller is configured. The device is running.

1. Double-click the PLC device object in the device tree to open the device editor.
2. Click the *"Files"* tab.

3. In *“Host”* | *“Location”* on the left part of the view, set the path in the local file system where files will be copied to and from. Example: `D:\FileTransferWithPLC`. If necessary, create a new directory by clicking the folder symbol ( ).

   ⇨ The files and directories are shown like in a file manager. Click the refresh symbol ( ) to update the display.

4. In *“Runtime”* on the right side of the view, set the required directory for the data transfer in the same way.

   ⇨ CODESYS shows the files on the controller.

5. Select the required files from the file system tree for the file transfer (multiple selection is possible). You can also select a directory for transferring all files in a folder.

6. Click the left and right arrow symbols (<< , >>) between the two parts of the view.

   ⇨ CODESYS copies the selected files to the other file system immediately. If a file is not already available in the target directory, then it is created. If it is already available and not write-protected, then it is overwritten. Otherwise a message is shown.

See also

- *Chapter 1.4.1.20.2.8.7 “Tab 'Files'” on page 848*

1.4.1.15 Using the Command-Line Interface

You can start the command line with the following options and arguments.

Syntax: `<folder>Automation Builder.exe --<option>`

> *Paths or option parameters must be written inside straight quotation marks when they contain spaces, dashes, or slash marks.*

Option `--culture` (language of the user interface)

CODESYS is started in the specified language.

Syntax: `--culture=<culture>`

`<Culture>`: Typical language codes are as follows: de, en, fr, it, es, zh-CHS.

Example Starting CODESYS with the user interface in English:

```
Automation Builder.exe --culture=en
```

See also

- *Chapter 1.4.1.20.4.13.13 “Dialog 'Options' – 'International Settings'” on page 1195*

Option `--profile` (CODESYS profile)

CODESYS is started directly with the specified profile. When you start CODESYS without this option, the *“Select Profile”* opens.

Syntax: `--profile="<profile name>"`

`<profile name>`: You have to specify the profile name exactly as it is displayed in the *“Help ➔ About”* splash screen of the development system or in the start menu on your computer.

Example

```
Automation Builder.exe --culture=de --profile="Automation Builder
2.5"
```

Option `--compare` (start project comparison)

After CODESYS is started, the comparison of two CODESYS projects is begun immediately. Type the path of the project file as arguments after the option and then the path of the reference project. CODESYS starts and opens the *"Project Comparison - Differences"* view.

Syntax:

```
--compare="<path of project file>" "<path of reference project file>"
```

Example

```
Automation Builder.exe --compare "D:\proj\project1.project"
"D:\proj\project2.project"
```

See also

- *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*

Option `--project` (open CODESYS project)

CODESYS is started and the specified project is opened.

Syntax:

```
--project="<path of project file>"
```

`<path of project file>`: File path of project

Example

Open the `test` project:

```
Automation Builder.exe --culture=de --
project="D:\projects\test.project"
```

See also

- *Chapter 1.4.1.20.3.1.2 "Command 'Open Project'" on page 957*

Option `--projectarchive` (open CODESYS project archive)

CODESYS is started, the specified project archive is extracted, and the project is opened.

Syntax:

```
--projectarchive="<path of project archive file>"
```

`<path of project archive file>`: File path of project archive

Example

Extract the `test.projectarchive` and open the project in the development system:

```
Automation Builder.exe --
projectarchive="D:\projects\test.projectarchive"
```

See also

- *Chapter 1.4.1.20.3.1.9 "Command 'Extract Archive'" on page 961*

Option `--runscript` (run script) The specified script file is run by CODESYS.

Table 30: Command-line options for `--runscript`

`--runscript="<scriptfile>.py"`	CODESYS runs the `<scriptfile>.py` script file at startup. You have to provide the complete path of the script file.
`--scriptargs:'<arg1> <arg2> ... <argn>'`	Use this option with the `--runscript` option. As a result, the arguments <arg1> ... <argn> are passed to the script. The arguments are passed to the Python variable `sys.argv`.
`--noUI`	Use this option with the `--runscript` option. The CODESYS user interface is not opened. CODESYS prints all errors, warnings, compiler reports, and command-line messages generated from the script. The script messages (1: Severity Text) can be separated from other messages (2: Severity FatalError, Error, Warning, Information) with the ">" operator.
`--enablescripttracing`	Use this option with the `--runscript` option. As a result, each command of the script file is shown in the output.
`--textPrompts`	Use this option with the `--noUI` option. As a result, message service methods and default dialogs are output in the command line for user input. If you do not specify `--textPrompts`, then all message service prompts are confirmed automatically with default values.
`scriptdebugger {="<debugger>"}`	Use this option with the `--runscript` option. It sets IronPython in debug mode so that external debuggers can be used to debug Python scripts. The following values are defined for <debugger> (uppercase/lowercase is irrelevant). ● `auto`: Automatically detects if a debugger is included in every script for the current process. At this time, only .NET-based debuggers can be detected automatically. A detected debugger overwrites the `--enablescripttracing` flag. ● `.NET`: Activates debugging for .NET-based debuggers, such as "Python Tools for Visual Studio" (PTVS) and SharpDevelop. With this option, a debugger can also be included in running scripts, as opposed to "auto". Note: This is currently the default value when `--scriptdebugger` is used without providing a value. ● `disabled`: Deactivates debugging and automatic detection. ● `script`: Switches the IronPython script engine to debug mode for activating the debugging for set-trace debuggers. The script itself must connect to and disconnect from the debugger. ● `tracing`: Activates the simple integrated script tracing mode and deactivates the automatic detection (same as the option `--scripttracing`). ● `$absolute_path.py$`: Absolute path to a Python script that initiates the connection to a Python-based debugger. The IronPython script engine is switched to debug mode for allowing the debugging for set-trace debuggers. This script is run one time during the initialization and should define the following non-parameterized functions: `scriptdebuggersetup` is run immediately before running the user script to establish the connection to the debugger. `scriptdebuggershutdown` is called immediately after running the user script or when the script engine is downloaded and should close the connection to the debugger.

Examples of using transfer parameters in script files with 'sys.argv'

```
start /b /wait Automation Builder.exe
--runscript="D:\Script\ArgvAnd__main__Test.py"
--scriptargs:'username password 3.14 "path=\"C:\temp\\"'
```

Script file: `ArgvAnd__main__Test.py`

```
from __future__ import print_function
import sys
print("sys.argv: ", len(sys.argv), " elements:")

for arg in sys.argv:
    print(" - ", arg)
print()
print("__name__: ", __name__)
```

Output result: `stdout`:

```
sys.argv: 6 elements:
- D:\TestScripts\ArgvAnd__main__Test.py
- username
- password
- 3.14
- path= "C:temp"
__name__: __main__
```

For more information about the `__name__` global variable, see the Python documentation.

Examples of the message output

```
start /b /wait Automation Builder.exe --
runscript="D:\Script\AmpelTest.py" --noUI 1>ScriptMessages.txt
```

CODESYS passes all messages that are generated by the script to the `ScriptMessages.txt` file. Other messages are printed to the command line.

```
start /b /wait Automation Builder.exe --
runscript="D:\Script\AmpelTest.py" --noUI 2>NUL
```

CODESYS suppresses all messages, except for script messages. The script messages are printed to the command line.

Example of option --scriptdebugger

The following `initdebug.py` script was tested successfully with `pydevd`-based debuggers, such as PyDev / LiClipse and PyCharm. To use this script, start CODESYS with the following command line:

```
--profile="Fanta Development Build" --
scriptdebugger="D:\test\charmdebug\initdebug.py"
```

File: `initdebug.py`:

```
from _future_ import print_function
from _future_ import unicode_literals
import sys
sys.path.append(r"D:\test\Env2\Lib\site-packages\pycharm-debug.egg")
import pydevd
def scriptdebuggersetup():
pydevd.settrace('localhost', port=51234, stdoutToServer=True,
stderrToServer=True)
def scriptdebuggershutdown():
pydevd.stoptrace()
```

See also

- *http://docs.python.org/tutorial/modules.html*

Option `--ignorewhitespace` (ignore whitespace in project comparison)

If you add this option after the option `--compare <project1> <project2>`, then white-space is ignored in the project comparison. Note that semantically relevant spaces, for example in `STRING` literals, are still taken into account.

Syntax

```
--compare="<path of project file>" "<path of reference project file>"
--ignorewhitespace="true"|"false"
```

Example

```
Automation Builder.exe --compare "D:\proj\project1.project"
"D:\proj\project2.project" --ignorewhitespace="true"
```

See also

- *Chapter 1.4.1.20.3.4.21 “Command 'Compare'” on page 1010*

Option `--ignorecomments` (ignore comments in project comparison)

If you add this option after the option `--compare <project1> <project2>`, then comments are ignored in the project comparison.

Syntax:

```
--compare="<path of project file>" "<path of reference project file>"
--ignorecomments="true"|"false"
```

Example

```
Automation Builder.exe --compare "D:\proj\project1.project"
"D:\proj\project2.project" --ignorecomments="true"
```

See also

- *Chapter 1.4.1.20.3.4.21 “Command 'Compare'” on page 1010*

Option `--ignoreproperties` (ignore object properties in project comparison)

If you add this option after the option `--compare <project1> <project2>`, then object properties (permissions, compile settings, directories, bitmaps, etc.) are ignored in the project comparison.

Syntax:

```
--compare="<path of project file>" "<path of reference project file>"
--ignoreproperties="true"|"false"
```

Example

```
Automation Builder.exe --compare "D:\proj\project1.project"
"D:\proj\project2.project" --ignoreproperties="true"
```

See also

- *Chapter 1.4.1.20.3.4.21 “Command 'Compare'” on page 1010*

Option `--skipunlicensedplugins` (do not load components without a license)

CODESYS is started. In this case, the query as to whether unlicensed components should still be loaded is skipped. If so, then CODESYS does **not** load these components by implication.

Example

```
Automation Builder.exe --skipunlicensedplugins
```

Option `--signaturethumbprint` (thumbprint of the certificate which is used for signing compiled libraries)

If you add this option after the option `--project="<path of project file>"`, then the project is opened and the thumbprint of the certificate for signing compiled libraries is passed.

Syntax:

```
--signaturethumbprint="<thumbprint of digital signature>"
```

Example

```
Automation Builder.exe --project="D:\projects\test.project"
signaturethumbprint="3E96C9B61010CBDC3186021A1CAA64946DDCAAF3"
```

See also

- *Chapter 1.4.1.20.3.3.18 “Command 'Security Screen'” on page 995*

Option `--enforcesignedcompiledlibraries` (enforce signing of compiled libraries)

If you add this option after the option `--project="<path of project file>"`, then the *“Enforce signing of compiled libraries”* option is enabled in the project in the *“Security Screen”* on the *“User”* tab.

NOTICE!
When the *“Security Screen”* is opened and closed, the current settings are applied in the user options, even when no active changes have been made.

Syntax:

```
--enforcesignedcompiledlibraries="true"|"false"
```

Example

```
Automation Builder.exe --project="D:\projects\test.library" --
enforcesignedcompiledlibraries="true"
```

See also

- *Chapter 1.4.1.20.3.3.18 “Command 'Security Screen'” on page 995*

Option `--timestampingserverurl` (set the time stamp server address)

If you add this option after the option `--project="<path of project file>"`, then the Internet address of the RFC-3161 time stamp server (*“Timestamping server”*) is set in the project in the *“Security Screen”* on the *“User”* tab.

NOTICE!
When the *“Security Screen”* is opened and closed, the current settings are applied in the user options, even when no active changes have been made.

Syntax: `--timestampingserverurl="<URL of RFC-3161 timestamping server>"`

Example

```
Automation Builder.exe --timestampingserverurl="http://
timestamp.comodoca.com/rfc3161"
```

See also

- *Chapter 1.4.1.20.3.3.18 “Command 'Security Screen'” on page 995*

1.4.1.16 Using Libraries

Library repository

The library repository is the storage location on the development system for libraries and associated metadata. You can link any installed the libraries into your project by means of a library manager. Moreover, the libraries are installed with version management for easy library updates.

You can create and edit more repositories in addition to the preinstalled `System` repository.

See also

- *Chapter 1.4.1.20.3.8.5 “Command 'Library Repository'” on page 1061*

Library Manager

In order to be able to use POUs, which are provided in a library POU, in the application, the library has to be integrated in the Library Manager in the project. The requirement for this is the installation of the library in the library repository.

The Library Manager displays all integrated libraries according to their library type and the respective properties. In the Library Manager, you can add more libraries from the library repository, remove libraries, and edit library properties.

The Library Manager can be inserted into the *“POUs”* view or the *“Devices”* view. As a result, a project can have a Library Manager for each application, as well as a Library Manager in the *“POUs”* view for use across all applications. The library POUs of the integrated libraries in the *“POUs”* view can be called regardless of the application. The library POUs of the integrated libraries in the *“Devices”* view can be called in the respective application code only. Furthermore, placeholder libraries behave differently when downloading depending on their positions.

Libraries that are integrated to a specific version in the project also have a placeholder for that version (placeholder library). You can define special placeholder resolutions. You can also use the placeholder resolution that is defined for a device in the device description or that is stored in the library repository for a library. The Library Manager notifies you about the actual placeholder resolution and shows the version that will be loaded when an application is downloaded (effective version).

When a Library Manager in the *“POUs”* view is integrated across all applications, you can access its contents globally. If placeholder libraries are integrated, then only the placeholder resolutions in the device description or library repository are checked.

A Library Manager is usually integrated in the *“Devices”* view. Then only the application code below it calls library POUs from it. Moreover, the special placeholder resolutions are checked first for placeholder libraries. Only after that are the placeholder resolutions checked that are in the device description or that originate from the library repository.

See also

- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*
- *Chapter 1.4.1.8.7 “Using Library POUs” on page 265*

1.4.1.16.1 Information for Library Developers

In order to avoid consistency problems and to adequately support the user, be sure to adhere to certain rules for the creation, referencing, encryption, protection, and documentation of libraries.

The following description provides only an overview of the library development possibilities. For a more detailed description of these topics, see the "LibDev-Summary" guidelines for library development.

See also

- *Chapter 1.4.1.16 “Using Libraries” on page 448*

General

- You can define categories for libraries. The libraries are then displayed in the library repository below these categories.
- You can define a namespace for a library in order to enable unambiguous access to the integrated objects. The access becomes unambiguous by adding the namespace in front of the POU name:
  <namespace>.<variable name>
  Example: `AC.Module`
- You can open the POUs of unencrypted libraries (`*.library`) by double-clicking the respective entry in the Library Manager.

- You can create the following library types:
  - `*.library`: Implementation library (source code of the library)
  - `*.compiled-library`, `*.compiled-library-v3`: Protected library; source code no longer accessible.
  - `*_Itfs.library`: Interface library; contains only objects that are used for the interface definition of a component (for example, constants, structures, or interfaces) and do not generate any code.
  - `*_Cnt.library`: Container library; does not contain any POUs; instead contains exclusively other libraries; therefore used to conveniently integrate an entire set of libraries whose POUs are published on the top level of the container library.
- You can integrate external libraries into the application. External libraries are programmed outside of CODESYS in a different programming language, for example C.

Protection of libraries

- Source code protection:
  When a library is prepared in "compiled-library" format, the source code of the library POUs is no longer visible after the library is integrated into a project.
- Signing:
  In CODESYS V3 SP15 and higher, a certificate is always used for the signing of library projects (`*.compiled-library-v3`). The signing can be enforced by means of a setting in the security screen. Then for generating a compiled library, you need a certificate suitable for code signing in your user profile.
  For library projects that have to be compatible with CODESYS < V3 SP15 (`*.compiled-library`), only the less safe signing is possible with a private key and a corresponding token. These deprecated methods should only be used for reasons of compatibility. Settings are configured in the *“Project Information”* on the *“Signing”* tab.
  Note: For signing libraries, you should use compiler version 3.5.15.0 or higher because a better storage format is used.
- Licensing:
  You can protect libraries by means of a license (dongle or soft container). License-protected libraries can be installed in the library repository. However, for use in the project, the valid license has to exist on the computer. Licenses are managed in the License Manager.

Library versions

- You can have several versions of a library installed on the system at the same time.
- You can have several versions of a library integrated into your project at the same time. However, we do not recommend doing this. In this case, each of the libraries **must** be assigned a unique namespace and access to the symbols **must** be qualified. Examples: `V1.SendBlob`, `V2.SendBlob`

Referenced libraries

- You can integrate a library into other libraries (referenced libraries). The nesting can be of any depth.
- You can define whether referenced libraries should be visible in the Library Manager.
- You can integrate referenced libraries via library placeholders. This way you avoid the problems that could occur due to version dependencies or the necessity to use vendor-specific libraries.

See also

- *Chapter 1.4.1.2.3.1 “Retrieving and Editing Project Information” on page 191*

1.4.1.16.2 Adding a Library to the Application

The following instructions describe how to integrate for example the library `Util` into your application in order to use its library POUs.

1. Select the Library Manager and click *“Project ➔ Edit Object”* to open it in the editor.
   ⇨ The Library Manager opens in the editor.
2. Click *“Library ➔ Add Library”*.
   ⇨ The *“Add Library”* dialog opens.
3. Type the string "util" into the input field above to search the library.
   ⇨ The library `Util` is displayed in the library view.
4. Select the library `Util` and click *“OK”* to close the dialog.
   ⇨ The library `Util` is added to the Library Manager.

See also

- *Chapter 1.4.1.8.7 “Using Library POUs” on page 265*
- *Chapter 1.4.1.20.3.14.1 “Command 'Add Library'” on page 1116*
- *Chapter 1.4.1.16.3 “Adding a library to the repository” on page 451*

1.4.1.16.3 Adding a library to the repository

The following instructions describe how to install a library in the library repository.

1. Select the command *“Tools ➔ Library Repository”*.
   ⇨ The dialog box *“Library Repository”* opens.
2. Click on the *“Install”* button.
3. Select the library that you wish to install. You can set a file filter here.
   Click on *“Open”*.
   ⇨ The library is added to the repository. The library can now be added in the Library Manager.

See also

- *Chapter 1.4.1.16.2 “Adding a Library to the Application” on page 450*

1.4.1.16.4 Exporting library files

You can export a library from the library manager of a project or from the library repository and then save it as a file to the hard disk.

Export from the library manager

1. Open a library manager of an application in a project.
2. Select a library in the library manager.
3. Click the export command in the context menu.
   ⇨ The *“Export Library”* dialog box opens.
4. If the selected library is linked in the project not only as a compiled library, but also in source format, then both file types are in the drop-down list for *“File type”*. Otherwise, the filter automatically shows the available type: "`*.library` or `*.compiled-library`.
5. Select the file type and storage location and click *“Save”*.

Export from the library manager

1. Open the CODESYS library repository (*“Tools”* menu).
2. Select a library version in the window of the installed libraries.

3. Click the *“Export”* button.
   ⇨ The *“Export Library”* dialog box opens.
4. As step 4 and 5 for "Export from the library manager".

See also

- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*
- *Chapter 1.4.1.20.3.8.5 “Command 'Library Repository'” on page 1061*

1.4.1.17 Managing devices

CODESYS manages the installed devices in the device repository. A device repository is a defined location in the file system. In the default CODESYS installation, it is defined with an absolute path as the system repository. You install or uninstall devices in the *“Device Repository”* dialog. The system installs a device by reading the device description file. The properties of a device are defined in these files regarding configurability, programmability, and possible connections to other devices.

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

See also

- *Chapter 1.4.1.20.3.8.8 “Command 'Device Repository'” on page 1067*
- *Chapter 1.4.1.17.1 “Installing devices” on page 452*

1.4.1.17.1 Installing devices

Install a device in the device repository in order to include it in your project.

1. Click *“Tools ➔ Device Repository”*.
   ⇨ The *“Device Repository”* dialog box opens.
2. Select the install location. *“System Repository”* is set by default.
3. Click *“Install”*.
   ⇨ The *“Install Device Description”* dialog box opens.
4. Select the file path of the device description.
5. Select the file type filter of the required device description.
   ⇨ All device descriptions of the selected file type are listed.
6. Select the required device description and click *“Open”*.
   ⇨ CODESYS adds the device description to the matching category of your device repository.

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

See also

- *Chapter 1.4.1.20.3.8.8 “Command 'Device Repository'” on page 1067*

1.4.1.18 Security

Due to the increased networking of controllers and plants, potential threats are also quickly rising. Therefore, you should carefully consider all possible security measures.

Security measures are absolutely necessary to protect data and communication channels from unauthorized access.

On the following help pages, you can learn more about the safety functions of CODESYS and the controller.

1.4.1.18.1 General Information

The following provides some general information about safety functions (security measures). This information applies regardless of the usage in CODESYS or one with a connection controller.

Access protection with user management

As a means of protecting against unauthorized access to data, it is necessary to configure user accounts with specific access rights. Only a user with the credentials has access to the data or functions.

Creating passwords according to the general recommendations for achieving a high password strength is a tremendous contribution to security.

The following types of user management are roughly distinguished as follows:

- Simple user management:
  To access data, only a password or the valid combination of user name and password has to be entered. This means that access can be only granted or denied. Graduated permissions cannot be configured.
- Group-based user management:
  The access rights are assigned to user groups. Users who belong to a group can access the data or functions after entering the credentials with precisely these assigned and different permissions.

Encryption, signature

Encryption:

Encryption of data means the following: Data is converted into an unreadable form and can only be made readable again with a matching key. In the simplest case, the key is a password or a key pair.

There are two types of encryption methods:

- Symmetric method: (the only type of encryption until the mid-1970s)
  Characteristic: Use of a secret key
  Advantages: Fast, simple encoding
  Disadvantages: The key has to be shared secretly.
- Asymmetric method:
  Characteristic: Use of a key pair (private/secret key and public key)
  Advantages: The public key can be made accessible to anyone, and authentication possible with it.
  Disadvantages: Slow (approx. 1,000 to 10,000 times slower than symmetric methods); complex encoding; long key lengths

Key exchange is usually performed by asymmetric methods; encryption and decryption by symmetric methods.

Signature:

In order for the irrefutable ownership and integrity of a message to be verifiable, it should be provided with a signature. These are usually the steps involved:

- Sender: Determines a unique hash value over the data (H)
- Sender: Encrypts the hash value with private key (He)
- Recipient: Also calculates the hash value and decrypts the He with the public key and compares the two values. This allows the sender to be identified uniquely and verifies that the sender owns the private key.

In the case of asymmetric encryption, a public key contained in a certificate is first exchanged between the sender and the recipient. In addition, each participant needs a private key with which they can decrypt the data if they have the certificate. So if you want to access a certificate, you need a certificate AND a private key.

Hash methods are necessary for this:

- Hash method:
  Characteristic: Unique thumbprint of the data (for example, checksum of the data)
  As low a collision as possible (it is very difficult to find / construct two different data for a single hash value)

Certificates

In order to assign the public key to an identity, it is usually embedded in a certificate.

In certificate-based systems, each user receives a digital certificate. The certificate is used for digital identification. It contains information about the identity and the public key of the user. Each certificate is authenticated by an issuing authority, which in turn may be authenticated by higher authorities. The trust system of this PKI (Public Key Infrastructure) is strictly hierarchical. The common trust anchor is a root certificate.

Contents of a certificate:

- Version
- Serial number
- Algorithm ID
- Issuer (authority or company)
- Validity from (not before) to (not after)
- Certificate owner (subject)
- Certificate owner key information (subject public key)
  - Public key algorithm
  - Public key of the certificate owner
- Unique ID of the issuer (optional)
- Unique ID of the owner (optional) The owner possess a private key matching the public key.
- Extensions
  - Purpose (extended key usage)
  - ...

The certificate consists of 2 parts/files:

- Public X.509 certificate (can be issued to anyone)
- Private key that matches the certificate or its public key only (has to be kept secret).

To manage the certificates in your local "Windows Certificate Store", see the following help page:

- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*

1.4.1.18.2 Security for the development system

In CODESYS, you can apply access protection to projects, libraries, as well as individual applications. In addition to a simple write protection for a project, a user management (credentials, access rights) and encryption using certificates should be used.

See current help:

- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*
- *Chapter 1.4.1.8.17 “Encrypting an application” on page 294*

1.4.1.18.3 Security for the Runtime/PLC

Communication with the controller connected in the CODESYS project should be protected against unauthorized access in the following ways:

- Enabling user management: simple or group-based
- Certificate-based encryption of communication with the controller

Enabling the security features

First switch the communication to encryption so that you do not reveal any credentials to other participants in the network when transferring the user management.

Enforcing encrypted communication

- On the controller:
  - Runtime version >= 3.5 SP14: Encryption can be enabled for *“Communication Policy”* and enforced for all clients.
- In CODESYS:
  - Encrypted communication can be selected as an option in the device editor on the *“Communication Settings”* tab (command or *“Change Communication Policy”* dialog) or in the *“Security Screen”* view.

    See the current help regarding this:

    Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381

    If the CODESYS Security Agent is installed, then see the help for CODESYS Security Agent.

Enforcing a user management

- On the controller:
  - Runtime version >= 3.5 SP17: User management is enforced by default for *“Communication Policy”*.

    Note: For enabling the user management, at least a CODESYS development system V3.5 SP16 is necessary. This means that, in the case of enforced user management which has not been enabled yet, you cannot connect to an older development system.
- In CODESYS:
  - See the current help regarding this:

    Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381

    Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385

1.4.1.18.4 Security for CODESYS WebVisu

Protect the connection between the web server of the controller and the visualization client with the following measures against unwanted access:

- Configure an HTTPS connection (encryption with SSL/TSL) between the visualization client and the web server.
- Restrict access to the visualization and configure a visualization user management.

Configure an encrypted connection.

An HTTPS connection between the web server and the visualization client requires authentication of the web server by means of a certificate. You can create a self-signed certificate in the *“Security Screen”*.

1. Click *“View ➔ Security Screen”*.
2. Create a certificate for the web server on your controller.
   ⇨ The certificate data for the web server is displayed.
3. **Stop your controller.**
4. Restart the controller.
   ⇨ The new certificate is active.
5. Download your application to the controller.
6. Open your browser and specify the URL address of your web server.
   The URL of a secure connection corresponds to the following format:
   `https:// <IP address/URL> :443/ <name of HTM file> .htm`.
   The HTML file name has to match the configured name as it is set in the *“Visualization Manager”* object below the WebVisu variant. You will find the IP address of the controller in the device editor when the a connection is active.
   ⇨ Example: `https://localhost:443/webvisu.htm`
   The browser establishes a connection. If the certificate is not rated as trusted, then a security notice appears.
7. Confirm that you know the risk and want to proceed.
   ⇨ You have created self-signed certificate and confirmed it as trusted.
   Now start the web application with the visualization. The lock symbol in the browser indicates secure communication.

See the chapter "Run as CODESYS WebVisu", which describes in detail how you use certificates in the security screen.

See also

- *Chapter "Run as CODESYS WebVisu"*
- *Chapter "User management of the visualization"*
- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

1.4.1.18.5 FAQ

Certificate expired

If the certificate from the controller for encrypted communication has expired (valid from "not before" until "not after"), you get a prompt with a corresponding message in CODESYS when you attempt to access the controller. For example, to renew the expired certificate, you can accept the expired certificate and connect to the controller.

You will see this message again every time you try to login until a valid certificate is installed on the controller.

If you have created or imported a new certificate on the controller, then this new certificate will be available for you to accept the next time you login.

See also

- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*

Other clients that communicate encrypted with the controller (for example, PLCHandler) will typically not accept an expired certificate. This means that no connection can be established here.

New certificate (while the current one is still valid)

A new certificate can be issued before the existing certificate expires. This makes it possible for the encrypted communication to continue seamlessly. As soon as a new certificate is available on the controller parallel to the one currently used, the new certificate will be offered by the controller at the next login attempt. All you have to do is accept it.

See also

- *“Installing a controller certificate for encrypted communication via the PLC shell of the device editor” on page 383*

Client does not support security feature

User management

The following CODESYS clients do not support user management yet:

- WebServer < V3.5.14.0

In order for these clients to be able to establish a connection to the controller, the user management must not be enabled.

Encrypted communication

The following CODESYS clients do not support encrypted communication yet:

- Data servers with compiler version =< V3.5.9.0
- WebVisu < V3.5.14.0 or in the case of enabled file transfer
- WebServer < V3.5.14.0
- Remote TargetVisu
- Data source ApplicationV3
- OPC Server V3
- PLCHandler < V3.5.14.0

In order for these clients to be able to establish a connection to the controller, the encrypted communication can be set as optional. Therefore clients can establish either an encrypted or an unencrypted connection.

Do not use the same user or password for encrypted and unencrypted communication.

See also

- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*

CA-signed certificates preferred (PLC shell)

Using CA-signed certificates is not conveniently supported yet in CODESYS. However, you can still request and use these types of certificates. In the device editor, on the *“PLC Shell”* tab, you export the required CSR files to the file system and import from there the CER files sent from the certification authority.

Requesting and providing a CA-signed certificate

☑ You are connected to the controller.

1. First you generate certificate signing requests (CSR) of all server certificates.

   For this purpose, click the *“PLC Shell”* tab of the controller and type the command `cert-createcsr` in the input line.

2. Click the *“Log”* tab and then the refresh button (🔄).

   ⇨ In the log entries, you can see that the CSR files were generated.

3. Click the *“Files”* tab and open the file path `cert/export` in the right side of the *“Runtime”* dialog.

   ⇨ The `export` folder contains the generated CSR files, for example `0_CmpsecureChannl.csr, 1_CmpApp.csr, 2_CmpWebServer.csr`.

4. Select a file path where you wish to insert the CSR files in the left side of the *“Host”* dialog, mark the CSR files in the right side of the dialog, and click <<.

   ⇨ The CSR files are copied to the required folder.

5. These requests can be signed for certification signing by a certificate authority (CA), and then you receive a signed certificate from the certification authority.

6. In the steps that follow, you import these signed server certificates to your controller.

   NOTICE!
   Caution: Self-signed certificates of the server must be deleted before importing the CA-signed certificates.

7. Select the *“Path”* `cert/import` in the right side of the *“Runtime”* dialog.

8. In the left side of the *“Host”* dialog, select the path in the file system where you saved the signed certificates and selected the certificates.

9. Click >>.

   ⇨ The certificates are copied to the `cert/import` folder.

10. Click the *“PLC Shell”* tab.

11. Type the command `cert-import own <file name of the certificate.cer>` in the input line of the tab and press the *[Enter]* key.

    ⇨ The signed certificates are available to the runtime system servers.

See also

- *Chapter 1.4.1.20.2.8.10 “Tab 'PLC Shell'” on page 852*
- *Chapter 1.4.1.20.2.8.8 “Tab 'Log'” on page 848*
- *Chapter 1.4.1.20.2.8.7 “Tab 'Files'” on page 848*

Problems at login

If you have entered an incorrect password when logging in to the user management of the controller, then the login dialog reappears immediately afterwards. After three incorrect attempts, the controller is locked for a defined period of time. However, stricter policies on the controller can lead to the user being locked out and only authorized again by an administrator.

See also

- *Chapter 1.4.1.20.3.6.2 “Command 'Login'” on page 1028*

Disabling User Management

NOTICE!
After disabling the user management, your controller is accessible again for everyone in the network of the controller. Therefore, you should only do this in justified exceptional cases or if the clients used do not support any user management.

For enabling the user management, at least a CODESYS development system V3.5 SP16 is necessary. This means that, in the case of enforced user management which has not been enabled yet, you cannot connect to an older development system.

1. If the security policy for device user management is set to "Enforced", first set it back to "Optional".
2. Execute the *“Reset Origin Device”* command. This deletes the user management and you can then reconnect to the controller without having to enter user credentials. Note: In CODESYS Version 3.5 SP16 Patch 20 and higher, you can exclude the boot application from the delete operation when you execute *“Reset Origin Device”*.

See also

- *“Changing the communication policy (encryption, user management)” on page 384*
  Chapter 1.4.1.20.3.6.13 “Command 'Reset Origin Device'” on page 1040
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*
- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*

Permitting encrypted communication again

Remember that not every controller supports the deactivation of encrypted communication.

NOTICE!
We strongly advise against disabling encrypted communication. Especially in connection with an enabled user management, encrypted communication should be enabled so that credentials do not fall into the wrong hands.

To disable encrypted communication with the controller again, proceed as follows:

1. If the communication policy for encrypted communication is set to "Enforced", first set it back to "Optional".
2. In the device editor, on the *"Communication Settings"* tab in the *"Device"* menu, disable *"Encrypted communication"*. If you have installed the CODESYS Security Agent, then you can also change the setting in the *"Security Screen"*.
   ⇨ CODESYS establishes unencrypted communication again with the controller. Other clients can also communicate again without encryption.

See also

- *Chapter 1.4.1.10.2 "Encrypting Communication, Changing Security Settings" on page 381*
- *Chapter 1.4.1.20.3.3.18 "Command 'Security Screen'" on page 995*

1.4.1.19 Reference, Programming

1.4.1.19.1 Programming Languages and Editors

You program a POU in each case in the editor for the implementation language that you selected when creating the POU. CODESYS offers a text editor for ST and graphic editors for SFC, FBD/LD/IL and CFC.

The editor opens with a double-click on the POU in the device tree or in the *“POUs”* view.

Each of the programming language editors consists of two sub-windows:

- In the upper part you make declarations in the “declaration editor”, in text or tabular form depending on the setting.
- In the lower part you insert the implementation code in the respective language.

You can configure the display and the behavior of each editor project-wide on the associated tab of the CODESYS options.

Declaration Editor

In the declaration editor, you declare variables in variable lists and POUs.

If the declaration editor is used with an implementation language editor, then is opens in a view above the implementation language editor.

The declaration editor offers two possible views: textual and tabular. In the *“Tools ➔ Options ➔ Declaration Editor”* dialog, you define whether only the textual view or only the tabular view is available, or whether you can switch between both views by means of the buttons on the right side of the editor view.

A rectangle selection is possible in the textual view of the declaration editor. The key combinations for the rectangle selection are located on the help page for the ST editor.

See also

- *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226*
- *Chapter 1.4.1.8.2 “Declaration of Variables ” on page 222*
- *Chapter 1.4.1.19.1 “Programming Languages and Editors” on page 460*
- *Chapter 1.4.1.20.4.13.4 “Dialog 'Options' – 'Declaration Editor'” on page 1190*
- *Chapter 1.4.1.19.1.3.1 “ST Editor” on page 463*

Declaration editor in online mode

In online mode, you see the tabular view of the editor. The header always contains the current object path:`<device name>.<application name>.<object name>`. In contrast to offline mode, the table also contains the *“Value”* and *“Prepared Value”* columns.

The *“Value”* column shows the actual value on the PLC, offering monitoring functionality. If the expression is an array with more than 1,000 elements, then you can define the range of the array indices to monitor. To do this, double-click in the *“Data Type”* column to open the *“Monitoring Area”* dialog. In this dialog, the declared array range is specified as the *“Valid area”* for monitoring. A maximum of 20,000 elements can be monitored per array. You define the range of the array indices to be monitored by specifying the *“Start”* and *“End”* indices. In order to move this area more easily while maintaining the same size, the available scrollbars can be used coupled. Tot toggle between coupled and not coupled, click the symbol on the right of the bar. In non-coupled state, you can increase or decrease the size of the area to be monitored as desired.

The *“Prepared Value”* column contains the value that you prepared for forcing or writing.

If you double-click a *“Prepared value”* field, then you can specify a value explicitly for writing or forcing. In the case of enumerations, a combo box opens from which you can select a value. In the case of a Boolean variable you can toggle the prepared value with the help of the *[Enter]* key or the *[Space]* bar. If an expression (variable) is of a structured data type, for example the instance of a function block or an array variable, then a plus or a minus sign is placed in front.

You can customize the format of the representation of floating-point values in the options for monitoring.

See also

- *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226*
- *Chapter 1.4.1.20.4.7 “Dialog Box 'Prepare Value'” on page 1153*
- *“Forcing in the declaration part” on page 402*
- *Chapter 1.4.1.20.4.13.18 “Dialog 'Options' - 'Monitoring'” on page 1197*

Common functions in graphical editors

The implementation part of the graphical editors for FBD, LD, CFC, and SFC contains a toolbar in the lower right corner.

	Return to normal editing mode: The mouse pointer changes back to the shape of the default arrow. You can select and edit elements in the editor view.
	Panning tool: The mouse pointer changes to the shape of two crossed arrows. You can click and drag anywhere in the editor view to move the visible area of the FBD/LD/IL editor or also pivot a CFC chart.
	Magnification tool: A magnified window opens in the lower right corner of the editor view and the mouse pointer changes to the shape of a cross. As you move the mouse pointer over your diagram, the magnification tool shows the area of the diagram under the cross at 100% magnification. Note: If you click in the view, then the magnification tool closes and the part of the diagram that the tool contained is displayed at 100% magnification. If you want to retain the set zoom factor, then you should use the default arrow () for returning to the default editing mode.
	Zooming tool: This opens a drop-down list with a selection of zoom factors. Clicking more selections (. . .) will open the *“Zoom”* dialog for typing other values. The current zoom factor is always shown to the left of the symbol.

Zooming with the scroll wheel: By holding down the *[Ctrl]* key and moving the scroll wheel, you can change the zoom factor in steps of 10%.

Every graphical editor has its own *“ToolBox”* view that is located on the right of the editor view by default. The toolbox contains elements that you can drag to insertion points in the editor view. CODESYS highlights the insertion points with gray position flags in the shape of diamonds, triangles, or arrows. These flags are green when you move the mouse pointer over them. When you release the mouse button, CODESYS inserts the element at the selected position.

It is also possible to use the mouse for moving elements in the editor.

You can drag function block declarations in the FBD, LD, and CFC graphical editors to the editor view. To do this, select the full declaration (variable name and data type) and drag it to a suitable position in the editor view. In the ladder diagram, you can also drag Boolean declarations to the editor and insert them as contacts.

See also

- *Chapter 1.4.1.19.1.4.1 “SFC editor” on page 476*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.6.1 “CFC Editor” on page 511*

Structured Text and Extended Structured Text (ExST)

ST Editor

The ST editor is a textual editor used for the implementation of code in Structured Text (ST) and Extended Structured Text (ExST).

The line numbering is displayed on the left side of the editor. When inputing programming elements, the "List components" functionality (activated in the CODESYS options, *“SmartCoding”* category) and the Input Assistant *[F2]* are also useful. When the cursor is placed over a variable, CODESYS shows a tooltip with information for declaring variables.

The box selection can be made with the following key combinations:

- *[Shift]+[Alt]+[Arrow Right]*: The selected area is extended one position to the right.
- *[Shift]+[Alt]+[Arrow Left]*: The selected area is extended one position to the left.
- *[Shift]+[Alt]+[Arrow Up]*: The selected area is extended one position up.
- *[Shift]+[Alt]+[Arrow Down]*: The selected area is extended one position down.

The behavior (for example parentheses, mouse actions, tabs) and appearance of the editor are configured in the CODESYS options in the *“Text Editor”* category.

For an incremental search for strings in the editor, open an input field at the bottom edge of the editor by means of the key combination *[Ctrl]+[Shift]+[i]*. As soon as you start typing in characters, the corresponding search locations are highlighted in color in the editor. The number of found matches is shown to the right of the input field. You can set the cursor at the search location by using the arrow keys or the key combinations *[Alt]+[Page Up]* or *[Alt]+[Page Down]*.

When you place the cursor on a symbol name, all occurrence locations of the symbol within the editor are highlighted in color. The search locations correspond to the hits in the cross-reference list. For very large projects, this can cause input delays. In this case, you can disable the function in the options of the text editor.

CODESYS identifies syntax errors already when inputing in the editor and shows the corresponding messages in the message view (*“Precompile”* category). If the corresponding option is selected in the CODESYS options (*“SmartCoding”* category), then the error locations in the text are also underlined with a wavy red line.

The *“Format Document”* command provides an automatic formatting of syntactically correct ST code.

See also

- *Chapter 1.4.1.8.3.3.1 “Programming structured text (ST)” on page 254*
- *Chapter 1.4.1.19.1.3.3 “ST expressions” on page 464*
- *Chapter 1.4.1.19.1.3.5.11 “ST – Comments” on page 475*
- *Chapter 1.4.1.19.1.3 “Structured Text and Extended Structured Text (ExST)” on page 463*
- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*
- *Chapter 1.4.1.20.3.3.5 “Command 'Messages'” on page 986*
- *Chapter 1.4.1.20.3.2.45 “Command 'Advanced' - 'Format Document'” on page 984*

ST editor in online mode

In online mode CODESYS displays the variables and expressions in the ST editor. The writing and forcing of the variables and expressions as well as debugging functions (breakpoints, single step execution) are also possible.

If you use assignments as expressions in ST programming, no further breakpoint positions are created within a line.

See also

- *Chapter 1.4.1.12.1.1 “Calling of monitoring in programming objects ” on page 410*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*
- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*
- *Chapter 1.4.1.11 “Testing and Debugging” on page 394*
- *Chapter 1.4.1.11 “Testing and Debugging” on page 394*
- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*
- *Chapter 1.4.1.11.6 “Flow Control” on page 406*
- *Chapter 1.4.1.11.7 “Determining the current processing position with the call stack” on page 408*

ST expressions

An expression is a construct that returns a value following its evaluation.

Expressions are composed of operators and operands. In Extended Structured Text (ExST) you can also use assignments as expressions. An operand can be a constant, a variable, a function call or a further expression.

Examples

`2014`	`(* Constant *)`
`ivar`	`(* Variable *)`
`fct(a,b)`	`(* Function call *)`
`(x*y)/z`	`(* Expression *)`
`real_var2 := int.var;`	`(* in ExST: Assignment *) *)`

See also

- *“ExST - Extended structured text” on page 254*

Evaluation of expressions

The evaluation of an expression takes place by processing the operators according to certain rules of binding. CODESYS processes the operator with the strongest binding first. Operators with the same binding strength are processed from left to right.

Operation	Symbol	Binding strength
Parenthesize	(Expression)	Strongest binding
Function Call	Function name (parameter list) all operators with syntax: <operator> ()	
Exponentiate	`EXPT`	
Negate Complementation	`-` `NOT`	

Multiplication
Division
Modulo
	`*`
	`/`
	`MOD`	

Addition Subtraction	`+` `-`	
Comparison	<,>,<=,>=	

Operation	Symbol	Binding strength
Equality Inequality	= <>	
Bool AND	AND AND_THEN	
Bool XOR	XOR	
Bool OR	OR OR_ELSE	Weakest binding

See also

- *Chapter 1.4.1.19.3 “Operators” on page 542*

Assignments

ST assignment operator

Syntax:

<operand> := <expression>

This assignment operator executes the same function as the MOVE operator.

See also

- *Chapter 1.4.1.19.3.6 “Operator 'MOVE'” on page 550*

ST assignment operator for outputs

The assignment operator => assigns the output of a function, a function block, or a method to a variable. The position on the right side of the operator can also be blank.

Syntax

<output> => <variable>

Example

```
FBcomp_Output1 => bVar1;
FBcomp_Output2 => ;
```

FBcom_Output1 and FB_Output2 are outputs of a function block. The value of FBcom_Output1 is assigned to the variable bVar1.

ExST assignment 'S='

When the operand of the Set assignment switches to TRUE, then TRUE is assigned to the variable to the left of the operator. The variable is set.

```
<variable name> S= <operand name> ;
```

The variables and the operand have the data type `BOOL`.

Example

```
PROGRAM PLC_PRG
VAR
    xOperand: BOOL := FALSE;
    xSetVariable: BOOL := FALSE;
END_VAR

xSetVariable S= xOperand;
```

When the operand `xOperand` switches from `FALSE` to `TRUE`, then `TRUE` is also assigned to the variable `xSetVariable`. But then the variable keeps this state, even if the operand continues to change its state.

Multiple assignments

NOTICE!
In the case of multiple assignments within a code line, the individual assignments are not processed from right to left, but all assignments refer to the operands at the end of the code line.

Example

```
FUNCTION funCompute : BOOL
VAR_INPUT
    xIn : BOOL;
END_VAR
IF xIn = TRUE THEN
    funCompute := TRUE;
    RETURN;
END_IF

PROGRAM PLC_PRG
VAR
    xSetVariable: BOOL;
    xResetVariable: BOOL := TRUE;
    xVar: BOOL;
END_VAR
xSetVariable S= xResetVariable R= funCompute(xIn := xVar);
```

`xResetVariable` gets the `R=` assignment of the return value of `funCompute`.
`xSetVariable` gets the `S=` assignment of ht return value of `funCompute`, but not from `xResetVariable`.

See also

- *“ExST - Extended structured text” on page 254*
- *Chapter 1.4.1.19.1.3.4.4 “ExST assignment 'R='” on page 466*

ExST assignment 'R='

When the operand of the Reset assignment switches to `TRUE`, then `FALSE` is assigned to the variable to the left of the operator. The variable is reset.

```
<variable name> R= <operand name> ;
```

The variables and the operand have the data type `BOOL`.

Example

```
VAR
    xOperand: BOOL := FALSE;
    xResetVariable: BOOL := TRUE;
END_VAR

xResetVariable R= xOperand;
```

When the operand `xOperand` switches from `FALSE` to `TRUE`, then `FALSE` is also assigned to the variable `xResetVariable`. But then the variable keeps its state, even if the operand continues to change its state.

Multiple assignments

NOTICE!
In the case of multiple assignments within a code line, the individual assignments are not processed from right to left, but all assignments refer to the operands at the end of the code line.

Example

```
FUNCTION funCompute : BOOL
VAR_INPUT
    xIn : BOOL;
END_VAR
IF xIn = TRUE THEN
    funCompute := TRUE;
    RETURN;
END_IF

PROGRAM PLC_PRG
VAR
    xSetVariable: BOOL;
    xResetVariable: BOOL := TRUE;
    xVar: BOOL;
END_VAR
xSetVariable S= xResetVariable R= funCompute(xIn := xVar);
```

`xResetVariable` gets the `R=` assignment of the return value of `funCompute`.
`xSetVariable` gets the `S=` assignment of ht return value of `funCompute`, but not from `xResetVariable`.

See also

- *“ExST - Extended structured text” on page 254*
- *Chapter 1.4.1.19.1.3.4.3 “ExST assignment 'S='” on page 465*

ExST – Assignment as expression

In ExST, as an extension to the IEC 61131-3 standard, CODESYS permits the use of assignments as expressions.

Examples

`int_var1 := int_var2 := int_var3 + 9;`	(* `int_var1` and `int_var2` receive the value of `int_var3 + 9` *)
`real_var1 := real_var2 := int_var;`	(* `real_var1` and `real_var2` receive the value of `int_var` *)
`int_var := real_var1 := int_var;`	(* incorrect assignment, the data types do not correspond! *)

`IF b := (i = 1) THEN`
`i := i + 1;`
`END_IF`	

See also

- *“ExST - Extended structured text” on page 254*

Assignment Operator 'REF='

The operator generates a reference (pointer) to a value.

Syntax:

```
<variable name> REF= <variable name> ;
```

Example

```
refA : REFERENCE TO DUT;
B : DUT;
C : DUT;

A REF= B; // corresponds to A := ADR(B);
A := C; // corresponds to A^ := C;
```

See also

- *Chapter 1.4.1.19.5.13 “Reference” on page 658*
- *Chapter 1.4.1.20.3.12.8 “Command 'REF= (Reference Assignment)'” on page 1091*

Statements

ST statement 'IF'

The IF statement is used for checking a condition and, depending on this condition, for executing the subsequent statements.

A condition is coded as an expression that returns a Boolean value. If the expression returns TRUE, then the condition is fulfilled and the corresponding statements after THEN are executed. If the expression returns FALSE, then the following conditions, which are identified with ELSIF, are evaluated. If an ELSIF condition returns TRUE, then the statements are executed after the corresponding THEN. If all conditions return FALSE, then the statements after ELSE are executed.

Therefore, at most one branch of the IF statement is executed. ELSIF branches and the ELSE branch are optional.

Syntax

```
IF <condition> THEN
    <statements>
( ELSIF <condition> THEN
        <statements> )*
( ELSE
        <statements> )?
END_IF;
// ( ... )* None, once or several times
// ( ... )? Optional
```

Example

```
PROGRAM PLC_PRG
VAR
    iTemp: INT;
    xHeatingOn: BOOL;
    xOpenWindow: BOOL;
END_VAR

IF iTemp < 17 THEN
    xHeatingOn := TRUE;
ELSIF iTemp > 25 THEN
    xOpenWindow := TRUE;
ELSE xHeatingOn := FALSE;
END_IF;
```

The program is run as follows at runtime:

For the evaluation of the expression `iTemp < 17 = TRUE`, the subsequent statement is executed and the heating is switched on. For the evaluation of the expression `iTemp < 17 = FALSE`, the subsequent ELSIF condition `iTemp > 25` is evaluated. If this is true, then the statements in ELSIF are executed and the view is opened. If all conditions are FALSE, then the statement in ELSE is executed and the heating is switched off.

See also

- *Chapter 1.4.1.19.1.3.3 “ST expressions” on page 464*

ST instruction 'FOR'

The FOR loop is used to execute instructions with a certain number of repetitions.

Syntax:

```
FOR <counter> := <start value> TO <end value> {BY <increment> } DO
<instructions>
END_FOR;
```

The section inside the curly parentheses {} is optional.

CODESYS executes the `<instructions>` as long as the `<counter>` is not greater, or - in case of negative increment - is not smaller than the `<end value>`. This is checked before the execution of the `<instructions>`.

Every time the instructions `<instructions>` have been executed, the counter `<counter>` is automatically increased by the increment `<increment>`. The increment `<increment>` can have any integral value. If you do not specify an increment, the standard increment is `1`.

Example

```
FOR iCounter := 1 TO 5 BY 1 DO
iVar1 := iVar1*2;
END_FOR;
Erg := iVar1;
```

If you have pre-configured `iVar1` with `1`, `iVar1` has the value `32` after the FOR loop.

CAUTION!

The end value <end value> may not attain the same value as the upper limit of the data type of the counter.

If the end value of the counter is equal to the upper limit of the data type of the counter, an endless loop results. For example, an endless loop results in the above example if `iCounter` is of the data type `SINT` and the `<end value>` equals `127`, since the data type `SINT` has the upper limit `127`.

As an extension to the IEC 61131-3 standard you can use the `CONTINUE` instruction within the `FOR` loop.

See also

- *Chapter 1.4.1.19.5.2 “Integer data types” on page 647*
- *Chapter 1.4.1.19.1.3.5.9 “EXST Statement 'CONTINUE'” on page 474*

ST instruction 'CASE'

Use this dialog box for pooling several conditional instructions containing the same condition variable into a construct.

Syntax:

```
CASE <Var1> OF
<value1>:<instruction1>
<value2>:<instruction2>
<value3, value4, value5>:<instruction3>
<value6 ... value10>:<instruction4>
...
<value n>:<instruction n>

{ELSE <ELSE-instruction>}

END_CASE;
```

The section within the curly brackets `{}` is optional.

Processing scheme of a CASE instruction.

- If the value of the variable <Var1> is <value i>, then the instruction <instruction i> is executed.
- If the variable <Var1> has non of the given values, then the <ELSE-instruction> is executed.
- If the same instruction is executed for several values of the variable, then you can write the values in sequence, seperated by commas.

Example

```
CASE iVar OF
1, 5: bVar1 := TRUE;
      bVar3 := FALSE;

2: bVar2 := FALSE;
   bVar3 := TRUE;

10..20: bVar1 := TRUE;
        bVar3= TRUE;
ELSE
   bVar1 := NOT bVar1;
   bVar2 := bVar1 OR bVar2;
END_CASE;
```

ST instruction 'WHILE'

The WHILE loop is used like the FOR loop in order to execute instructions several times until the abort condition occurs. The abort condition of a WHILE loop is a boolean expression.

Syntax:

```
WHILE <boolean expression> DO
 <instructions>
 END_WHILE;
```

CODESYS repeatedly executes the <instructions> for as long as the <boolean expression> returns TRUE. If the boolean expression is already FALSE at the first evaluation, then CODESYS never executes the instructions. If the boolean expression never adopts the value FALSE, then the instructions are repeated endlessly, as a result of which a runtime error results.

Example

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

NOTICE!
You must ensure by programming means that no endless loops are caused.

In a certain sense the WHILE and REPEAT loops are more powerful than the FOR loop, since you don't need to already know the number of executions of the loop before its execution. In some cases it is thus only possible to work with these two kinds of loop. If the number of executions of the loop is clear, however, then a FOR loop is preferable in order to avoid endless loops.

As an extension to the IEC 61131-3 standard you can use the CONTINUE instruction within the WHILE loop.

See also

- *Chapter 1.4.1.19.1.3.5.2 “ST instruction 'FOR'” on page 469*
- *Chapter 1.4.1.19.1.3.5.9 “EXST Statement 'CONTINUE'” on page 474*

ST Statement 'REPEAT'

The `REPEAT` loop is used like the `WHILE` loop, but with the difference that CODESYS only checks the abort condition after the execution of the loop. The consequence of this behavior is that the `REPEAT` loop is executed at least once, regardless of the abort condition.

Syntax:

```
REPEAT
<instructions>
 UNTIL <boolean expression>
 END_REPEAT;
```

CODESYS executes the `<instructions>` until the `<boolean expression>` returns `TRUE`.

If the boolean expression already returns `TRUE` at the first evaluation, CODESYS executes the instructions precisely once. If the boolean expression never adopts the value `TRUE`, then the instructions are repeated endlessly, as a result of which a runtime error results.

Example

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

In a certain sense the `WHILE` and `REPEAT` loops are more powerful than the `FOR` loop, since the number of executions of the loop doesn't already need to be known before its execution. In some cases you can only work with these two kinds of loop. If the number of executions of the loop is clear, however, then a `FOR` loop is preferable in order to avoid endless loops.

As an extension to the IEC 61131-3 standard you can use the CONTINUE instruction within the `WHILE` loop.

See also

- *Chapter 1.4.1.19.1.3.5.4 “ST instruction 'WHILE'” on page 471*
- *Chapter 1.4.1.19.1.3.5.2 “ST instruction 'FOR'” on page 469*
- *Chapter 1.4.1.19.1.3.5.9 “EXST Statement 'CONTINUE'” on page 474*

ST statement 'RETURN'

Use the `RETURN` statement in order to exit from a function block. You can make this dependent on a condition, for example.

Example

```
IF xIsDone = TRUE THEN
    RETURN;
END_IF;

iCounter := iCounter + 1;
```

If the value of `xIsDone` is equal to `TRUE`, then the function block is exited immediately and the statement `iCounter := iCounter + 1;` is not executed.

See also

- *Chapter 1.4.1.19.1.3.5.1 “ST statement 'IF'” on page 469*

ST instruction 'JMP'

The `JMP` instruction is used to execute an unconditional jump to a program line that is marked by a jump label.

Syntax:

```
<label>: <instructions>
JMP <label>;
```

The jump label `<label>` is any unique identifier that you place at the beginning of a program line. On reaching the `JMP` instruction, a return to the program line with the `<label>` takes place.

Example

```
iVar1 := 0;
_label1: iVar1 := iVar1+1;
(*instructions*)

IF (iVar1 < 10) THEN
JMP _label1;
END_IF;
```

NOTICE!
You must ensure by programming means that no endless loops are caused. For example, you can make the jump conditional.

ST instruction 'EXIT'

The `EXIT` instruction is used in a `FOR`, `WHILE` or `REPEAT` loop in order to end the loop regardless of other abort conditions.

See also

- *Chapter 1.4.1.19.1.3.5.2 “ST instruction 'FOR'” on page 469*
- *Chapter 1.4.1.19.1.3.5.4 “ST instruction 'WHILE'” on page 471*
- *Chapter 1.4.1.19.1.3.5.5 “ST Statement 'REPEAT'” on page 472*

EXST Statement 'CONTINUE'

`CONTINUE` is an instruction of the Extended Structured Text (ExST).

The instruction is used inside `FOR`, `WHILE` and `REPEAT` loops in order to jump to the beginning of the next execution of the loop.

Example

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:=INT1/2;
 IF INT1=0 THEN
  CONTINUE; (* to avoid a division by zero *)
 END_IF
Var1:=Var1/INT1; (* executed, if INT1 is not 0 *)
END_FOR;

Erg:=Var1;
```

See also

- *Chapter 1.4.1.19.1.3.5.2 “ST instruction 'FOR'” on page 469*
- *Chapter 1.4.1.19.1.3.5.4 “ST instruction 'WHILE'” on page 471*
- *Chapter 1.4.1.19.1.3.5.5 “ST Statement 'REPEAT'” on page 472*

ST function block call

Syntax

```
<FB-instance>(<FB input variable>:=<value or address>|, <other FB input variables>);
```

Example

```
  TMR:TON;

                    TMR (IN:=%OX5, PT:=T#300ms);
                    varA:=TMR.Q;
```

The timer function block `TON` is instanced in `TMR:TON` and called with assignments for the parameters `IN` and `PT`.

The output `Q` is addressed with `TMR.Q` and assigned to the variable `varA`.

See also

- *Chapter 1.4.1.20.2.18.2 “Object 'Function Block'” on page 883*

ST – Comments

Comment	Description	Example
Single-line	There are two ways of marking: ● Starts with `//` and ends at the end of the line ● Starts with `///` and ends at the end of the line In CODESYS, these comments are handled the same way. However, if library documentation is created using the LibDoc Scripting Collection, the following applies: ● When the property `LibDocContent = DocsOnly` is entered in the project information, only comments marked with `///` are processed into library documentation. See the example for this below the table. ● When `LibDocContent = CommentsAndDocs` (default setting) is defined, all comments are processed into library documentation.	`/// This is a comment.` `/// This is a comment.`
Multiline	Starts with `(*` and ends with `*)`.	`(* This is a multiline comment *)`
Nested	Starts with `(*` and ends with `*)`. Additional comments `(*....*)` can be contained within this comment.	`( * a:=inst.out; (* comment 1 *) b:=b+1; (* comment 2 *) *)`

Comments for tooltips and POU documentation

```
A tooltip in the header of a POU is defined by the following
comment:
    // tooltip text - line 1
    // tooltip text - line 2
    // tooltip text - line 3

    Afterwards the documentation is defined as follows:
    /// reStructuredText
```

Note: It is not recommended to mix the different comment types because this can cause unwanted side-effects when the documentation is generated.

Sequential Function Chart (SFC)

SFC editor

The SFC editor is graphical editor. A new SFC POU includes an Init step and a subsequent transition.

In the SFC editor, you can insert individual elements into the diagram by means of commands in the *“SFC”* menu, the context menu, or the *“ToolBox”* view.

When inserting by means of a menu command, the elements that can be inserted at the currently selected position are available.

Before inserting branches parallel to multiple actions and transitions, you must highlight these actions and transitions in a multiple selection.

You can also drag SFC elements from the *“ToolBox”* view to the diagram. When you drag an element over the editor, CODESYS marks all possible insertion points with gray boxes. If you move the mouse over a gray box, then the color of the box changes to green. When you release the mouse button, the object is inserted at that location.

If you drag a branch into the diagram, then you must set the beginning and the end of the branch using the mouse pointer. You set the beginning of the branch by releasing the mouse button at an insertion point. The color of the box then changes to red. You set the end of the branch by clicking the second insertion point. Then CODESYS inserts a branch around the objects between the beginning and end markers.

For copying step and transition elements that call action objects or transition objects, two different duplication modes can be set. Either the references are copied at the same time, or the referenced objects are embedded and duplicated when copying.

You define the look and feel of the editor in the CODESYS options (*“SFC Editor”*).

See also

- *Chapter 1.4.1.19.1.2 “Common functions in graphical editors” on page 462*
- *Chapter 1.4.1.19.1.4 “Sequential Function Chart (SFC)” on page 476*
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.20.4.13.22 “ Dialog 'Options' - 'SFC Editor'” on page 1200*

SFC Editor in Online Mode

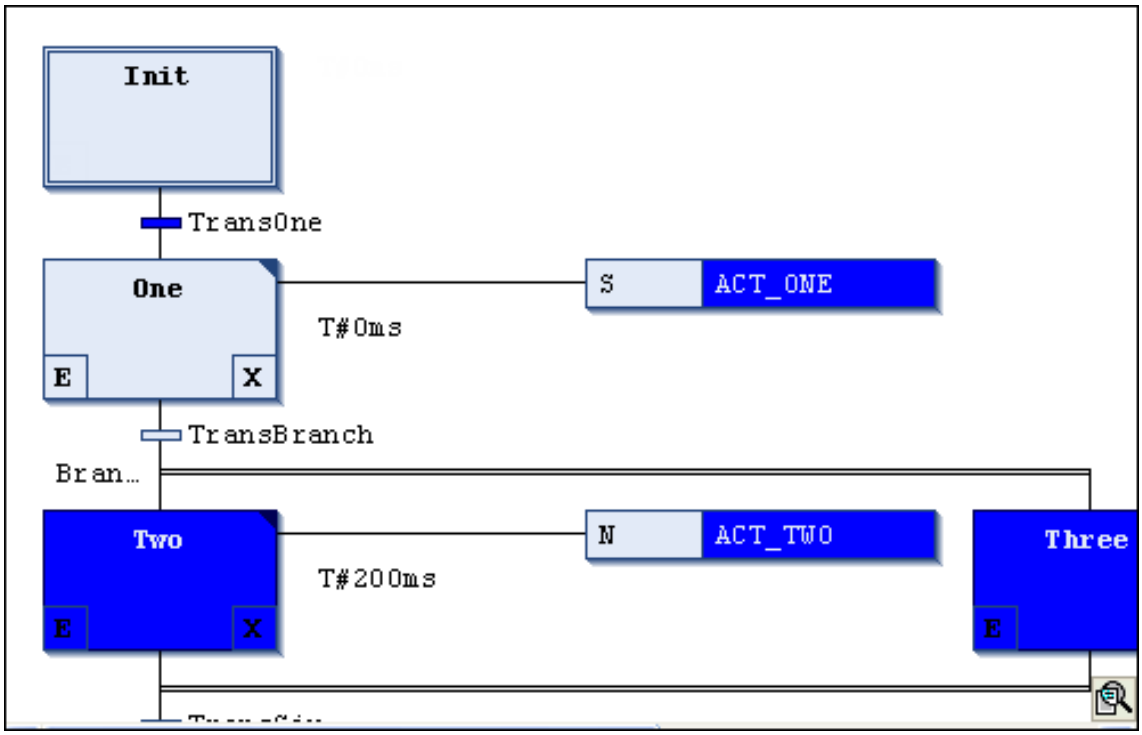
In the SFC editor, the variables and expressions in use on the controller can be displayed at runtime. You can also write and force variables and expressions. Debugging functions, such as breakpoints and step-by-step execution, are not available yet.

In the SFC editor options, you can set the online representation of the SFC elements and attributes.

In the case that you have declared SFC flags explicitly, then they are displayed in the declaration part in online mode. They are not displayed in offline mode.

Note the processing order of elements in an SFC diagram.

In online mode, CODESYS displays active steps in blue.



See also

- *Chapter 1.4.1.19.1.4.5 “Implicit variables” on page 480*
- *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226*
- *Chapter 1.4.1.19.1.4.3 “Processing order in SFC” on page 477*
- *Chapter 1.4.1.20.4.13.22 “ Dialog 'Options' - 'SFC Editor'” on page 1200*

Processing order in SFC

Basic element behavior

- Active step: An active step includes actions currently being executed. In online mode, CODESYS displays active steps in blue.
- Initial step: In the first cycle after calling a POU in SFC, the initial step is activated automatically and the step action is executed.
- CODESYS executes IEC actions at least two times: the first time is when the step is activated, and the second time when the step is deactivated (but not until the next cycle).
- Alternative branches: If the step before the branch is active, then CODESYS passes the first transition of each alternative branch line from left to right. CODESYS activates the subsequent step in the first branch line with a transition yielding `TRUE`.
- Parallel branches: If the step before the branch (horizontal double line) is active and the transition before the branch yields `TRUE`, then CODESYS activates the first steps in every branch line. The branch lines are then processed at the same time. The step after the end of the branch is activated when every last step in each branch line is active and the transition after the double line yields `TRUE`.

Processing order

1. **Reset IEC actions**

   CODESYS resets the internal action control flags of the action qualifiers (N, R, S, L, D, P, SD, DS, SL). These flags control IEC actions. However, flags are not reset when they are called within actions.

2. **Execute exit actions**

   CODESYS verifies whether all steps fulfill the condition for executing the exit action for each step. The order of verification follows the layout in the SFC diagram, from top to bottom and from left to right.

   CODESYS executes an exit action when the step is deactivated (after any entry and step actions have been executed in the preceding cycle and the condition for the subsequent step yields `TRUE`).

3. **Execute entry actions**

   CODESYS verifies whether all steps fulfill the condition for executing the entry action for each step. The order of verification follows the layout in the SFC diagram, from top to bottom and from left to right. If the conditions are fulfilled, then CODESYS executes the entry actions.

   CODESYS executes an entry action as soon as the transition of the preceding step has been processed and yields `TRUE`, thus indicating that the step has been activated.

4. **Time check / Execute step actions**

   CODESYS performs the following check for each step in the order of the SFC layout:

   - CODESYS copies the elapsed time of the active step to the respective implicit step variable `<step name>.t`. (not yet implemented)
   - If a timeout occurs, then CODESYS sets the respective error flags. (not yet implemented)
   - For non-IEC steps: CODESYS executes the step action.

5. **Execute IEC actions**

   CODESYS executes the IEC actions in alphabetical order, passing through the list of actions two times. In the first pass, CODESYS executes the IEC actions for each step that was deactivated in the preceding cycle. In the second pass, the IEC actions are executed for each active step.

6. **Transition check / Activate next steps**

   The transitions are passed as follows: If a step is active in the current cycle and the subsequent transition yields `TRUE` and any defined minimum time of the step has elapsed, then the subsequent step is activated.

NOTICE!

Please note when executing actions:

An action can be executed multiple times within the same cycle if you use it in multiple SFC diagrams. For example, if a sequential function chart includes two IEC actions A and B, both of which are programmed in SFC and call an IEC action C, then the IEC action C is called two times.

If you use the same IEC action at the same time in different levels of an SFC diagram, then this can lead to unpredictable results when processing. For this reason, CODESYS issues a corresponding error message. This error message can appear for projects that have been created in an earlier version of the development system.

Please note: It is possible to use implicit variables to monitor the processing status of steps and actions and to control processing.

See also

- *Chapter 1.4.1.19.1.4.5 “Implicit variables” on page 480*
- *Chapter 1.4.1.19.1.4.4 “Qualifiers for Actions in SFC” on page 479*

Qualifiers for Actions in SFC

You assign qualifiers to IEC steps. Qualifiers describe how a step action is processed.

Qualifiers are processed by the `SFCActionControl` function block in the library `IecSfc.library`. The library is automatically integrated into the project by the SFC plug-in.

Table 31: Available qualifiers

`N`	Non-stored	The action is active as long as the step.
`R`	overriding Reset	The action is deactivated.
`S`	Set (Stored)	CODESYS executes this action as soon as the step is active. The action execution is continued even when the step has been deactivated until it gets a reset.
`L`	time Limited	CODESYS executes this action as soon as the step is active. The action is executed until the step is deactivated or the given time span has elapsed.
`D`	time Delayed	CODESYS begins executing the action only after the given delay time has elapsed following step activation and the step is still active. The action is executed until the step is deactivated.
`P`	Pulse	CODESYS executes the action exactly two times: one time when the step is activated and one time when the step is deactivated.
`SD`	Stored and time Delayed	CODESYS begins executing the action only after the given delay time has elapsed following step activation. The action is executed until it gets a reset.
`DS`	Delayed and Stored	CODESYS begins executing the action only after the given delay time has elapsed following step activation and the step is still active. The action is executed until it gets a reset.
`SL`	Stored and time limited	CODESYS executes this action as soon as the step is activated. It is executed until the specified time has elapsed or it gets a reset.

You have to specify the times for the L, D, SD, DS, and SL qualifiers in the format of a TIME constant.

When an IEC action is deactivated, it is executed one more time. This means that CODESYS executes this kind of action at least two times. This also applies to actions with the `P` qualifier.

See also

- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*

Implicit variables

Every SFC object supplies implicit variables for you to monitor the status of steps and IEC actions at runtime. These implicit variables are declared automatically by CODESYS for each step and each IEC action.

The implicit variables are structure instances of the type `SFCStepType` for steps and type `SFCActionType` for actions. The variables have the same names as their elements, for example "step1" variable name for "step1" step name. The structure members describe the status of a step or action or the currently elapsed time in an active step.

In the element properties, you can define whether CODESYS should export a symbol definition for this flag to the symbol configuration.

See also

- *Chapter 1.4.1.19.1.4.8.6 “SFC element properties” on page 493*

Step and action status

Syntax for the implicit variable declaration:

```
<step name>:SFCStepType;
_<action name>:SFCActionType;
```

Table 32: The following implicit variables are available for step or IEC action status:

Step	
`<step name>.x`	Shows the activation status in the current cycle. When `<step name>.x = TRUE`, CODESYS processes the step in the current cycle.
`<step name>._x`	Shows the activation status for the next cycle. When `<step name>._x = TRUE` and `<step name>.x = FALSE`, CODESYS processes the step in the next cycle. This means that `<step name>._x` is copied to `<step name>.x` at the beginning of a cycle.
`<step name>.t`	The flag `t` yields the current elapsed time since the step was activated. This applies only to steps, regardless of whether a minimum time has been defined or not in the step properties. Also see SFC flag `SFCError`.
`<step name>._t`	For internal use only
IEC action	
`_<action name>.x`	`TRUE` when the action is being executed.
`_<action name>._x`	`TRUE` when the action is active.

NOTICE!
You can use the above variables to force a specific status value to a step (activate a step). However, note that this can cause an unstable status in the SFC.

See also

- *Chapter 1.4.1.19.1.4.6 “SFC Flags” on page 481*

Access to implicit variables

Syntax for access:

Assign the implicit variable directly in the POU: `<variable name>:=<step name>.<implicit variable>` or `<variable name>:=_<action name>.<implicit variable>`

Example

```
status:=step1._x;
```

From another POU, with the POU name: `<variable name>:=<POU name>.<step name>.<implicit variable>` or `<variable name>:=<POU name>._<action name>.<implicit variable>`

Example

```
status:=SFC_prog.step1._x;
```

Symbol generation

In the element properties of a step or action, you define whether CODESYS should add a symbol definition for the step or action flag. In the *"Properties"* view, you have to select the necessary access rights in the *"Symbol"* column.

See also

- *Chapter 1.4.1.19.1.4.8.6 "SFC element properties" on page 493*

SFC Flags

SFC flags are implicitly generated variables with predefined names. You can use them to influence the processing of an SFC diagram. You can use these flags, for example, to display timeouts or reset step chains. In addition, you can activate jogging mode specifically to activate transitions. You have to declare and activate these variables in order to have access to them.

SFC flags

Name	Data Type	Description
`SFCInit`	Bool	`TRUE`: CODESYS resets the sequence to the initial step. The other SFC flags are also reset (initialization). While the variable is `TRUE`, the initial step remains set (active), but its actions are not executed. Only when you set `SFCInit` again to `FALSE` is the POU further processed normally.
`SFCReset`	Bool	This function behaves similar to `SFCInit`. However, CODESYS continues processing after the initialization of the initial step. For example, in the initial step, you could immediately reset the `SFCReset` flag to `FALSE`.
`SFCError`	Bool	`TRUE` if a timeout occurs in an SFC diagram. If second timeout occurs in the program, it is not registered unless you previously reset the variable `SFCError`. The declaration of `SFCError` is a requirement for other flag variables to function for controlling the chronological sequence (*"SFCErrorStep"*, `SFCErrorPOU`, `SFCQuitError`).
`SFCEnableLimit`	Bool	Used specifically for activating (`TRUE`) and deactivating (`FALSE`) the timeout control in steps using `SFCError`. If you declare and activate this variable (SFC settings), then you must set it to `TRUE` for `SFCError` to work. If you do not, then the timeouts are ignored. The is useful, for example, at start-up or in manual operation. If you do not declare the variable, then `SFCError` will work automatically. The requirement is the declaration of `SFCError`.
`SFCErrorStep`	String	Stores the name of the step that caused a timeout, which was registered by `SFCError`. The name is kept until the registered step error is reset by means of `SFCQuitError` (FALSE -> TRUE). The requirement is the declaration of `SFCError`.

Name	Data Type	Description
SFCErrorPOU	String	Stores the name of the block in which a timeout occurred and was registered by SFCError. The name is saved until the timeout is reset by SFCQuitError. The requirement is the declaration of SFCError.
SFCQuitError	Bool	As long as this Boolean variable is TRUE, CODESYS pauses the processing of the SFC diagram and any timeout, saved in the variable SFCError, is reset. If you reset the variable to FALSE, then all previous times in the active steps are reset. The requirement is the declaration of SFCError.
SFCPause	Bool	As long as this variable is TRUE, CODESYS pauses the processing of the SFC diagram.
SFCTrans	Bool	TRUE if a transition is active.
SFCCurrentStep	String	Shows the name of the active step, regardless of the time monitoring. In parallel branches, the name of the step of the rightmost branch line is always stored.
SFCTip, SFCTipMode	Bool	Controls the jogging mode of the SFC block. If you enable this flag with SFCTipMode=TRUE, then you can activate the next step only by setting SFCTip to TRUE. While SFCTipMode is set to FALSE, transitions can also be used to continue activation.
SFCErrorAnalyzation,		Contains as string all variables that contribute to the total value TRUE of SFCError (timeout in one step). SFCError needs to be activated for this. SFCErrorAnalyzation implicitly uses the function of the POU AnalyzeExpression of the library Analyzation.
SFCErrorAnalyzationTable,		Contains in a table all variables that contribute to the total value TRUE of SFCError (timeout in one step). SFCError needs to be activated for this. SFCErrorAnalyzationTable implicitly uses the function of the POU AnalyzeExpressionTable of the library Analyzation.

Implicit generation of SFC flags

CODESYS declares SFC flags automatically when you activate the respective options. You can set this option in the *“SFC Settings”* tab of the properties dialog for each POU, or in the *“SFC”* project settings dialog for each SFC POU in the project.

The SFC settings for the SFC flags of individual POUs are effective only if you have not selected the “Use defaults” option. When you select this option, the settings apply that were defined in the project settings.

SFC flags that you declare in the SFC settings dialog are visible only in the online view of the SFC block.

See also

- *“Flag” on page 1166*

Explicit generation of SFC flags

Manual declaration, which was necessary in CoDeSys V2.3, is now only required to enable write access from another block. In this case, you should note that when you declare the flag in a global variable list, you must deactivate its *“Declare”* setting in the SFC settings dialog. If you do not do this, then a local SFC flag is implicitly declared that CODESYS uses instead of the global variable.

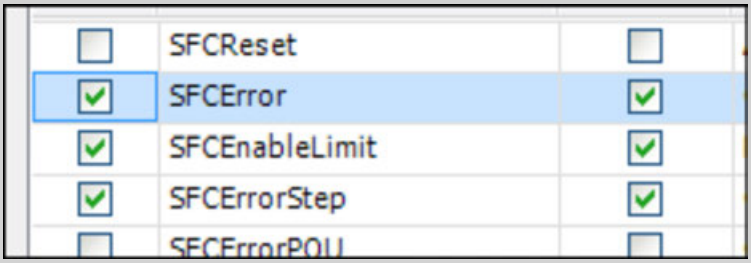
Application example for SFCError

Example

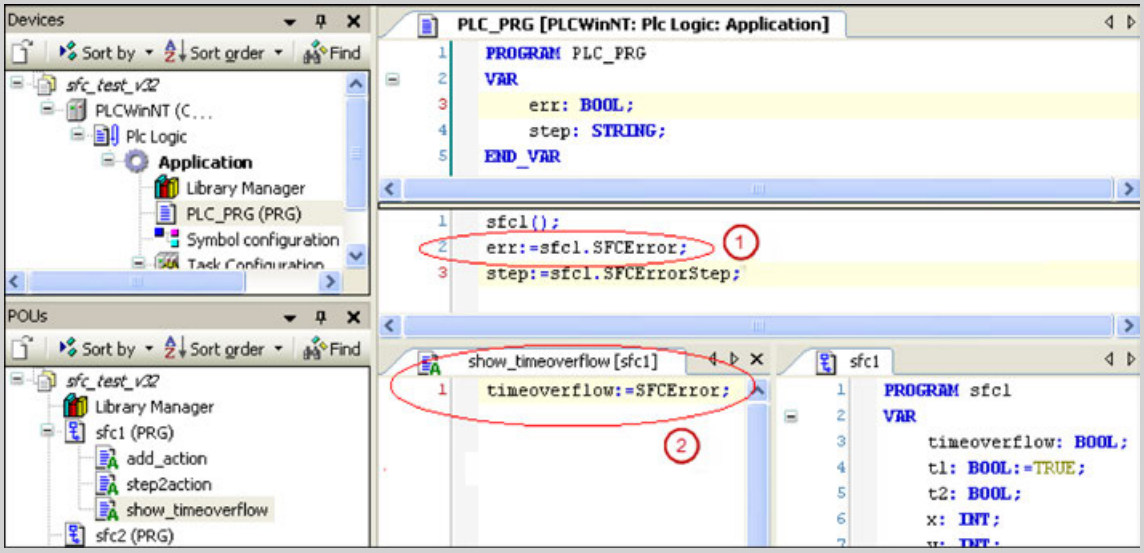
You have created an SFC block named `sfc1`, which contains the `s1` step. You have defined timeouts in the step properties. (See "Online view of SFC block `sfc1`" below.)

If for any reason the `s1` step remains active longer than its time properties have permitted (timeout), then CODESYS sets the `SFCError` flag to permit access by the application.

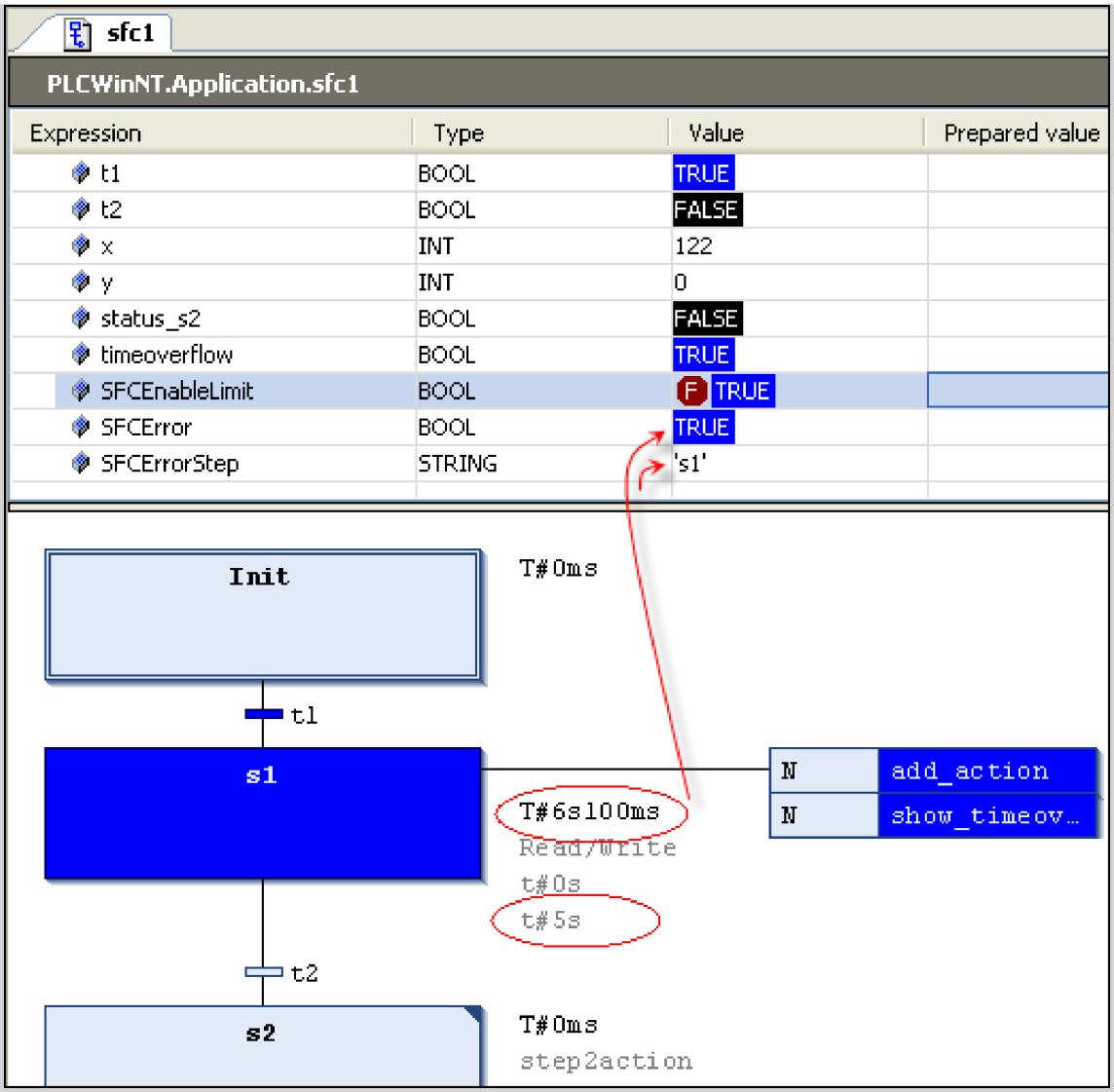
To permit access, you have to declare and activate the SFC flag in the SFC settings. If you have only declared it, then the SFC flag is only displayed in the online view of `sfc1` in the declaration part, but it has no function.



Now the SFC flag can be referenced within the POU, for example in an action (2) or outside of the block (1).



Online view of the SFC block `sfc1`



`SFCError` is `TRUE` as soon as a timeout occurs within `sfc2`.

Note that you can use the flags `SFCErrorAnalyzation` and `SFCErrorAnalyzationTable` to determine the components of the expression that contributes to the value `TRUE` of the `SFCError`.

See also

- *Chapter 1.4.1.19.1.4.7 “Library "Analyzation"” on page 485*

Access to the flags

Syntax for access:

You assign the flag directly within the POU: `<variable name>:=<SFC flag>`

Example

```
checkerror:=SFCerror;
```

From another POU with POU name: `<variable name>:=<POU name>.<SFC flag>`

Example:

```
checkerror:=SFC_prog.SFCerror;
```

If you need write access from another block, then you also have to declare the SFC flag explicitly as a VAR_INPUT variable in the SFC block or globally in a GVL.

Example

Local declaration:

```
PROGRAM SFC_prog
VAR_INPUT
  SFCinit:BOOL;
END_VAR
```

Global declaration in a global variable list:

```
VAR_GLOBAL
  SFCinit:BOOL;
END_VAR
```

```
PROGRAM PLC_PRG
VAR
  setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit;  // write access to SFCinit in SFC_prog
```

See also

- *Chapter 1.4.1.19.1.4.7 “Library "Analyzation"” on page 485*

Library "Analyzation"

This library contains POUs for the analysis of expressions. When a composite expression has the total value of `FALSE`, those of its components that contribute to this result can be determined. In the SFC editor, the flags `SFCErrorAnalyzation` and `SFCErrorAnalyzationTable` use these functions implicitly to examine the transition expressions. Then the flags provide the identifiers of the variables that contributed to a timeout error. They keep this information until they are reset explicitly by means of the SFC flag `SFCQuitError`.

An analysis POU cannot be called by means of a pointer. This kind of call is ignored. Call the POU as a single instance.

For a description of the library POUs and an example of how the SFC flags display the analysis results in CODESYS, see the documentation for the library (online help or directly in the Library Manager).

See also

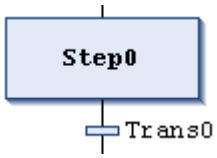
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.19.1.4.6 “SFC Flags” on page 481*

Elements

SFC elements 'Step' and 'Transition'

Step symbol ; Transition symbol

As a rule, CODESYS inserts steps and transitions as combinations. Inserting a step without a transition or a transition without a step causes an error when compiling. You can modify this by double-clicking the name.



NOTICE!
Step names must be unique within the scope of the parent block. Consider this especially when using actions that were also programmed in SFC.

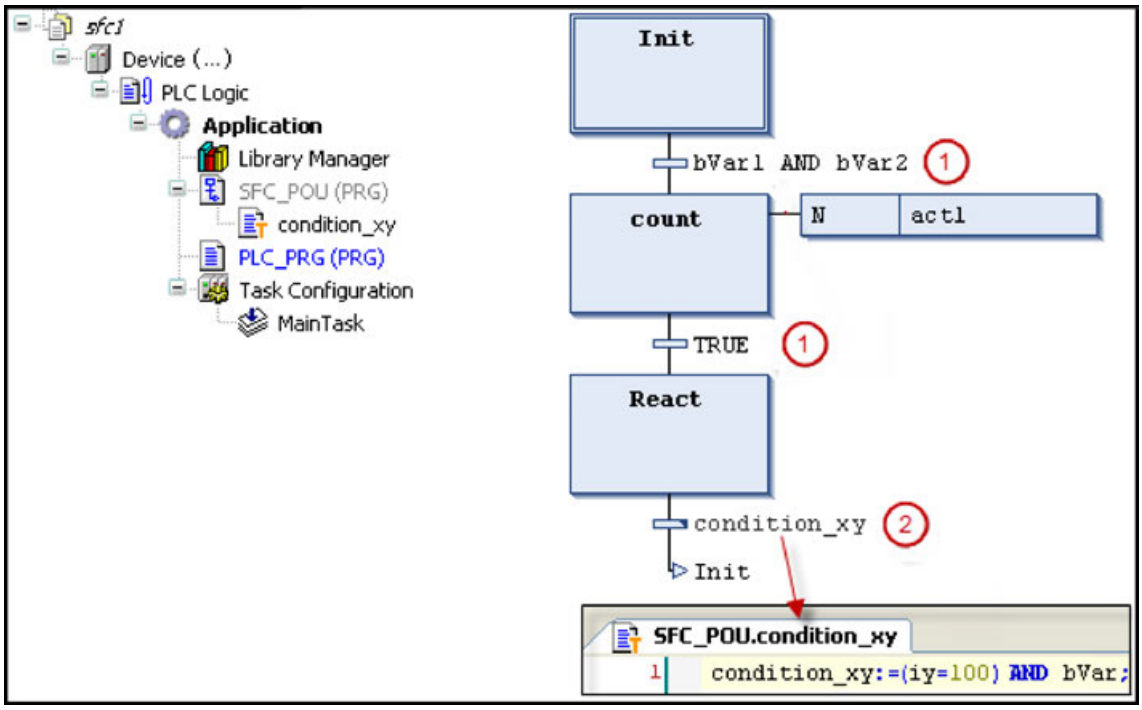
Please note that you can convert a step into an initial step by clicking *"Init step"* or by setting the respective property in the SFC properties.

All steps are defined by the step properties, which you can display and edit in the *"Properties"* view, depending on the set options.

You have to add those actions to the step which are to be executed when the step is active. A distinction is made between IEC actions and step actions. Details for this are found in the chapter about the SFC element "Action".

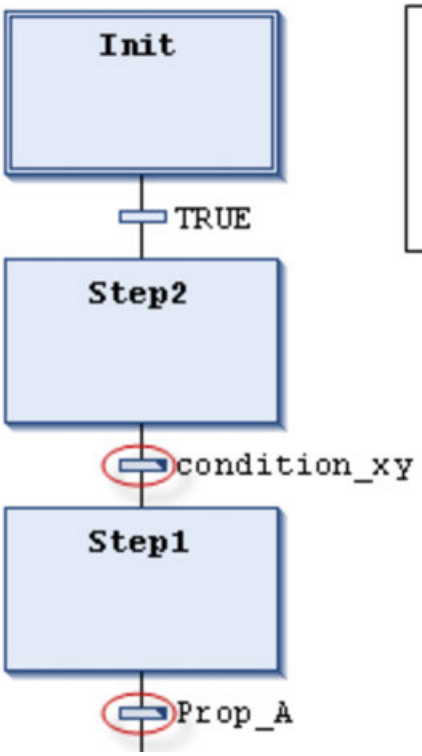
A transition must include the condition for the subsequent step to be active as soon as the value of the condition yields `TRUE`. Therefore, a transition condition must yield `TRUE` or `FALSE`. It can be defined in one of two ways:

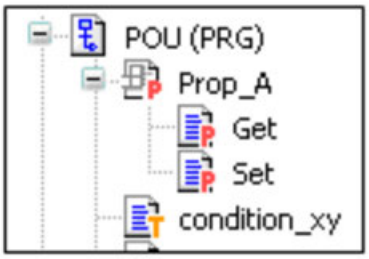
- (1) Inline condition (direct): You replace the default transition name with either the name of a Boolean variable, a Boolean address, a Boolean constant, or a statement with a Boolean result, for example `(i<100) AND b`. You cannot specify programs, function blocks, or assignments here.
- (2) Multi-use condition (separate transition or property object): You replace the default transition name with the name of a transition or property object (,). You create these objects by clicking *"Project ➔ Add Object"*. This allows multiple use of transitions, for example "condition_xy" in the figures below. Like an inline condition, the object can contain a Boolean variable, Boolean address, Boolean constant, or an statement with a Boolean result. In addition, it can also contain multiple statements with any code.



NOTICE!
The user is responsible for assigning the required expression to a transition variable if the transition includes multiple instructions.

Transitions that reference a transition or property object are marked with a small triangle in the upper right corner of the transition box.





As opposed to CoDeSys V2.3, now CODESYS treats a transition condition like a method call. The entry has the following syntax:

```
<transition name>:=<transition condition>
```

(for example `trans1:= a=100`)

or only

`<transition condition>`

(for example `a=100`)

You will find an example (condition_xy) in the figure above.

See also

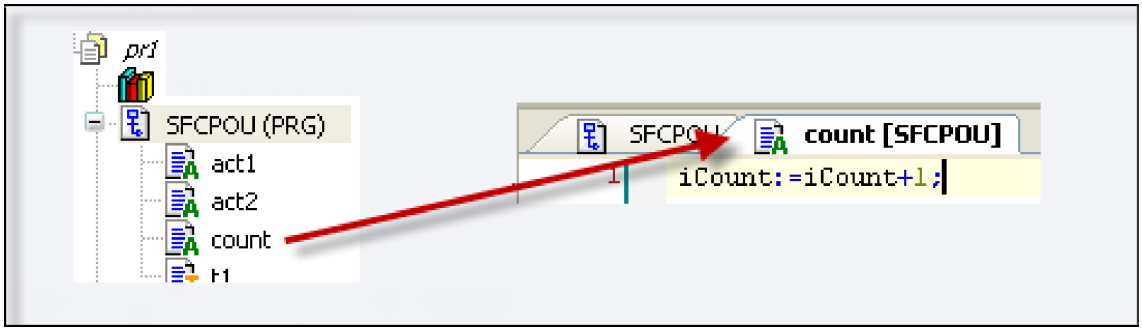
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.20.3.11.6 “Command 'Insert Step-Transition'” on page 1081*
- *Chapter 1.4.1.19.1.4.8.2 “SFC Element 'Action'” on page 488*
- *Chapter 1.4.1.20.3.11.1 “Command 'Init Step'” on page 1079*
- *Chapter 1.4.1.19.1.4.8.6 “SFC element properties” on page 493*
- *Chapter 1.4.1.8.22.4 “Calling methods” on page 314*

SFC Element 'Action'

Symbol: 

An action includes one or more statements in one of the valid implementation languages. You can assign an action to a step.

Actions that you use in SFC steps have to be created as POUs in the project.



Exception: In the case of IEC actions, which you add to a step as action association, you can also specify a Boolean variable instead of an action object. The value of these variables is switched between `FALSE` and `TRUE` each time the action is executed.

> **NOTICE!**
> You have to define unique step names within the scope of the parent block. An action written in SFC must not contain a step with a name identical to the step to which the action is assigned.

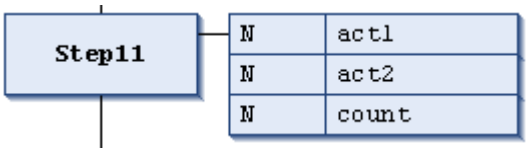
A distinction is made between IEC actions and step actions:

1. IEC actions

IEC actions comply with the IEC 61131-3 standard. They are executed according to their qualifiers.

IEC actions are executed two times: first when the step is activated and second when the step is deactivated. If you assign multiple actions to one step, then the action list is processed from top to bottom.

Each action box includes the qualifier in the first column and the action name in the second column. Both can be edited directly.



NOTICE!
When the same global Boolean variable is associated as an IEC action in different SFC POUs, unwanted overwriting can result.

In contrast to step actions, you can use different qualifiers for IEC actions. Moreover, each IEC action is provided with a control flag. This directs CODESYS to execute an action only one time at any moment, even if the action is called by another step at the same time. This cannot be guaranteed for step actions.

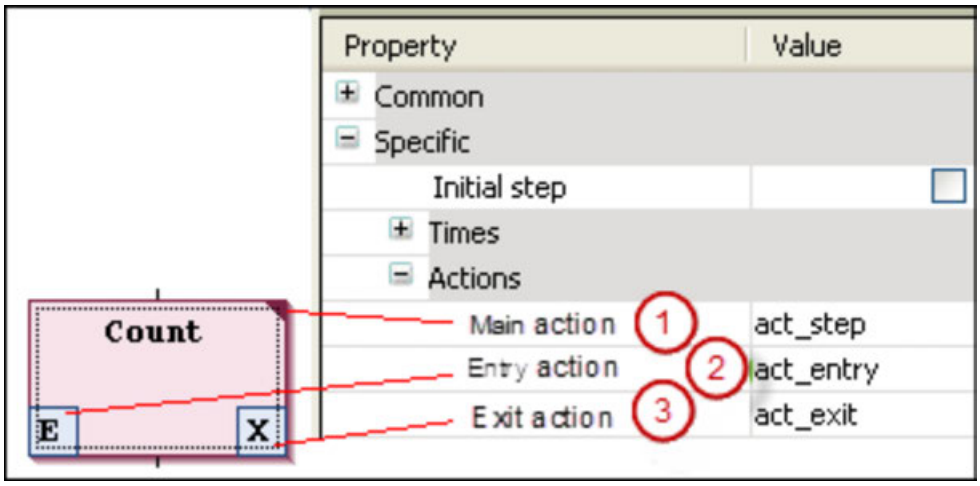
You assign IEC actions to steps by clicking *"SFC ➔ Insert Action Association"*.

See also

- *Chapter 1.4.1.20.3.11.14 "Command 'Insert Action Association'" on page 1084*
- *Chapter 1.4.1.19.1.4.4 "Qualifiers for Actions in SFC" on page 479*

2. Step actions These are actions that you can use to extend the IEC standard.

- **Entry action**:
  CODESYS executes this action after the step is activated and before the main action is executed.
  You reference a new action, or an action created below the SFC object, from a step by means of the *"Entry action"* element property (2). You can also add a new action to the step by means of the *"Add Entry Action"* command. The entry action is marked with an `E` in the lower left corner of the step box.
- **Main action**:
  CODESYS executes this action when the step is active and any entry actions have already been processed. However, in contrast to IEC actions (see above), these step actions are not executed a second time when the step is deactivated. Moreover, you cannot use qualifiers here.
  You add an existing action to a step by means of the *"Main action"* element property (1). You can create and add a new action by clicking the step element. A main action is marked with a filled triangle in the upper right corner of the step box.
- **Exit action**:
  CODESYS executes this action one time when the step is deactivated. However, note that an exit action is not executed in the same cycle, but at the beginning of the next cycle.
  You reference a new action, or an action created below the SFC object, from a step by means of the *"Exit action"* element property (3). You can also add a new action to the step by means of the *"Insert Exit Action"* command. The exit action is marked with an `X` in the lower right corner of the step box.



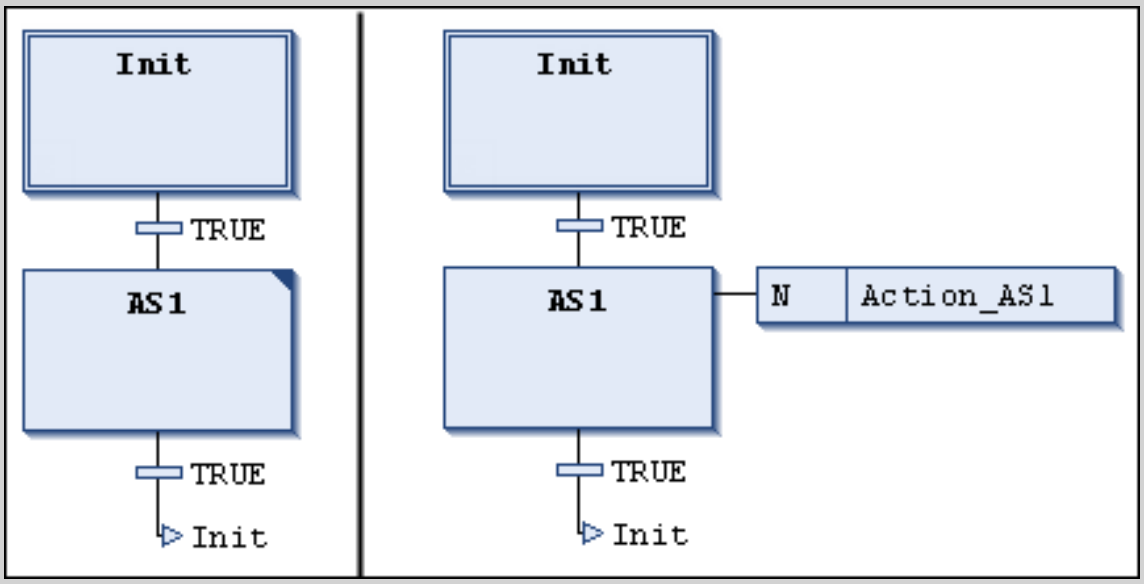
See also

- *Chapter 1.4.1.19.1.4.8.6 “SFC element properties” on page 493*

Difference between IEC actions and step actions

The main difference between step actions and IEC actions with a qualifier N is that an IEC action is always executed two times: when the step is activated and when the step is deactivated. See the following example:

Example



You have attached the `Action_AS1` action to the `AS1` step as a step action (left) and as an IEC action with qualifier N (right). Because two transitions are activated in each case, the time to reach the initial step again is two PLC cycles. This is true as long as the `iCounter` counter variable was initialized at 0 and then incremented in the `Action_AS1` action. After the `Init` step is reactivated, `iCounter` returns a value of 1 in the example on the left. In the example on the right, a value of 2 is returned because the IEC action is executed a second time due to the deactivation of `AS1`.

Another difference: Step actions can be pseudo-embedded. In this case, they can be called only from the related step. If you copy this step, CODESYS creates new action objects automatically and copies the respective implementation code. You define whether or not a step action is embedded, either when the first action is inserted into the step, or later in the *“Duplicate when copying”* element property. In general, this behavior can also be preset in the SFC options.

Moreover, for IEC actions, a Boolean variable can be specified instead of an action object. This is not possible for step actions.

SFC element 'Branch'

Symbol

Use branches to program parallel or alternative sequences in the sequential function chart.

For alternative branches, CODESYS processes just one of the branch lines at a time, depending on the preceding transition condition. Parallel branches are processed at the same time.

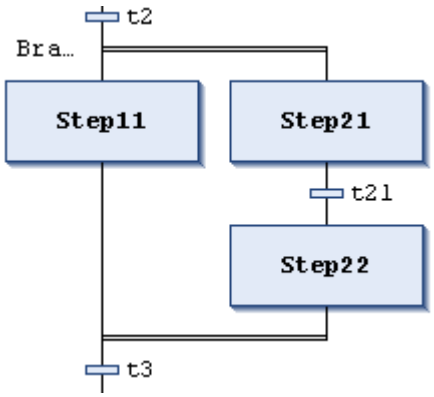
See also

- *Chapter 1.4.1.19.1.4.3 “Processing order in SFC” on page 477*
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.20.3.11.13 “Command 'Insert Branch Right'” on page 1083*

Parallel branch

For parallel branches, the branch lines must begin and end with steps. Parallel branch lines can contain additional branches.

The horizontal lines before and after the branch are double lines.



Processing in online mode: If the preceding transition (t2 in the example) yields `TRUE`, then the first steps in all parallel branch lines are active (Step11 and Step21). CODESYS processes the individual branch lines at the same time and the subsequent transition is passed afterwards (t3).

The "Branch<n>" jump marker is added automatically to the horizontal line that indicates the beginning of a branch. You can define this marker as the jump destination.

Please note that you can convert a parallel branch into an alternative branch by clicking *“Alternative”*.

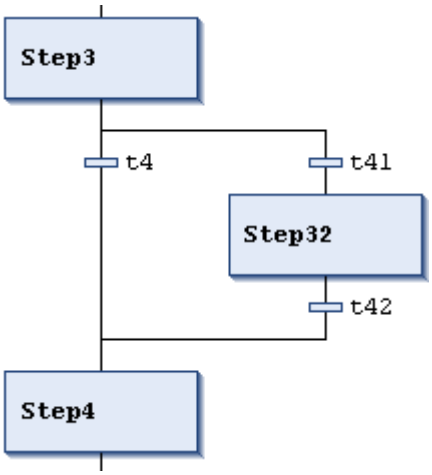
See also

- *Chapter 1.4.1.20.3.11.11 “Command 'Alternative'” on page 1083*

Alternative branch

The horizontal line before and after the branch is a single line.

In an alternative branch, the branch lines must begin and end with transitions. The branch lines can contain additional branches.



If the step before the branch is active, then CODESYS passes the first transition of each alternative branch line from left to right. For the first transition that yields `TRUE`, the associated branch line opens, thus activating the step following the transition.

Please note that you can convert an alternative branch into a parallel branch by clicking *“Parallel”*.

See also

- *Chapter 1.4.1.20.3.11.10 “Command 'Parallel'” on page 1082*

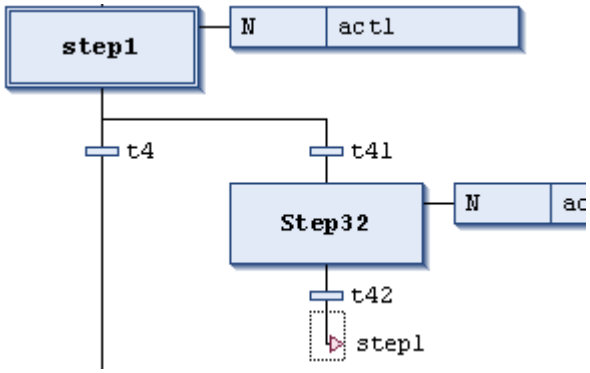
SFC element 'Jump'

Symbol

Use a jump to define which actions in a step should be executed next as soon as the transition preceding the jump is `TRUE`. Jumps may become necessary, as execution paths cannot cross or lead upwards.

Excluding the required jump at the end of a diagram, you can generally insert jumps only at the end of a branch.

The destination of a jump is defined by the added text string, which you can edit directly. The jump destination can be a step name or the marker for a parallel branch.



See also

- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.20.3.11.16 “Command 'Insert Jump'” on page 1085*

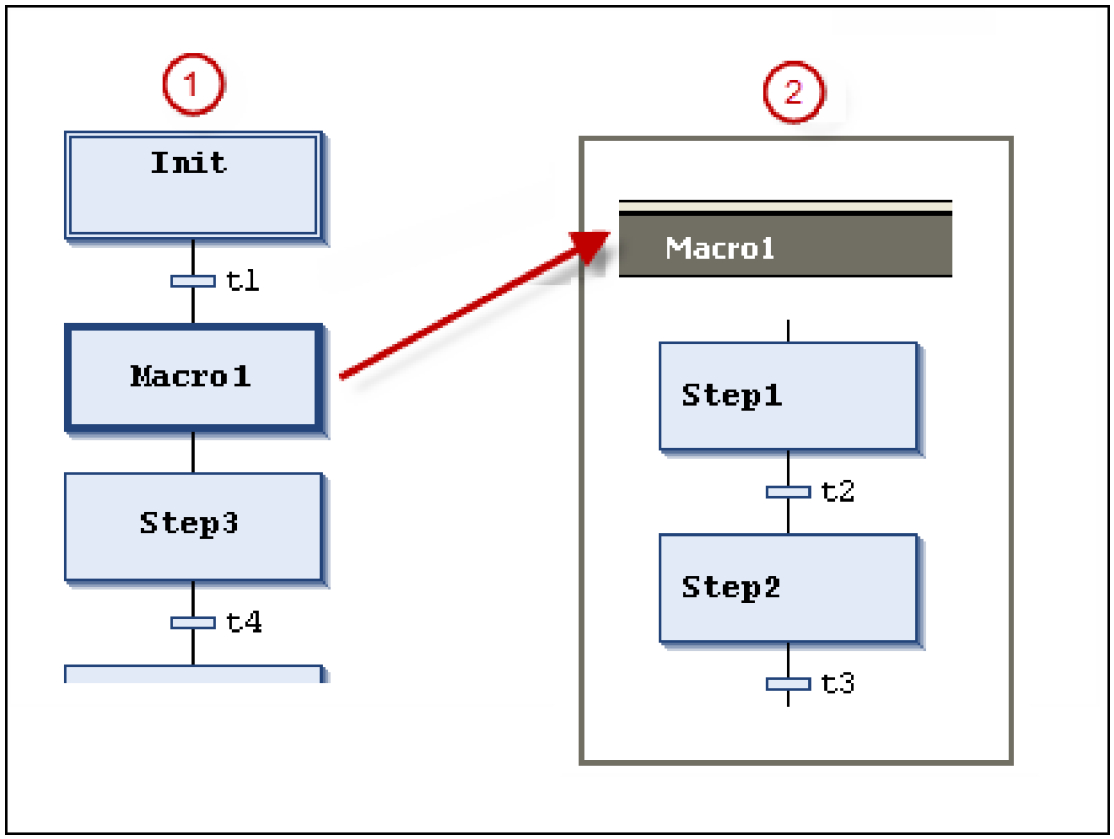
SFC element 'Macro'

Symbol

A macro includes part of the SFC diagram, but it is not displayed in detail in the main view of the editor.

Using macros does not influence the processing flow. Macros are used for hiding specific parts of the diagram, for example to increase overall clarity.

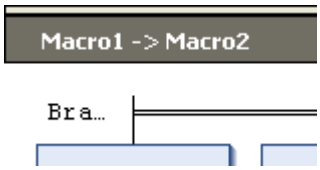
You open the macro editor by double-clicking the macro box or by clicking *"SFC ➔ Zoom Into Macro"*. You can program here just like in the main view of the SFC editor. To close the macro editor, click *"SFC ➔ Zoom Out of Macro"*.



① Main view in the SFC editor

② Macro editor view for Macro1

Macros can also include other macros. The caption of the macro editor always shows the path of the open macro within the diagram, for example:



See also

- *Chapter 1.4.1.8.3.4.1 "Programming in SFC" on page 255*
- *Chapter 1.4.1.20.3.11.20 "Command 'Zoom Into Macro'" on page 1086*
- *Chapter 1.4.1.20.3.11.21 "Command 'Zoom Out of Macro'" on page 1086*

SFC element properties

You edit the properties of an SFC element in the *"Properties"* view. Click *"View ➔ Element Properties"* to open this view. The properties to be displayed depend on the currently selected element.

The properties that are displayed in the SFC diagram next to the element depend on the settings in the “View” tab of the SFC editor options.

General

Property	Value description
“Name”	Element name, by default "<element><consecutive number>", for example step name "Step0", "Step1", branch name "Branch0”, etc.
“Comment”	Element comment in text, for example “counter reset”. You can insert line breaks by pressing *[Ctrl]+[Enter]*.
“Symbol”	For each SFC element, CODESYS declares an implicit variable with the same name as the element. The configuration determines whether this flag variable should be exported to the symbol configuration and which access rights for the symbol should be applied in the PLC. ● *“No access”*: The symbol is exported to the symbol configuration but cannot be accessed from the PLC. ● *“Read”*: The symbol is exported to the symbol configuration and can be read from the PLC. ● *“Write”*: The symbol is exported to the symbol configuration and can be written from the PLC. ● *“Read/Write”*: Combination of read and write. ● Empty: A symbol is not exported to the symbol configuration.

Specific

Property	Value description
“Init step”	☑: This option is activated only for the defined initial step. By default, this is the first step in an SFC diagram. Note: If you activate this property for another step, then it must be deactivated in the previous step to prevent compilation errors.
“Duplicate when copying”	This option is available for steps that contain a step action (entry action, main action, or exit action), and for transitions that are linked to a transition object. ☑: When copying the step or transition, a new object is created for each called action or transition. It contains a copy of the implementation code of the copied object. ☐: When copying a step or transition, the link to the called object is retained for the respective action or transition. No new objects are generated. The source and the copies of the step or transition call the same action or transition.

Property	Value description
"Times" ● *"Minimum active"* ● *"Maximum active"*	Minimum time that the step is active, even when the subsequent transition is TRUE. Maximum time that the step can be active. If this time is exceeded, then CODESYS sets the `SFCError` implicit variable to TRUE. Times according to IEC syntax (for example t#8s) or the TIME variable; default: t#0s.
"Actions" ● *"Entry action"* ● *"Step action"* ● *"Exit action"*	● *"Entry action"*: CODESYS executes these actions after activating the step. ● *"Step action"*: CODESYS executes this action when the step is active and any entry actions have already been processed. ● *"Exit action"*: CODESYS executes this action in the subsequent cycle when the step is deactivated. Please note the processing sequence.

When using the respective implicit SFC variables and flags, you receive information about the status of a step or an action or about timeouts.

See also

- *Chapter 1.4.1.20.4.13.22 " Dialog 'Options' - 'SFC Editor'" on page 1200*
- *Chapter 1.4.1.19.1.4.5 "Implicit variables" on page 480*
- *Chapter 1.4.1.19.1.4.8.2 "SFC Element 'Action'" on page 488*

Function Block Diagram / Ladder Diagram / Instruction List (FBD/LD/IL)

FBD/LD/IL Editor

The FBD/LD/IL editor is a combined editor of the programming languages FBD, LD and IL.

If necessary, IL can be activated in the CODESYS options.

There is a common set of commands and elements and CODESYS automatically converts the 3 programming languages into one another internally.

The code in the implementation part is structured in all three languages with the aid of networks.

The *"FBD/LD/IL"* menu provides the commands for working in the editor.

In offline and online modes, you can switch editors at any time by using the menu command in *"View"*.

The behavior of the FBD/LD/IL editor is defined by the settings in *"Tools ➔ Options"* (category *"FBD, LD and IL"*).

NOTICE!

There are some special elements that CODESYS cannot convert and thus it displays only in the applicable language. There are also constructs that are not clearly convertible between IL and FBD and are therefore 'normalized', i.e. nullified, when converted back to FBD. This concerns: negation of expressions and explicit/implicit assignment of function block inputs and outputs.

An error-free conversion between the languages requires syntactically correct code. Otherwise parts of the implementation can be lost.

See also

- *Chapter 1.4.1.8.3.1 “FBD/LD/IL” on page 235* (programming)
- *Chapter 1.4.1.20.3.13 “Menu 'FBD/LD/IL'” on page 1104* (commands)
- *Chapter 1.4.1.20.4.13.9 “Dialog 'Options' - 'FBD, LD, and IL'” on page 1192*
- *Chapter 1.4.1.19.1.2 “Common functions in graphical editors” on page 462*

FBD and LD editor

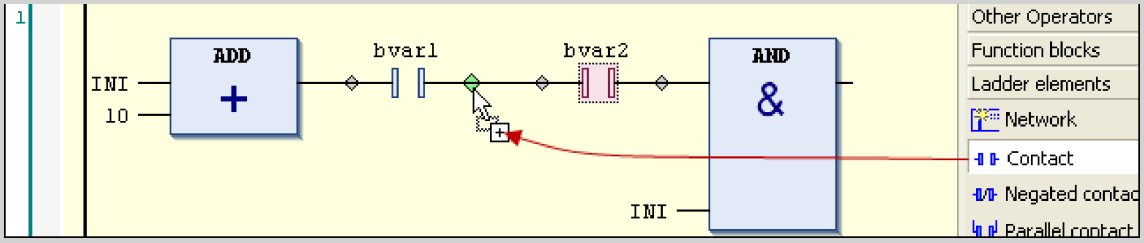
Inserting and arranging elements

You can drag elements with the mouse from the view *“Tools”* (toolbox) into the implementation part of the editor. Alternatively you can use the commands of the context menu or the *“FBD/LD/IL”* menu.

Settings for the display and interface are defined in the CODESYS options, category *“FBD/LD/IL”*.

If you drag an element with the mouse over a network in the editor, all possible insertion positions are displayed with gray diamond-shaped, triangular or arrow-shaped position marks. As soon as the mouse pointer is located over one of these marks, the mark turns green. If the mouse button is now released, CODESYS inserts the element at this position.

Example



If you drag a function block or an operator from the toolbox or a network at the left-hand side of the network onto one of the two arrows, then CODESYS automatically creates a new network and inserts the element there.

In order to replace an element, drag a suitable other element onto its position with the mouse. Elements that you can replace by the new element are marked by CODESYS in the editor with text fields, for example *“Replace”*, *“Attach input”*.

You can use the usual commands in the menu *“Edit”* for cutting, copying, pasting and deleting elements. Copying also works with drag-and-drop by holding down the *[Ctrl]* key.

NOTICE!

The operators with EN/ENO functionality can only be inserted in the FBD and IL editors.

Selecting elements

A box or a connecting line in the editor is selected by clicking on it with the mouse so that it has the focus. Multiple selection is possible by keeping the *[Ctrl]* key pressed. A selected element is shaded red.

Tooltip

If the cursor points to certain elements, for example to a variable or to an input, a tooltip appears showing information about this element.

In the case of elements underlined with a wavy red line, the tooltip shows the pre-compile error message of the error that occurs with this element.

Navigating in the editor

Table 33: Navigating in the editor

With the help of the keys and commands described below, you can place the focus within the editor on a different cursor position. The change between the positions is also network-spanning in function.	
[←] *[→]*	Change to the neighboring cursor position, along the signal flow, i.e. from left to right and vice versa.
[↑][↓] *[↓]*	Change to the next cursor position above or below the current position, if this neighboring position belongs to the same logical group. For example, all connections of a box form a logical group. If such a logical group does not exist: change to the first cursor position in the next higher or lower neighboring element. In the case of parallel-connected elements, navigation takes place along the first branch.
[Ctrl] + [Home]	Change to the first network; this will be selected.
[Ctrl] + [End]	Change to the last network; this will be selected.
[Page Up]	Scroll upwards by one page; the highest network on this page is selected.
[Page Down]	Scroll downwards by one page; the lowest network on this page is selected.
Command *“Go to…”*	Change to a certain network.

Opening function blocks

If a function block is inserted in the editor, then you can open its implementation by a double-click or with the context menu command *“Browse for symbol ➔ Go to Definition”*.

See also

- *“Function block diagram (FBD)” on page 235*
- *“Ladder diagram (LD)” on page 235*
- *Chapter 1.4.1.8.3.1.1 “Programming function block diagrams (FBD)” on page 237*
- *Chapter 1.4.1.8.3.1.2 “Programming ladder diagrams (LD)” on page 239*
- *Chapter 1.4.1.19.1.5.4 “Elements” on page 504*
- *Chapter 1.4.1.20.4.13.9 “Dialog 'Options' - 'FBD, LD, and IL'” on page 1192*
- *Chapter 1.4.1.19.1.5.2 “FBD/LD/IL editor in online mode” on page 499*
- *Chapter 1.4.1.20.3.13.44 “Command 'Go to'” on page 1116*

IL editor

Inserting and arranging elements:

You can insert elements with the help of the commands of the menu *“FBD/LD/IL”* in the context menu. You can also drag a new network from the tool box into the implementation part of the editor by drag-and-drop.

You can use the usual commands in the menu *“Edit”* for cutting, copying, pasting and deleting elements. Copying also works with drag-and-drop by holding down the *[Ctrl]* key.

NOTICE!
Please note that operators with EN/ENO functionality can only be inserted in the FBD and IL editors.

Each program line is entered in a table row.

Table 34: Structure of networks in the IL editor

1st line: Network title Requirement: The option is activated in the CODESYS options.		
2nd line: Network comment Requirement: The option is activated in the CODESYS options.		
3rd line and on:		
Column	**Contents**	**Description**
1	Operator	Contains the IL operator (LD, ST, CAL, AND, OR, etc.) or a function name. If you call a function block, you must additionally specify the corresponding parameters here; in the preceding field you must enter in this case := or =>.
2	Operand	Contains precisely one operand or the name of a jump label. In the case of several operands you must enter them in several rows and when doing so insert a comma directly behind the individual operands. (See example below)
3	Address	Contains the address of the operand as defined in its declaration. non-editable You can activate/deactivate the display via the option *"Display symbol address"*. To do this, select the command *"Tools ➔ Options"* and the *"General"* tab in the category *"FBD, LD and IL"*.
4	Symbol comment	Contains the comment that was specified for the operand if necessary in the declaration. non-editable You can activate/deactivate the display via the option *"Display symbol comment"* if you select the command *"Tools ➔ Options"* and the *"General"* tab in the category *"FBD, LD and IL"*.
5	Operand comment	Comment for the current program line. You can activate/deactivate the display via the option *"Operand comment"* if you select the command *"Tools ➔ Options"* and the *"General"* tab in the category *"FBD, LD and IL"*.

Example

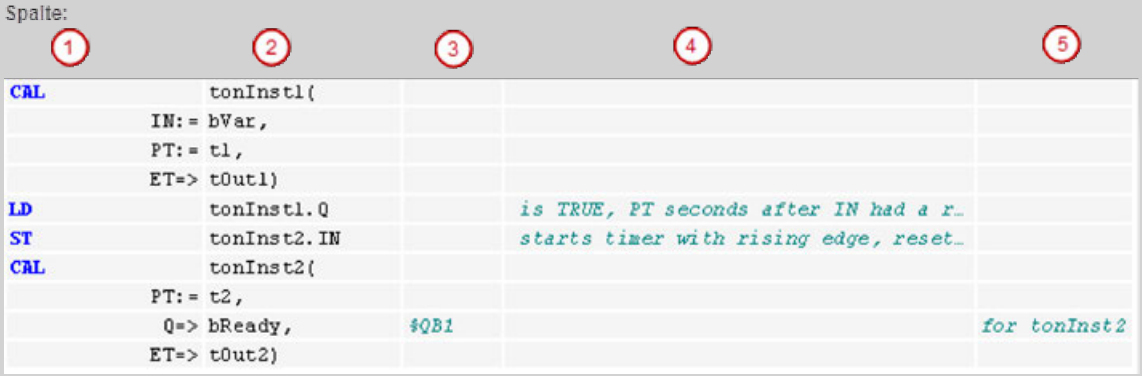


Table 35: Navigating in the editor

Key(s)/ command	Cursor movement
[↑] *[↓]*	Jumps to the field located above/below.
[Tab]	Jumps one field to the right within the row.
[Shift]+ [Tab]	Jumps to the left to the preceding field within the row
[Space]	Opens the editing frame for the selected field. Alternatively you can click with the mouse on the field. If applicable the button for the input assistant dialog box is available.
[Ctrl] + [Enter]	Inserts a new row below the current row.
[Del]	Deletes the current row.
[Ctrl] + [Home]	Sets the focus at the start of the document and marks the first network.
[Ctrl] + [End]	Sets the focus at the end of the document and marks the last network.
[Page Down]	Scrolls up by one page and marks the top rectangle.
[Page Up]	Scrolls down by one page and marks the top rectangle.

See also

- *“Instruction list (IL)” on page 236*
- *Chapter 1.4.1.8.3.1.3 “Programming in instruction list (IL)” on page 240*
- *Chapter 1.4.1.19.1.5.3 “Modifiers and operators in IL” on page 500*
- *Chapter 1.4.1.20.4.13.9 “Dialog 'Options' - 'FBD, LD, and IL'” on page 1192*
- *Chapter 1.4.1.19.1.5.2 “FBD/LD/IL editor in online mode” on page 499*

FBD/LD/IL editor in online mode

FBD/LD/IL editor in online mode

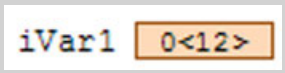
In online mode the current value of each variable is displayed behind the variable in the editor. Writing/forcing and the setting of breakpoints is possible.

If the variable is presently forced, this is indicated directly in front of the forced value by F. If a value has been prepared for writing or forcing, this value is displayed directly behind the current value in square brackets `<value>`.

Example

Forced variable:

Prepared value



In the online view of a ladder diagram (LD) the connecting lines are marked in color: connections with the value `TRUE` are displayed as a thick blue line, connections with the value `FALSE` as a thick black line. Conversely, connections with an unknown or analog value are displayed normally (thin black line).

NOTICE!
Note that values of the connections are calculated from the monitored variables. This is not a genuine flow control.

Breakpoints

Possible positions for breakpoints are in principle the positions at which values of variables can change (instructions), at which the program branches or at which another box is called.

Possible breakpoint positions:

- On the entire network: causes the breakpoint to be set at the first possible position in the network.
- On a box, if the box contains an assignment. Not possible with operator boxes, for example ADD, DIV.
- On assignments.
- At the end of the box at the position of the return to the calling box. In online mode an empty network automatically appears here; it is marked by 'RET' in place of a network number.

NOTICE!
At present you cannot directly set a breakpoint on the first box in the network. However, if you set a breakpoint on the entire network, this breakpoint marking is transferred automatically to the first box in online mode.

NOTICE!
Breakpoints in methods: CODESYS automatically sets a breakpoint in all methods that can be called. Therefore, if a method managed by an interface is called, breakpoints are set in all methods that occur in function blocks that implement this interface as well as in all derived function blocks that use the method. If a method is called by a pointer to a function block, CODESYS sets the breakpoints in the method of the function block and in all derived function blocks that use the method.

See also

- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*

Modifiers and operators in IL

Table 36: Modifiers

Modifier	Combined with operator	Description
C	JMP, CAL, RET	The command is only executed if the result of the preceding expression is TRUE.
N	JMPC, CALC, RETC	The command is only executed if the result of the preceding expression is FALSE.
N	otherwise	negation of the operand (not of the accumulator).

Table 37: Operators with the possible modifiers

Operator	N	Meaning	Example
LD	N	Loads the (negated) the value of the operand into the accumulator.	LD ivar
ST	N	Stores the (negated) content of the accumulator in the operand.	ST iErg
S		Sets the operand (type BOOL) to TRUE if the content of the accumulator is TRUE.	S bVar1

Operator	N	Meaning	Example
R		Sets the operand (type BOOL) to FALSE if the content of the accumulator is TRUE.	R bVar1
AND	N, (	Bitwise AND of the accumulator value and (negated) operand	AND bVar2
OR	N, (	Bitwise OR of the accumulator value and (negated) operand	OR xVar
XOR	N, (	Bitwise exclusive OR of the accumulator value and (negated) operand	XOR N, (bVar1,bVar2)
NOT		Bitwise negation of the accumulator value	
ADD	(	Addition of the accumulator value and the operand; result is written into the accumulator.	ADD ivar1
SUB	(	Subtraction of the operand from the accumulator value; result is written into the accumulator.	SUB iVar2
MUL	(	Multiplication of accumulator value and operand; result is written into the accumulator.	MUL ivar2
DIV	(	Division of the accumulator value by the operand; result is written into the accumulator.	DIV 44
GT	(	Checks whether the accumulator value is greater than the operand value; result (BOOL) is written into the accumulator; >	GT 23
GE	(	Checks whether the accumulator value is greater than or equal to the operand value; result (BOOL) is written into the accumulator.	GE iVar2
EQ	(	Checks whether the accumulator value is equal to the operand value; result (BOOL) is written into the accumulator.	EQ iVar2
NE	(	Checks whether the accumulator value is not equal to the operand value; result (BOOL) is written into the accumulator;	NE iVar1
LE	(	Checks whether the accumulator value is smaller than or equal to the operand value; result (BOOL) is written into the accumulator.	LE 5
LT	(	Checks whether the accumulator value is smaller than the operand value; result (BOOL) is written into the accumulator.	LT cVar1
JMP	CN	Unconditional (conditional) jump to the specified jump label	JMPN next
CAL	CN	(Conditional) call of a program or a function block (if the accumulator value is TRUE)	CAL prog1
RET		Exit the box and return to the calling box.	RET
RET	C	If the accumulator value is TRUE: exit the box and return to the calling box.	RETC
RET	CN	If the accumulator value is FALSE: exit the box and return to the calling box.	RETCN
)		Evaluation of the reset operation	

Example

```
1
    AND     TRUE    load TRUE to accumulator
    ANDN    bVar1   execute AND with negated value of bVar1
    JMPC    m1      if accum. is TRUE, jump to label "m1"

    LDN     bVar2   store negated value of bVar2...
    ST      bRes    ... in bRes
2
    m1:

    LD      bVar2   store value of bVar2...
    ST      bRes    ... in bRes
```

Application	Description	Examples

Several operands for 1 operator	Options
	● You enter the operands into consecutive rows, separated by commas in the 2nd column.
	● You repeat the operator in consecutive rows.
		Variant 1 :
		`LD 2`
		`ADD 3,`
		`4,`
		`6`
		`ST iVAR`
		Variant 2 :
		`LD 2`
		`ADD 3`
		`ADD 4`
		`ADD 6`
		`ST iVAR`

Complex operands	For a complex operand, you enter the opening parenthesis (in the first column. You enter the closing parenthesis in the first column in a separate row following the operand entries of the following rows.	A string is rotated by a character each cycle:
		`LD stRotate`
		`RIGHT( stRotate`
		`LEN`
		`SUB 1`
		`)`
		`CONCAT( stRotate`
		`LEFT 1`
		`)`
		`ST stRotate`

Application	Description	Examples

Function block call, program call	Column 1: Operator `CAL` or `CALC`
	Column 2: Name of the function block instance or the program and opening parenthesis `(`. If no parameters follow, the closing parenthesis `)` is entered here.
	rows following that:
	Column 1: parameter name followed by `:=` for input parameter or `=>` for output parameter
	Column 2: parameter value followed by a comma `,` if further parameters follow. The closing parenthesis `)` is input after the last parameter.
	As a limitation according to the IEC standard, complex expressions cannot be used here. You must assign such constructs to the function block or the program before the call.
		`CAL POUToCall(`
		`iCounter:= 1,`
		`iDecrement:= 1000,`
		`wError=> wResult)`
		`LD POUToCall.bError`
		`ST bErr`

Function Call	Row 1: Column 1: `LD`
	Column 2: input variable
	Row 2: Column 1: Function name Column 2: further input parameters separated by commas.
	CODESYS writes the return value into the accumulator.
	Row 3: Column 1: `ST` Column 2: variable into which the return value is written
		`LD X7`
		`GeomAverage 25`
		`ST Ave`

Action call	Like function block call or program call. The action name is appended to the name of the FB instance or the program.	`CAL POU1.ResetAction`

Application	Description	Examples
Jump	Column 1: operator `JMP` or `JMPC`. Column 2: Name of the jump label of the destination network. In the case of an unconditional jump, the preceding instruction sequence must end with one of the following commands: `ST`, `STN`, `S`, `R`, `CAL`, `RET`, `JMP` In the case of a conditional jump the execution of the jump depends on the loaded value.	`LD BVar1` `JMPC Label1`

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *“Instruction list (IL)” on page 236*
- *Chapter 1.4.1.8.3.1.3 “Programming in instruction list (IL)” on page 240*

Elements

FBD/LD/IL element 'Network'

Symbol 

A network is the base unit of an FBD or LD program. In the FBD/LD/IL editor, the networks are arranged in a list. Each network is provided with a sequential network number on the left side and can include: logical and arithmetic expressions, program/function/function block calls, jumps, or return statements.

An IL program consists of at least one network. This network can include all IL statements of the program.

You can provide each network with a title, comment, or label. In the CODESYS options (category *“FBD, LD, and IL”*, you can define whether network title, comment, and separator between individual networks are displayed in the editor.

Click the first line of the network to enter a network title. Click the second line of the network to enter a network comment.

See also

- *Chapter 1.4.1.20.4.13.9 “Dialog 'Options' - 'FBD, LD, and IL'” on page 1192*
- *Chapter 1.4.1.20.3.13.1 “Command 'Insert Network'” on page 1104*

FBD/LD/IL element 'Box'

Symbol: 

A box and its call can represent additional functions, for example IEC function blocks, IEC functions, library function blocks, operators.

A box can have any number of inputs and outputs.

If the box also provides an image file, the box icon is displayed inside the box. The requirement is that the option *“Show box symbol”* is activated in the CODESYS options, category *“FBD, LD and IL”*.

If you have changed the box interfaces, you can update the box parameters with the command *“FBD/LD/IL ➔ Update Parameters”* without having to re-insert the box.

See also

- *Chapter 1.4.1.20.3.13.5 “Command 'Insert Box'” on page 1105*
- *Chapter 1.4.1.20.4.13.9 “Dialog 'Options' - 'FBD, LD, and IL'” on page 1192*
- *Chapter 1.4.1.20.3.13.38 “Command 'Update Parameters'” on page 1114*

FBD/LD/IL element 'Assignment'

Symbol:

The FBD editor shows a newly inserted assignment as a line with 3 question marks after it. The LD editor shows a newly inserted assignment as a coil with 3 question marks located above it.

After insertion you can replace the placeholder `???` by the name of the variable to which the signal coming from the left is to be assigned. The input assistant is available to you for this.

In IL an assignment is programmed via the operators `LD` and `ST`.

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.4 “Command 'Insert Assignment'” on page 1105*

FBD/LD/IL element 'Box with EN/ENO'

Symbol: 

The element is available only in the FBD and LD editors.

The box generally corresponds to the FBD/LD/IL element *“Box”*; however, this box additionally contains an EN input and an ENO output. EN and ENO have the data type `BOOL`.

Function of the `EN` input and `ENO` output: if the input `EN` has the value `FALSE` at the time of the calling the box, the operations defined in the box are not executed. Otherwise, i.e. if `EN` has the value `TRUE`, these operations are executed. The ENO output has the same value as the EN input.

See also

- *Chapter 1.4.1.20.3.13.6 “Command 'Insert Box with EN/ENO'” on page 1106*
- *Chapter 1.4.1.19.1.5.4.2 “FBD/LD/IL element 'Box'” on page 505*

FBD/LD/IL element 'Input'

Symbol: 

The maximum number of inputs depends on the type of box.

A newly added input is first marked with `???`. You can replace the string `???` by a variable or a constant.

See also

- *Chapter 1.4.1.20.3.13.13 “Command 'Insert Input'” on page 1107*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

FBD/LD/IL element 'Label'

The label is an optional identifier for a network in FBD and LD, which you can specify as a destination for a jump.

If you insert a jump label in a network, it will be added as an editable field *“Label:”* in the network.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.11 “Command 'Insert Label'” on page 1107*

FBD/LD/IL element 'Jump'

Symbol 

In FBD or LD a jump is inserted either directly before an input, directly after an output or at the end of the network, depending on the current cursor position.

You enter a jump label as the jump destination directly behind the jump element.

In IL you program a jump with the instruction `JMP`.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.10 “Command 'Insert Jump'” on page 1107*
- *Chapter 1.4.1.19.1.5.4.6 “FBD/LD/IL element 'Label'” on page 506*

FBD/LD/IL element 'Return'

This element immediately interrupts the execution of the box if the input of the `RETURN` element goes `TRUE`.

In an FBD or LD network you can place the Return instruction parallel to or after the preceding elements.

In IL the `RET` instruction is available to you for this purpose.

See also

- *Chapter 1.4.1.20.3.13.12 “Command 'Insert Return'” on page 1107*
- *Chapter 1.4.1.19.1.5.3 “Modifiers and operators in IL” on page 500*

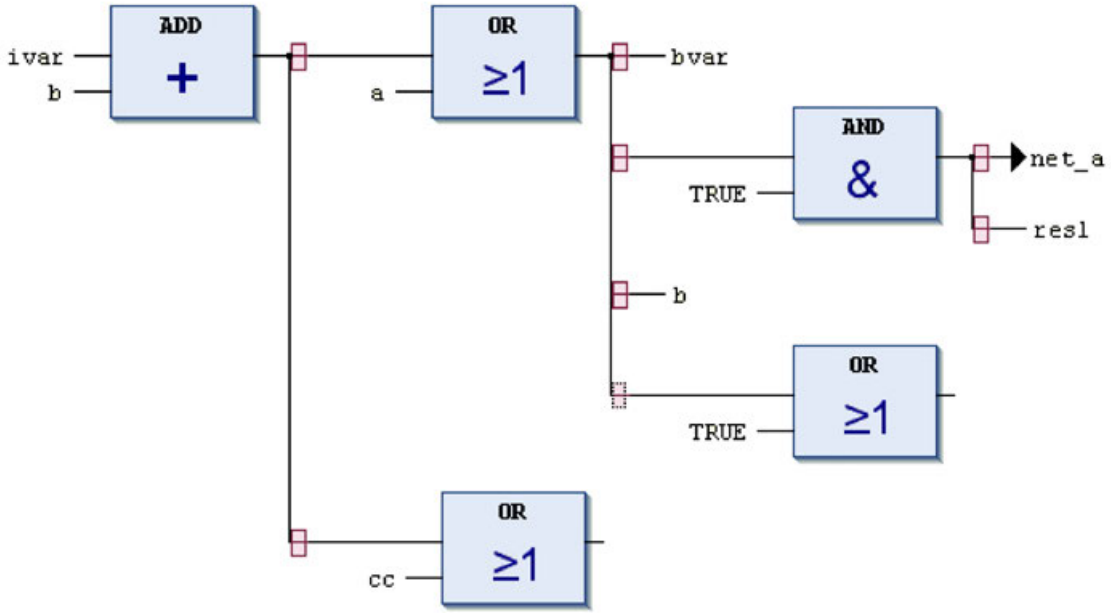
FBD/LD/IL element 'Branch'

Symbol: 

The element is available in the LD and FBD editor and represents an open line branch. A line branch splits the processing line from the current cursor position onwards into 2 subnetworks, which are executed in succession from top to bottom. You can branch each subnetwork further, as a result of which multiple branches are created within a network.

Each subnetwork is given a marker symbol (rectangle) at the branch point, which you can select in order to execute further commands.

> *The commands “Copy”, “Cut” and “Paste” are not available for subnetworks.*



In order to delete a subnetwork, you must first delete all elements of the network and then the marker symbol of the subnetwork.

See also

- *Chapter 1.4.1.20.3.13.33 “Command 'Insert Branch'” on page 1113*
- *Chapter 1.4.1.20.3.13.34 “Command 'Insert Branch Above'” on page 1113*
- *Chapter 1.4.1.20.3.13.35 “Command 'Insert Branch Below'” on page 1113*

FBD/LD/IL element 'Execute'

Symbol:

The element is a box that enables you to directly enter ST code in the FBD and LD editors.

You can drag the *“Execute”* element with the mouse from the *“Tools”* view into the implementation part of your POU. If you click on *“Enter ST code here...”*, an input field opens where you can input multiple-line ST code.

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

LD element 'Contact'

Symbol: , in the editor

The element is available only in the LD editor.

A contact passes on the signal `TRUE` (ON) or `FALSE` (OFF) from left to right until the signal finally reaches a coil in the right-hand part of the network. For this purpose a boolean variable containing the signal is assigned to the contact. To do this, replace the placeholder `???` above the contact with the name of a boolean variable.

You can arrange several contacts both in series and in parallel. In the case of two parallel contacts, only one needs to obtain the value `TRUE` in order for ON to be passed on to the right. If contacts are connected in series, all of them must obtain the value `TRUE` in order for ON to be passed on to the right by the last contact in the series. Hence, you can program electrical parallel and series connections with LD.

A negated contact forwards the signal `TRUE` if the variable value is `FALSE`. You can negate an inserted contact with the help of the command *"FBD/LD/IL ➔ Negation"* or insert a negated contact from the *"Tools"* view.

If you place the mouse pointer on a contact with the left mouse button pressed and with a network selected, the button *"Convert to coil"* appears in the network. If you now move the mouse pointer onto this button, still with the mouse button pressed, and then release the mouse button over this button, CODESYS converts the contact into a coil.

See also

- *Chapter 1.4.1.20.3.13.17 "Command 'Insert Contact'" on page 1108*
- *Chapter 1.4.1.20.3.13.22 "Command 'Insert Negated Contact'" on page 1110*
- *Chapter 1.4.1.20.3.13.18 "Command 'Insert Contact (Right)'" on page 1109*
- *Chapter 1.4.1.20.3.13.20 "Command 'Insert Contact in Parallel (Above)'" on page 1109*
- *Chapter 1.4.1.20.3.13.19 "Command 'Insert Contact in Parallel (Below)'" on page 1109*
- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*

LD element 'Coil'

Symbol: , in the editor

The element is available only in the LD editor.

A coil adopts the value supplied from the left and saves it in the boolean variable assigned to the coil. Its input can have the value `TRUE` (ON) or `FALSE` (OFF).

Several coils in a network can only be arranged in parallel.

In a negated coil the negated value of the incoming signal is stored in the boolean variable that is assigned to the coil.

Set coil, Reset coil

Symbol: , , in the editor: ,

Set coil: If the value `TRUE` arrives at a set coil, the coil retains the value `TRUE`. As long as the application is running, the value can no longer be overwritten here.

Reset coil: If the value `TRUE` arrives at a reset coil, the coil retains the value `FALSE`. As long as the application is running, the value can no longer be overwritten here.

You can define an inserted coil as a set or reset coil with the help of the command *"FBD/LD/IL ➔ Set/Reset"* or insert it as an element *"Set Coil"* and *"Reset Coil"* from the *"Tools"* view.

See also

- *Chapter 1.4.1.20.3.13.14 "Command 'Insert Coil'" on page 1108*
- *Chapter 1.4.1.20.3.13.16 "Command 'Insert Reset Coil'" on page 1108*
- *Chapter 1.4.1.20.3.13.29 "Command 'Negation'" on page 1112*
- *Chapter 1.4.1.20.3.13.31 "Command 'Set/Reset'" on page 1112*

LD element 'Branch Start/End'

Symbol:

The element serves the closed line branch.

See also

- *Chapter 1.4.1.19.1.5.4.14 “Closed branch” on page 509*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.36 “Command 'Set Branch Start Point'” on page 1113*
- *Chapter 1.4.1.20.3.13.37 “Command 'Set Branch End Point'” on page 1114*

Closed branch

A closed branch is available in LD only, and it contains a starting point and an end point. It is used for implementing parallel analyses of logical elements.

Inserting a closed branch

- Command *“FBD/LD/IL ➔ Insert Contact Parallel (Below) ”*
- Command *“FBD/LD/IL ➔ Insert Contact Parallel (Above) ”*
- Command *“FBD/LD/IL ➔ Set Branch Start/End Point”*

Closed branch at a contact

When you select one or more contacts and then execute the command *“Insert Contact in Parallel”*, a parallel branch is added with a single vertical line. For this kind of branching, the signal flow passes through both branches. This is an OR construct of both branches.

Closed branch at a block, OR evaluation, or short-circuit evaluation

New: When you select a box and execute the command *“Insert Contact in Parallel”*, a parallel branch is inserted with a double vertical line. This indicates that a short-circuit evaluation (SCE) is implemented. SCE allows for the execution of a function block with a Boolean output to be bypassed if a specific condition is `TRUE`. The condition can be displayed in the LD editor as a branch connected parallel to the function block branch. The short circuit condition is defined by one or more contacts in this branch that are interconnected parallel or sequentially.

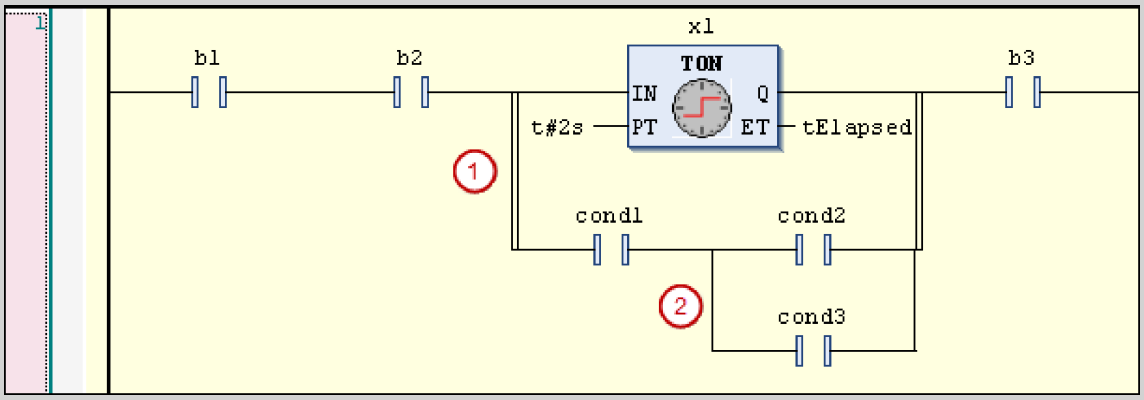
Functional principle:

The branches that do not include the function block are processed first. If CODESYS detects the value `TRUE` for one of these branches, then the function block is not called in the parallel branch. In this case, the value at the input of the function block is sent directly to the output. If CODESYS determines `FALSE` for the SCE condition, then the box will be called and the Boolean result of its processing is passed on. If all branches contain function blocks, they are analyzed from top to bottom and their outputs are logically ORed. If there are no branches with function blocks, normal OR operations are performed.

Example

The function block instance `x1` (TON) has a Boolean input and a Boolean output. The execution of `x1` is skipped if `TRUE` is determined for the condition in the parallel line branch. The condition value results from the OR and AND operations that connect contacts `cond1`, `cond2` and `cond3`.

`x1` is executed if the condition value from the connection of the contacts `cond1`, `cond2` and `cond3` is FALSE.



(1) Indicates from the double vertical connections that it is a construct subject to an SCE.

(2) Indicates from the single vertical connections that it is an OR construct.

The given LD example is shown below as ST code. `P_IN` and `P_OUT` are the Boolean values at the input (split point) and output (reunification point) of the parallel line branch.

```
P_IN := b1 AND b2;

IF ((P_IN AND cond1) AND (cond2 OR cond3)) THEN
   P_OUT := P_IN;
ELSE
   x1(IN := P_IN, PT := {p 10}t#2s);
   tElapsed := x1.ET;
   P_OUT := x1.Q;
END_IF
bRes := P_OUT AND b3;
```

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.36 “Command 'Set Branch Start Point'” on page 1113*
- *Chapter 1.4.1.20.3.13.37 “Command 'Set Branch End Point'” on page 1114*
- *Chapter 1.4.1.20.3.13.20 “Command 'Insert Contact in Parallel (Above)'” on page 1109*
- *Chapter 1.4.1.20.3.13.19 “Command 'Insert Contact in Parallel (Below)'” on page 1109*
- *Chapter 1.4.1.20.3.13.21 “Command 'Toggle Parallel Mode'” on page 1110*

Continuous Function Chart (CFC) and Page-Oriented CFC

From an external point of view, a Function Block Diagrams consists of inputs and outputs, with data being processed between them. From an internal point of view, a Function Block Diagrams consists of POUs and their connections which represent data (signals) and act as assignment operators in ST. The overall behavior is composed of the behavior of the inserted POUs which call other POUs or library POUs.

Code in the *"Continuous Function Chart (CFC)"* implementation language mainly illustrates the data flow through the system. Therefore, a continuous function chart is also referred to as a "signal flow chart".

In the page-oriented CFC editor, you can wire POUs to each other and create well-structured Function Block Diagrams distributed over multiple pages. The page-oriented editor behaves like the CFC editor, but provides the following functionality:

The page-oriented editor behaves like the CFC editor, but provides additional functionality.

The editor supports you with the following functions:

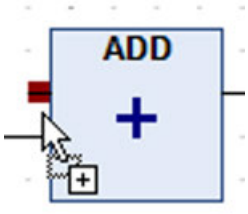
- Creating pages
- Setting the page size
- Copying and inserting pages in the page navigator
- Copying the implementation of a POU in the CFC implementation language and inserting into a page
- Well-structured and space-saving arranging of inputs, outputs, and connection marks in the border areas
- Connection over pages with connection marks

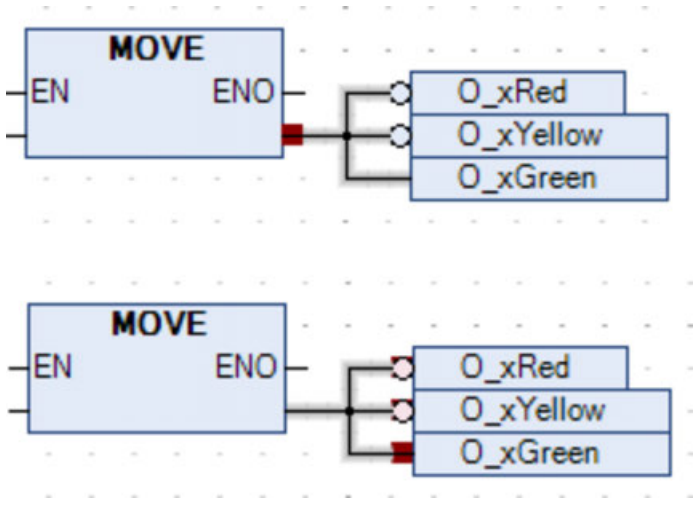
CFC Editor

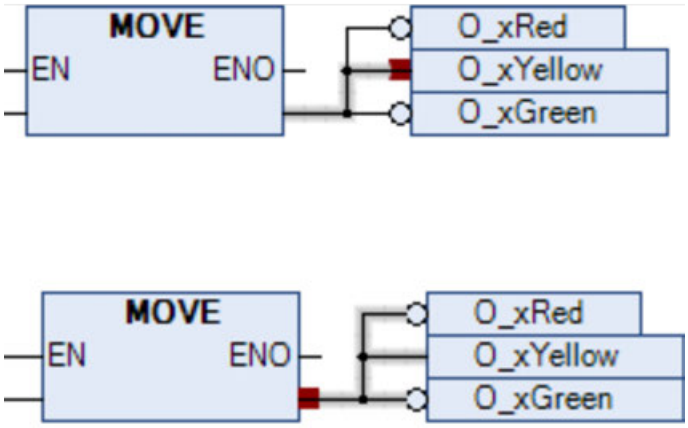
Configuring the editor

You can configure the appearance, behavior, and printing for the entire project in the CODESYS options in the *"CFC Editor"* category. For example, on the *"View"* tab, you can configure the color of the connecting lines depending on the data type.

Editing

Cursor symbol:	Requirement: *"Pointer"* is selected in the *"Toolbox"* view. The symbol indicates that you can edit in the editor. Select elements or connections to move them or to execute commands.
Cursor symbol: +	Requirement: An element is selected in the *"Toolbox"* view. Clicking in the editor inserts the selected element. You can also drag an element to the editor.
Dragging a function block instance from the declaration to the editor	Requirement: A line is selected in the declaration of the CFC. The instance is inserted as a box with name, type, and all pins.
Dragging a variable from the declaration to a box pin to the editor	The variable is inserted as an input or output with a connection to the box pin in focus. Hint: The cursor indicates when your focused location is valid for a variable: 

Dragging a variable from the declaration part to the editor	Requirement: The respective element is selected in the declaration. ● Function block instance: A POU with the corresponding data type is created. ● Declaration of `VAR_INPUT` or `CONSTANT`: An input element is inserted. ● Declaration of `VAR_OUTPUT`: An output element is inserted. ● Declaration of `VAR`, `VAR_GLOBAL`: A window opens at the insert location, where you can select whether an input element or output element should be inserted. When a variable is dragged from the declaration part to an existing replaceable element, the existing element is replaced.
Dragging a function block or POU to the editor from the *"Devices"* view, *"POUs"* view, or from the Library Manager.	A box element with the corresponding type is inserted. ● When a box is dragged to an existing connecting line and both the input and output of the box are compatible with the data type of the line, the box is inserted in the line. Here its first matching input and output are connected to the elements that were previously connected by the connecting line. ● When a box is dragged to an existing box, the existing box is replaced.
Resorting the order of inputs and outputs within a function block by means of drag&drop	Requirement: The text field of the input or output, which should be resorted to another location, is selected.
[Ctrl] + click in the programing area	Requirement: An element is selected in the *"Toolbox"* view. As long as you hold down the *[Ctrl]* key, a selected element is created each time you click in the programming area.
[Ctrl]+[Right Arrow]	Requirement: In the CFC program, **exactly** one output pin is selected for an element. The selection is moved so that the input pin at the end of the connecting line is selected. In the case of multiple pins, all are selected. 

[Ctrl]+[Left Arrow]	Requirement: In the CFC program, **exactly** one input pin is selected for an element. The selection is moved so that the output pin at the beginning of the connecting line is selected. In the case of multiple pins, all are selected. Example: 

See also

- *Chapter 1.4.1.19.1.2 “Common functions in graphical editors” on page 462*

Connecting

You can insert connecting lines between element connections. Connecting lines are inserted by means of auto-routing so that connecting lines are automatically optimal and as short as possible. The connecting lines are checked for collisions.

Dragging a pin to another	A connecting line is inserted between the two element pins.
Dragging a pin to a function block	Dropping can be done on a pin or on the text field of a pin. In the case of extendable operators (example: ADD), dropping can also be done within the box. The following behavior applies for this: If there are still unconnected input pins, then the top free pin is connected. If there are no more unconnected input pins, then a new pin is automatically inserted below.
Command *“Connect Selected Pins”*	Requirement: Multiple pins are selected. The pins are marked in red.
Move an inserted element so that it touches the pin of another element.	Requirement: The *“Enable AutoConnect”* option is selected. The touching pins are connected automatically.
	The connection icon is located in the upper right corner of the editor. A green icon indicates collision-free connections. A red icon indicates collisions. Clicking the icon opens a menu with commands for collision processing, for example the *“Show Next Collision”* command.
	Requirement: A connection is selected and the *“Connection Mark”* command is executed. Instead of a long connecting line, a connection is represented by connection marks.

See also

- *Chapter 1.4.1.20.3.12.22 “Command 'Show Next Collision'” on page 1098*

Commands when editing

See also

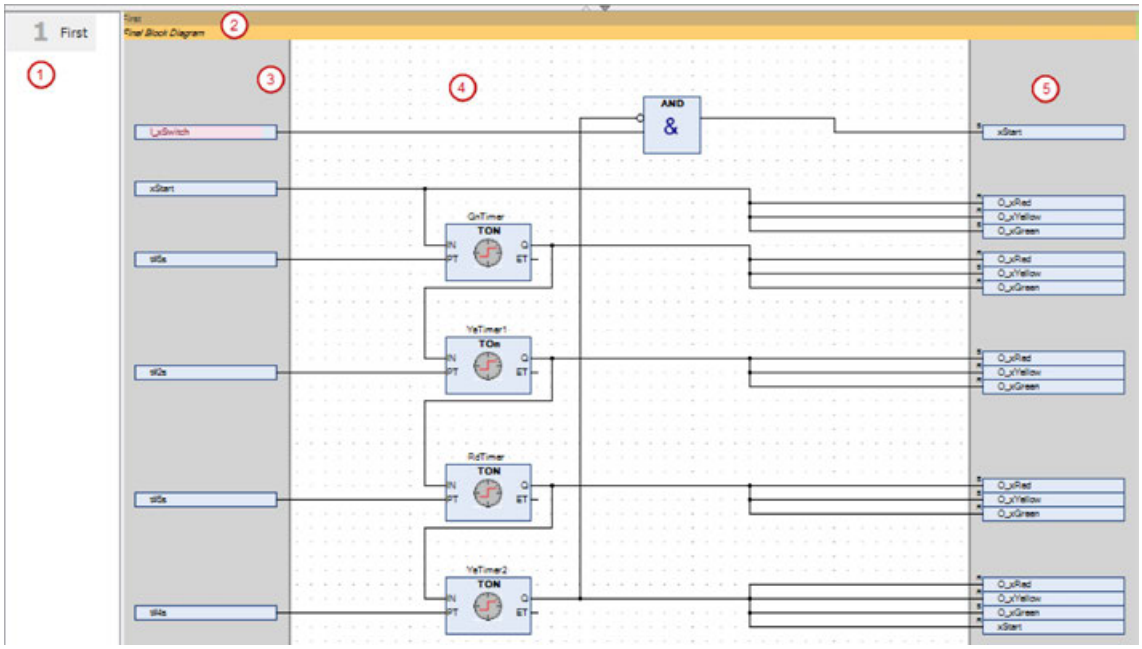
- *Chapter 1.4.1.20.3.12 “Menu 'CFC'” on page 1089*

See also

- *Chapter 1.4.1.19.1.2 “Common functions in graphical editors” on page 462*
- *Chapter 1.4.1.8.3.2.2 “ Programming in the CFC editor” on page 246*
- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.3.12 “Menu 'CFC'” on page 1089*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*

CFC editor, page-oriented

POUs generated in the “Continuous Function Chart (CFC) - page-oriented” cannot be converted into “Continuous Function Chart (CFC)” POUs or back.



- (1) Page navigator
- (2) Page header with name and description
- (3) Left border area reserved for inputs and sink connection marks
- (4) Program area
- (5) Right border area reserved for outputs and source connection marks

Editing

You can drag a *“Page”* element from the *“ToolBox”* view to the page navigation. Then an additional page is inserted.

You can select existing pages in the page navigation and duplicated them by clicking *“Edit ➔ Copy”* and *“Edit ➔ Paste”*.

The size of the page is changed by means of the *“Edit Page Size”* command.

Connections over multiple pages are established by means of the *“Connection Mark - Source”* and *“Connection Mark - Sink”* elements. When you drag a connecting line from an input pin or an output pin to the border area, a new connection mark is created automatically. The advantage is that the "List components" input assistance provides all previously defined connection mark sources.

If you have selected an element in the editor, then you can use the arrow keys to move the selection from one element to the next to navigate through the circuit. If you then select a connection mark and press another arrow key, even the corresponding connection mark of the next/previous page will be selected.

You can transfer networks from a CFC POU to the program area of a page-oriented CFC by clicking *“Edit ➔ Copy”* and *“Edit ➔ Paste”* (from the clipboard). You can also use drag&drop.

Execution order

The execution order is determined automatically according to the order of the pages as they are sorted in the page navigator of the editor. Within a page, a page-oriented CFC object behaves like a CFC object. Therefore, you can switch between *“Auto Data Flow Mode”* and *“Explicit Execution Order Mode”*.

Additional commands in *“CFC page-oriented”*

See also

- *Chapter 1.4.1.20.3.12.2 “Command 'Edit Page Size'” on page 1090*
- *Chapter 1.4.1.20.3.12.1 “Command 'Edit Worksheet'” on page 1089*

See also

- *Chapter 1.4.1.19.1.2 “Common functions in graphical editors” on page 462*
- *Chapter 1.4.1.8.3.2.2 “ Programming in the CFC editor” on page 246*
- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.3.12 “Menu 'CFC'” on page 1089*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*

Keyboard Shortcuts in the CFC Editors

Keyboard shortcuts in the CFC editor and page-oriented CFC editor

Keyboard shortcuts	**Command**
[Ctrl]+[Shift]+[A]	Select All
Insert elements:	
[Ctrl]+[B]	Insert Box The *“Input Assistant”* dialog opens in order to select the box.
[Ctrl]+[Shift]+[B]	Insert Empty Box
[Ctrl]+[Shift]+[E]	Insert Box with EN/ENO The *“Input Assistant”* dialog opens in order to select the box.
[Ctrl]+[Q]	Insert Input Inserts an input element
[Ctrl]+[A]	Insert Output Inserts an output element
[Ctrl]+[L]	Insert Jump
Edit already inserted elements:	
[Ctrl]+[N]	Negate
[Ctrl]+[M]	Toggle between Set, Reset, REF, and None
[Ctrl]+[U]	Reset Pins

After inserting an element, the inserted element is selected in the editor.

See also

- *Chapter 1.4.1.19.1.6.5.5 “CFC Element 'Box'” on page 523*
- *Chapter 1.4.1.19.1.6.5.3 “CFC Element 'Input'” on page 522*
- *Chapter 1.4.1.19.1.6.5.4 “CFC Element 'Output'” on page 522*
- *Chapter 1.4.1.19.1.6.5.6 “CFC element 'Jump'” on page 523*

- *Chapter 1.4.1.20.3.12.3 “Command 'Negate'” on page 1090*
- *Chapter 1.4.1.20.3.12.7 “Command 'S (Set)'” on page 1091*
- *Chapter 1.4.1.20.3.12.6 “Command 'R (Reset)'” on page 1091*
- *Chapter 1.4.1.20.3.12.8 “Command 'REF= (Reference Assignment)'” on page 1091*
- *Chapter 1.4.1.20.3.12.5 “Command 'None'” on page 1091*
- *Chapter 1.4.1.20.3.12.24 “Command 'Reset Pins'” on page 1098*

CFC Editor in Online Mode

In online mode, you can monitor and change variable values of the controller. In addition, debugging features are provided such as breakpoints and stepping.

Monitoring

As usual, you can monitor values in the declaration part as well as in the implementation part (with inline monitoring).

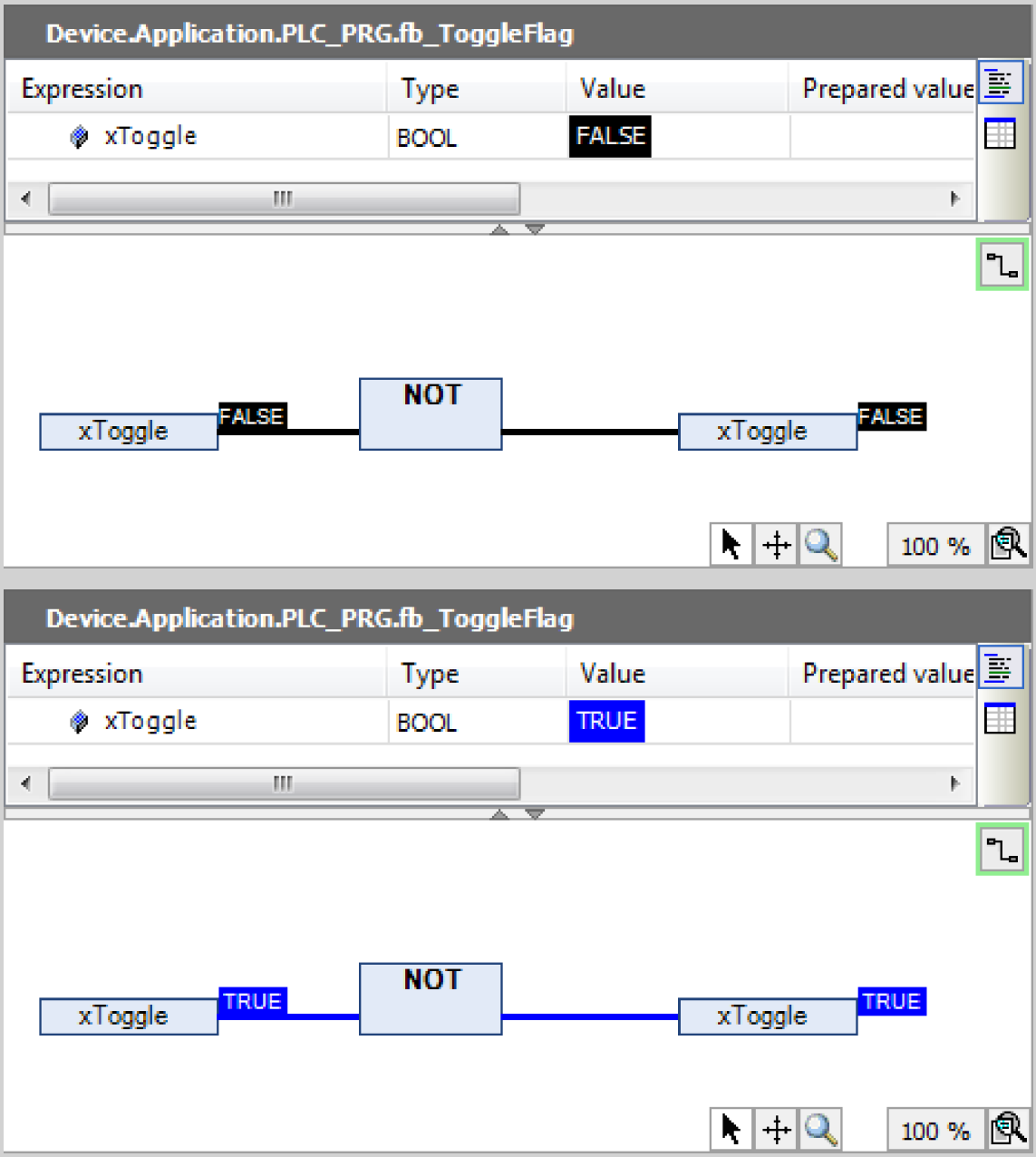
Inline monitoring of a function block is possible only when an instance of the function block is open. No values are displayed in the basic implementation view.

Monitoring a Boolean variable

The connections between Boolean variables are displayed in color according to their actual value: `TRUE` in blue and `FALSE` in black. The element pins are decorated with the actual value.

Example

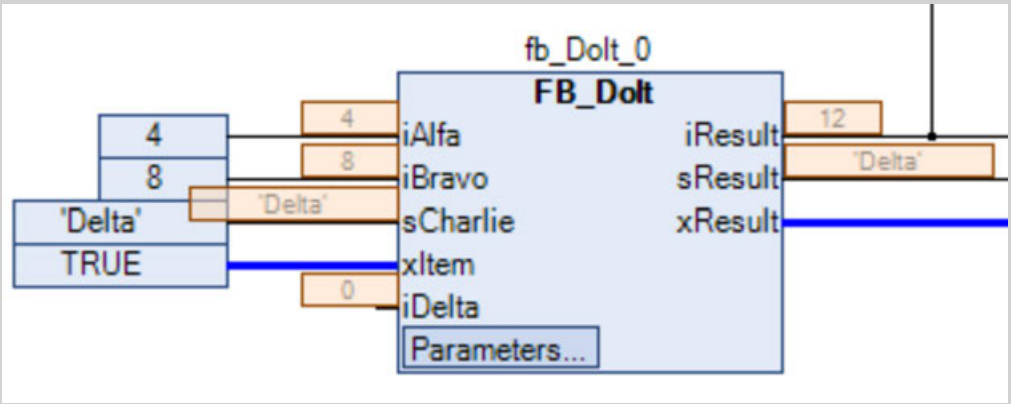
An application contains a CFC POU. An internal Boolean variable is switched there. With each cycle, the variable `iToggle` switches its state from `TRUE` to `FALSE`.



Monitoring a scalar variable

In the case of scalar variables, the element pins are decorated with the actual values.

Example



Forcing and writing of variables

In online mode in the declaration editor, you can prepare a value for forcing or writing a monitored variable.

When you select the *"Prepare values in implementation part"* check box in the *"CFC Editor"* category of the CODESYS options, you can also prepare values in the implementation part.

To do this, open the *"Prepare Value"* dialog by double-clicking an element or the monitoring box next to an element. No dialog appears for Boolean variables. However, with each mouse click on the value displayed next to the variable, the values `TRUE` and `FALSE` are toggled.

Prepared values are displayed in angle brackets. After executing a write or a force, a red "F" is shown in the monitoring box.

Changing of constant input parameters of function block instances

You can write input parameters of function block instances of type `VAR_INPUT CONSTANT` in online mode and modify the parameters in this way. After logging out, you save these parameters by clicking *"Save Prepared Parameters to Project"*.

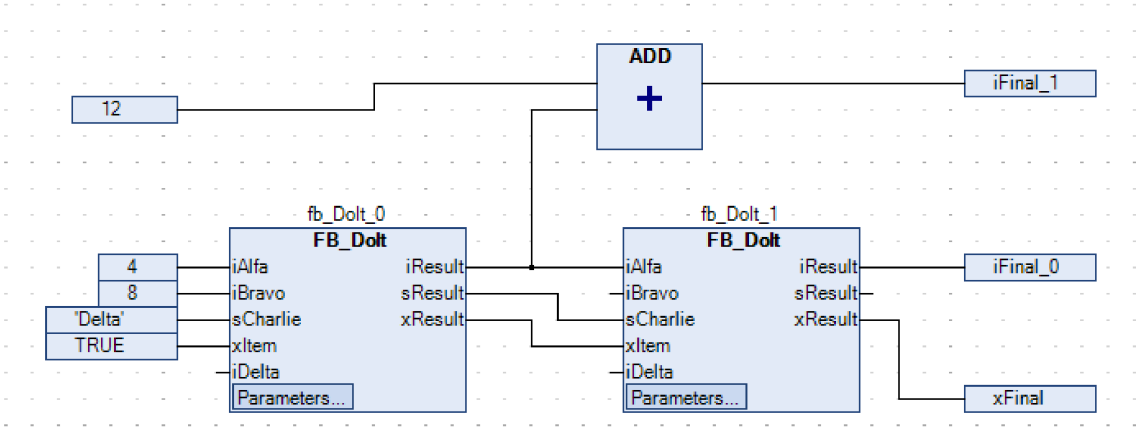
☑ Requirement: A CFC editor is active. An instantiated function block has `VAR_INPUT CONSTANT` variables in its declaration.

1. In the editor, open the POU by calling the function block instance.

   ⇨

```
FUNCTION_BLOCK FB_DoIt
VAR_INPUT
    iAlfa : INT;
    iBravo: INT;
    sCharlie : STRING := 'Charlie';
    xItem : BOOL;
    iDelta : INT;
END_VAR
VAR_INPUT CONSTANT
    MAXIMUM : INT := 12;
END_VAR
VAR_OUTPUT
    iResult : INT;
    sResult : STRING;
    xResult : BOOL;
END_VAR
```

   The declaration of `FB_DOIt` has been supplemented by the constant `MAXIMUM`.



   The graphical representation of the function block instances contains the *"Parameters"* button.

2. Login to the controller.
3. Click the *"Parameters"* button of the function block instance.

   ⇨ The *"Edit Parameters"* dialog opens.

4. Click the *"Value"* column in an inline monitoring field of a parameter.

   ⇨ The *"Prepare Value"* dialog opens.

5. Type `20` in the *"Prepare a new value for the next write or force operation"* field.
6. Click *"OK"* to confirm the entry.

   ⇨ The prepared value is shown in angle brackets next to the current value (for example, `<20>`).

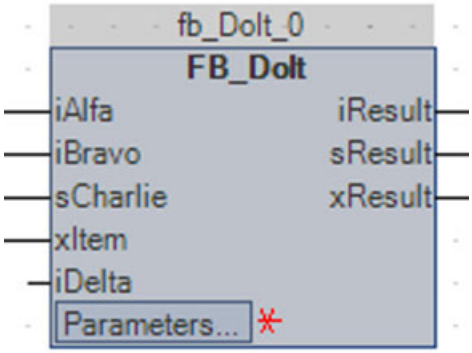
Parameter	Type	Value	Initial value	Min	Max	Unit	Description
MAXIMUM	INT	12 <20>	12				

7. Click *“Debug ➔ Write Values”*.

   ⇨ The prepared value is written. The parameter is changed and displayed in the project in brackets after the value.

Parameter	Type	Value	Initial value	Min	Max	Unit	Description
MAXIMUM	INT	[20] 20	12				

The difference between both values is shown by a red cross next to the parameter field of the function block instance.



8. Click *“Edit Parameters”* to close the dialog. Logout.
9. Click *“CFC ➔ Save Prepared Parameters to Project”*.

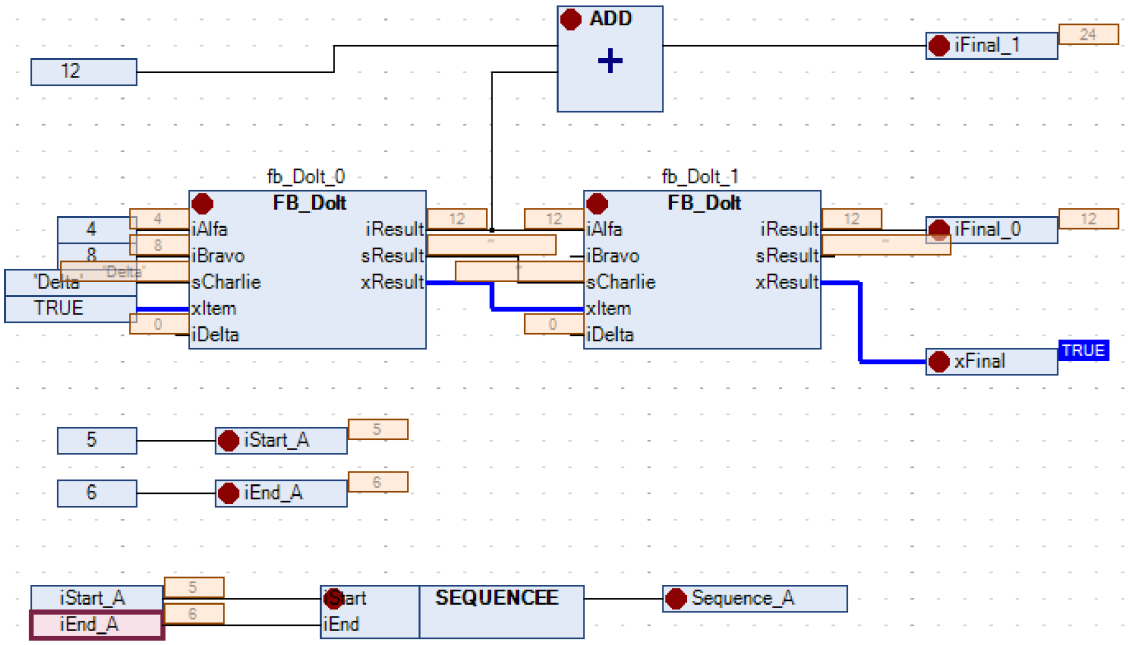
   ⇨ The change parameter values are saved to the project. The asterisk next to the parameter field disappears.

Breakpoint locations

Possible position of a breakpoint

- Element *“Output”*
  Variables are described.
- Element *“Box”*
  POUs are called.
- Element *“RETURN”*
  The program flow is branched.
- Element *“Selector”*
  Structure elements are described.

Click *“Debug ➔ Toggle Breakpoint”* to set a new breakpoint or delete an existing breakpoint. A red circle in the block diagram represents an active breakpoint.



> **NOTICE!**
>
> A breakpoint is set automatically in all methods that can be called.
>
> Therefore, if a method is called that is defined over an interface, then breakpoints are set in all methods of function blocks that implement this interface. This also applies to all inherited function blocks that define methods.

Stepping into a POU

You can process a POU in steps in debug mode. A called POU is supplemented internally by a `RETURN` at the beginning before the element with the number `0` and at the end after the last element. When stepping, these are started automatically.

Commands in online mode

See also

- *Chapter 1.4.1.20.3.12.35 “Command 'Force Function Block Input'” on page 1101*
- *Chapter 1.4.1.20.3.12.34 “Command 'Prepare Box for Forcing'” on page 1101*
- *Chapter 1.4.1.20.3.12.18 “Command 'Edit Parameters'” on page 1096*
- *Chapter 1.4.1.20.3.12.19 “Command 'Save Prepared Parameters to Project'” on page 1097*

See also

- *Chapter 1.4.1.12.1 “Monitoring of Values” on page 409*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *“Forcing a function block input in CFC” on page 403*
- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*
- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*

Elements

CFC element 'Page'

Symbol: 

The element inserts a new page into the editor. It is available only in the page-oriented CFC editor. The number of the page is automatically assigned in accordance with its position. You can enter the name and the description of the page into the orange header. The page size is adapted with the *“Edit Page Size”* command.

See also

- *Chapter 1.4.1.20.3.12.2 “Command 'Edit Page Size'” on page 1090*

CFC element 'Control Point'

Symbol: 

Use a control point in order to fix points of a connection before you adapt the line routing. To do this, drag the element to the desired position on a connecting line. Connecting lines with control points are no longer routed automatically.

See also

- *Chapter 1.4.1.8.3.2.2 “ Programming in the CFC editor” on page 246*
- *Chapter 1.4.1.20.3.12.30 “Command 'Create Control Point'” on page 1100*
- *Chapter 1.4.1.20.3.12.29 “Command 'Remove Control Point'” on page 1099*

CFC Element 'Input'

Symbol:

Keyboard shortcuts for inserting the element: *[Ctrl]+[Q]*

By default, CODESYS inserts an input element with the text "???". You can edit this input field directly by clicking it and typing in a constant value or a variable name. Alternatively, you could click  to open the Input Assistant to select a variable.

CFC Element 'Output'

Symbol:

Keyboard shortcuts for inserting the element: *[Ctrl]+[A]*

By default, CODESYS inserts an output element with the text "???". You can edit this input field directly by clicking it and typing in a constant value or a variable name. Alternatively, you could click ... to open the Input Assistant to select a variable.

CFC Element 'Box'

Symbol:

Keyboard shortcuts for inserting the element

- *[Ctrl]+[B]*
- *[Ctrl]+[Shift]+[B]*: Empty box
- *[Ctrl]+[Shift]+[E]*: Box with EN/ENO

You use the element in order to insert an operator, a function, a function block, or a program. By default, CODESYS inserts the element with the name *“???”*. You can edit this field directly by clicking it and typing in a function block name. Alternatively, you could click ... to open the Input Assistant and select a function block

In the case of a function block, CODESYS also displays an input field (*“???”*) above the function block symbol. You have to replace this name with the name of the function block instance. If you instantiate a function block with constant input parameters, then the function block element displays the "Parameter..." field in the bottom left corner. You click on this field to edit the parameters.

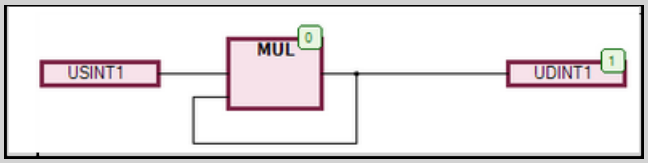
In order to replace an existing box, you replace only the currently inserted identifier with the new desired name. When you do this, note that CODESYS adapts the number of input and output pins according to the definition of the POU and that existing assignments may be deleted as a result.

NOTICE!

Because feedback is allowed in CFC, implicit variables with the data type of the input variable are created at the output of a box (in the example: `temp_USINT`). If the result of the operation of a function block is a value which exceeds the number range of the data type of the input variable, then the overflow is written to the implicit variable. The actual output variable gets the value of the implicit variable, thus the overflow and not the actual result of the operation (see example).

Example

Implicitly generated variables at the box output:



Implicitly generated code:

```
temp_USINT := USINT1 * temp_USINT;
UDINT1 := temp_USINT;
```

See also

- *Chapter 1.4.1.20.3.12.18 “Command 'Edit Parameters'” on page 1096*

CFC element 'Jump'

Symbol:

You use the element in order to define a position at which program execution is to continue. You must define this target position by a label. To do this, enter the name of the mark in the input field *“???”*. If you have already inserted the corresponding label, you can also select it via the input assistant (...).

See also

- *Chapter 1.4.1.19.1.6.5.7 “CFC element 'Label'” on page 524*

CFC element 'Label'

Symbol:

A label defines a position to which program execution jumps with the help of a jump element.

In online mode CODESYS automatically inserts a RETURN flag at the end of a CFC function block.

See also

- *Chapter 1.4.1.19.1.6.5.6 “CFC element 'Jump'” on page 523*

CFC element 'Return'

Symbol:

Use the element in order to exit the function block.

Please note that in online mode in the CFC editor a return element is automatically inserted before the first line and after the last element. In single-step execution CODESYS automatically jumps to the return element at the end before exiting the function block.

CFC element 'Composer'

Symbol:

The composer element is for handling structural components. The individual components of a structure are made available to you as an input. For this purpose you must name the composer element like the structure concerned (replace the *“???”*).

The composer element is the counterpart to the selector element.

See also

- *Chapter 1.4.1.19.1.6.5.10 “CFC element 'Selector'” on page 524*

CFC element 'Selector'

Symbol:

The selector element is for handling structural components. The individual components of a structure are made available to you as an output. For this purpose you must name the selector element like the structure concerned (replace the *“???”*)

The selector element is the counterpart to the composer element.

See also

- *Chapter 1.4.1.19.1.6.5.9 “CFC element 'Composer'” on page 524*

CFC element 'Comment'

Symbol:

With this element you input a comment in the CFC editor. Replace the placeholder text in the element by the comment text. A line break can be inserted with the aid of the shortcut *[Ctrl]* + *[Enter]*.

CFC element 'Connection Mark - Source/Sink'

Symbol: ,

You can use connection marks instead of a connecting line between elements. That helps you to display complex diagrams more clearly.

For a valid connection you must connect an element *“Connection Mark - Source”* with the output of an element and an element *“Connection Mark - Sink”* with the input of another element. Both marks must bear the same name. The names are not case-sensitive.

Notes on naming

- The standard name for connection marks is `C-<nr>`. `<nr>` is a sequential number starting with 1.
- You can rename the standard name. In doing so, you must make sure that the connection mark - source and connection mark - sink have the same name.
- If you change the name of the connection mark - source, the destination name is automatically renamed.
- If you change the name of the connection mark - sink, the source name is retained.

Observe the command “Connection Mark” for the automatic transformation of an existing connection.

See also

- *Chapter 1.4.1.20.3.12.31 “Command 'Connection Mark'” on page 1100*
- *Chapter 1.4.1.19.1.6.1 “CFC Editor” on page 511*

CFC element 'Input Pin'

Symbol: 

Depending on the type of function block you can add further inputs to an inserted function block element. To do this you must select the function block element and drag the function block input element onto the body of the function block.

Please note: You can drag an input or output connection to another position on the function block with the *[Ctrl]* key pressed.

See also

- *Chapter 1.4.1.19.1.6.5.14 “CFC element 'Output Pin'” on page 525*

CFC element 'Output Pin'

Symbol: 

Depending on the type of function block you can add further outputs to an inserted function block element. To do this you must select the function block element and drag the function block output element onto the body of the function block.

Please note: You can drag an input or output connection to another position on the function block with the *[Ctrl]* key pressed.

See also

- *Chapter 1.4.1.19.1.6.5.13 “CFC element 'Input Pin'” on page 525*

1.4.1.19.2 Variables

The scope of a variable defines how and where you can use a variable. You define the scope in the variable declaration.

Local variables - VAR

Local variables are declared between the keywords `VAR` and `END_VAR` in the declaration part of programming objects.

You have read-only access to local variables by using the instance path.

You can extend local variables with an attribute keyword.

Example

```
VAR
    iVar1 : INT;
END_VAR
```

See also

- ↳ *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

Input variables - VAR_INPUT

Input variables are used at the inputs of function blocks.

`VAR_INPUT` variables are declared between the keywords `VAR_INPUT` and `END_VAR` in the declaration part of programming objects.

You can extend input variables with an attribute keyword.

Example

```
VAR_INPUT
    iIn1 : INT; (* 1st input variable *)
END_VAR
```

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

Output variables - VAR_OUTPUT

Output variables are used at the outputs of function blocks.

`VAR_OUTPUT` variables are declared between the keywords `VAR_OUTPUT` and `END_VAR` in the declaration part of programming objects. CODESYS returns the values of this variable to the calling POU. There you can retrieve the values and continue using them.

You can extend output variables with an attribute keyword.

Example

```
VAR_OUPUT
    iOut1 : INT; (*1st output variable *)
END_VAR
```

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

Output variables in functions and methods

According to the IEC 61131-3 standard, functions and methods have additional outputs. You have to assign these additional outputs when calling the function, as shown below.

Example

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

Input/Output Variable (VAR_IN_OUT)

A `VAR_IN_OUT` variable is an input/output variable, which is part of a POU interface and serves as a formal pass-by-reference parameter.

Syntax declaration

```
<keyword> <POU name>
VAR_IN_OUT
    <variable name> : <data type> ( := <initialization value> )? ;
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

You can declare an input/output variable in the `VAR_IN_OUT` declaration section in the POUs `PRG`, `FUNCTION_BLOCK`, `METHOD`, or `FUNCTION`. As an option, a constant of the declared data type can be assigned as an initialization value. The `VAR_IN_OUT` variable can be read and written.

Usage

- **Call**: When a POU is called, the formal `VAR_IN_OUT` variable receives the actual variable (pass-by-reference variable) as the argument. At runtime, no copies are generated when parameters are passed. Instead, the formal variable receives a reference to the actual variable passed remotely. The referential variables contain a memory address internally as a value to the actual value (*pass as pointer*, *call-by reference*). It is not possible to specify a constant (literal) or a bit variable directly as an argument.
- **Read/write access within the POU**: If the variable is written to within the POU, then this affects the passed variable. When the POU is exited, any performed changes are retained. This means that a POU uses its `VAR_IN_OUT` variables just like the calling POU uses its variables. Read access is always permitted.

- **Read/write access remotely**: `VAR_IN_OUT` variables **cannot** be directly read or written remotely via `<function block instance name>.<variable name>`. This works only for `VAR_INPUT` and `VAR_OUTPUT` variables.
- **Passing string variables**: If a string variable is passed as an argument, then the actual variable and the formal variable should have the same length. Otherwise the passed string can be manipulated unintentionally. This problem does not occur in the case of `VAR_OUTPUT CONSTANT` parameters.
- **Passing bit variables**: A bit variable cannot be passed directly to a `VAR_IN_OUT` variable because it needs an intermediate variable.
- **Passing properties**: Not permitted.

If a string is passed as a variable or a constant to a formal `VAR_IN_OUT CONSTANT` variable, then the string is automatically passed completely. You do not have to check the string length.

See also

- Chapter *"Transfer variable `VAR_IN_OUT CONSTANT`" on page 530*

Example

Passing arrays

```
TYPE DUT_A :
STRUCT
    xA: BOOL;
    iB: INT;
END_STRUCT
END_TYPE

FUNCTION_BLOCK FB_SetArray
VAR_IN_OUT
    aData_A : ARRAY[0..1] OF DUT_A; // Formal variable
END_VAR
aData_A[0].xA := TRUE;
aData_A[0].iB := 100;

PROGRAM PLC_PRG
VAR
    fbSetA : FB_SetArray;
    aSpecialData : ARRAY[0..1] OF DUT_A; // Actual variable
END_VAR
fbSetA(aData_A := aSpecialData);
```

Expression	Type	Value	Prepared value	Address	Comment
fbSetA	FB_SetArray				
aSpecialData	ARRAY [0..1] OF DUT_A				Actual variable
aSpecialData[0]	DUT_A				
xA	BOOL	TRUE			
iB	INT	100			
aSpecialData[1]	DUT_A				
xA	BOOL	FALSE			
iB	INT	0			

Passing strings

```
{attribute 'qualified_only'}
VAR_GLOBAL
    g_sDEV_STATUS : STRING(25) := 'Device_A';
END_VAR

FUNCTION_BLOCK FB_SetStatus
VAR_IN_OUT
    sDeviceStatus : STRING(25); // Formal parameter
END_VAR
sDeviceStatus := CONCAT(sDeviceStatus, ' Activ');

PROGRAM PLC_PRG
VAR
    fbDoB : FB_SetStatus;
END_VAR
fbDoB(sDeviceStatus := GVL.g_sDEV_STATUS); //Call with actual
parameter
```

The variable `sDeviceStatus` is part of the POU interface of `FB_B`. When calling `fbDoB`, first a device name is assigned to the string and then the string is manipulated.

Passing bit variables

```
VAR_GLOBAL
    xBit0 AT %MX0.1 : BOOL;
    xTemp : BOOL;
END_VAR

FUNCTION_BLOCK FB_DoSomething
VAR_INPUT
```

```
    xIn : BOOL;
END_VAR
VAR_IN_OUT
    xInOut : BOOL;
END_VAR
IF xIn THEN
    xInOut := TRUE;
END_IF

PROGRAM PLC_PRG
VAR
    xIn : BOOL;
    DoSomething_1  : FB_DoSomething;
    DoSomething_2  : FB_DoSomething;
END_VAR

// The following line of code causes a compiler error:
// C0201: Typ 'BIT' is not equal to type 'BOOL' of VAR_IN_OUT
'xInOut'
DoSomething_1(xIn := xIn, xInOut := xBit0);

// Workaround
xTemp := xBit0;
DoSomething_2(xIn := xIn, xInOut := xTemp);
xBit0 := xTemp;
```

The program calls the function block instances `DoSomething_1` and `DoSomething_2`. As a result of the direct assignment of the bit variable `xBit0` to the `VAR_IN_OUT` input, a compiler error is generated when the `DoSomething_1` instance is called. In contrast, calling the `DoSomething_2` instance with the assignment of an intermediate variable is correct code.

Transfer variable VAR_IN_OUT CONSTANT

A `VAR_IN_OUT CONSTANT` variable serves as a constant pass-by-reference parameter, to which a `STRING` or `WSTRING` type variable or constant (literal) can be passed. The parameter can be read, but not written. Passing of properties is not permitted.

Syntax declaration

```
<keyword> <POU name>
VAR_IN_OUT CONSTANT
    <variable name> : <data type>; // formal parameter
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

`VAR_IN_OUT CONSTANT` variables are declared without assigning an initialization value.

Usage

- When calling the POU, a `STRING` or `WSTRING` constant variable or literal can be passed. Consequently, write access is not permitted.
- Passing parameters of a string constant: The string length of the constants can be any size, and the string length does not depend on the string length of the `VAR_IN_OUT CONSTANT` variables.

If the “Replace constants” option is selected in “Project ➔ Project Settings” in the “Compile Options” category, then passing the parameters of a constant with basic data type or a constant variable with basic data type generates a compiler error.

The variable is supported in compiler version >= 3.5.2.0.

Example

Passing parameters of string constants and string variables

```
FUNCTION funManipulate : BOOL
VAR_IN_OUT
    sReadWrite : STRING(16); (* Can be read or written here in POU
*)
    dwVarReadWrite : DWORD; (* Can be read or written here in POU *)
END_VAR
VAR_IN_OUT CONSTANT
    c_sReadOnly : STRING(16); (* Constant string variable can only
be read here in POU *)
END_VAR

sReadWrite := 'String_from_POU';
dwVarReadWrite := STRING_TO_DWORD(c_sReadOnly);

PROGRAM PRG_A
VAR
    sVarFits : STRING(16);
    sValFits : STRING(16) := '1234567890123456';
    dwVar: DWORD;
END_VAR

// The following line of code causes the compiler error C0417:
// C0417: VAR_IN_OUT parameter 'sReadWrite' needs a variable with
write access as input.
funManipulate(sReadWrite:='1234567890123456',
c_sReadOnly:='1234567890123456', dwVarReadWrite := dwVar);

//  Correct code
funManipulate(sReadWrite := sValFits, c_sReadOnly := '23',
dwVarReadWrite := dwVar);
funManipulate(sReadWrite := sVarFits, c_sReadOnly := sValFits,
dwVarReadWrite := dwVar);
```

In the code, strings are passed to the `funManipulate` function via different `VAR_IN_OUT` variables. When passing a string literal, a compiler error is output to a `VAR_IN_OUT` variable. When passing a constant variable to a `VAR_IN_OUT CONSTANT` variable, correct code is generated even for passing string variables.

See also

- *Chapter 1.4.1.8.2 “Declaration of Variables ” on page 222*
- *Chapter 1.4.1.20.4.11.3 “Dialog Box 'Project Settings' - 'Compileoptions'” on page 1173*
- *Chapter 1.4.1.20.2.18.3 “Object 'Function'” on page 886*
- *Chapter 1.4.1.20.2.18.2 “Object 'Function Block'” on page 883*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.20.2.18.4 “Object 'Interface'” on page 888*
- *Chapter 1.4.1.20.2.18.6 “Object 'Interface Method'” on page 894*
- *Chapter 1.4.1.19.2.11 “Constant Variables - 'CONSTANT'” on page 534*

Global variables - VAR_GLOBAL

Global variables are ordinary variables, constants, external or remanent variables that are recognized within the entire project.

You declare global variables in global variable lists or in the declaration section of programming objects between the keywords `VAR_GLOBAL` and `END_VAR`.

The system recognizes a global variable when you prepend the variable name with a dot (for example, `.iGlobVar1`).

> **NOTICE!**
> If a local variable that is declared in a block has the same name as a global variable, then it has precedence within the block.

> **NOTICE!**
> For compiler version 3.2.0.0 and later, CODESYS always initializes global variables before the local POU variables.

Example

```
VAR_GLOBAL
    iVarGlob1 : INT;
END_VAR
```

See also

- *Chapter 1.4.1.20.2.10 “Object 'GVL' - Global Variable List” on page 871*
- *Chapter 1.4.1.19.3.69 “Operator - Global namespace” on page 629*

Temporary variable - VAR_TEMP

This function is an extension of the IEC 61131-3 standard.

You declare temporary variables locally between the keywords `VAR_TEMP` and `END_VAR`.

`VAR_TEMP` declarations are possible only in program blocks and function blocks.

CODESYS initializes temporary variables each time the block is called.

The application can access the temporary variables only in the implementation section of a program block or a function block.

Example

```
VAR_TEMP
        iVarTmp1 : INT; (*1st temporary variable *)
END_VAR
```

Static variables - VAR_STAT

This function is an extension of the IEC 61131-3 standard.

You declare static variables locally between the keywords `VAR_STAT` and `END_VAR`. CODESYS initializes static variables the first time each block is called.

You can access static variables only from within the namespace where the variables are declared (like static variables in C). But static variables retain their values when the application leaves the block. For example, you can use static variables as counters for function calls.

You can extend static variables with an attribute keyword.

Example

```
VAR_STAT
    iVarStat1 : INT;
END_VAR
```

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

External variables - VAR_EXTERNAL

External variables are global variables that are imported into a block.

You declare these variables between the keywords `VAR_EXTERNAL` and `END_VAR`. If the global variable does not exist, then an error message is printed.

> **NOTICE!**
> CODESYS does not require you to declare a global variable as external in order to use it in a POU. The keyword exists only for maintaining compliance with IEC 61131-3.

Syntax

```
<POU keyword> <POU name>
VAR_EXTERNAL
    <variable name> : <data type>;
END_VAR
```

Initialization is not permitted.

Example

```
FUNCTION_BLOCK FB_DoSomething
VAR_EXTERNAL
    iVarExt1 : INT; (* 1st external variable *)
END_VAR
```

See also

- *Chapter 1.4.1.20.2.10 “Object 'GVL' - Global Variable List” on page 871*

Instance variables - VAR_INST

CODESYS does not save a `VAR_INST` method variable in a method stack, but in the stack of the function block instance. This means that the `VAR_INST` variable functions like other Variables of the function block instance, and it is not reinitialized each time the method is called.

`VAR_INST` variables are permitted in methods only and you can access these variables only within the method. The variable values of instance variables are monitored in the declaration section of the method.

You can extend instance variable with an attribute keyword.

Example

```
METHOD meth_last : INT
VAR_INPUT
  iVar : INT;
END_VAR
VAR_INST
  iLast : INT := 0;
END_VAR
meth_last := iLast;
iLast := iVar;
```

Configuration variables - VAR_CONFIG

Use configuration variables for assigning complete addresses to variables that are declared in function blocks with incomplete addresses and will be mapped on device I/Os.

Declare the variables in a global variables list between `VAR_CONFIG` and `END_VAR`. The global variables list is termed "variables configuration", where you type the configuration variables with a complete instance path and the correct address.

Example

Declaration of the variable `xLocIn` with incomplete address `%I*` in a function block:

```
FUNCTION_BLOCK locio

VAR
    xLocIn AT %I* : BOOL := TRUE;
END_VAR
```

The `locio` function block is used in the `PLC_PRG` program:

```
PROGRAM PLC_PRG

VAR
    locioVar1 : locio;
END_VAR
```

The correct variables configuration in the global variable list is as follows:

```
VAR_CONFIG

  PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;

END_VAR
```

See also

- ↳ *Chapter 1.4.1.8.11.1 “Variables configuration - VAR_CONFIG” on page 279*

Constant Variables - 'CONSTANT'

Constant variables are declared in global variable lists or in the declaration part of programming objects. In implementations, constant variables can be accessed as read-only via the instance path.

Syntax

```
<scope> CONSTANT
    <identifier> : <data type> := <initial value> ;
END_VAR

<scope> : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
<data type>: <elementary data type> | <user defined data type> |
<function block>
<initial value> : <literal value> | <identifier> | <expression>
```

Always assign an initialization value when declaring a constant variable. Then the constant cannot be written any more.

Example

Declaration

```
VAR CONSTANT
    c_rTAXFACTOR : REAL := 1.19;
END_VAR
```

Call

```
rPrice := rValue * c_rTAXFACTOR;
```

You have read-only access to constant variables in an implementation. Constant variables are located to the right of the assignment operator.

See also

- *Chapter 1.4.1.19.2.4 “Input/Output Variable (VAR_IN_OUT)” on page 527*
- *“Constants and literals” on page 632*

Persistent Variable - PERSISTENT

Persistent variables are declared in the declaration section `VAR_GLOBAL RETAIN PERSISTENT` in the persistent global variable list. For variables that are marked with the `PERSISTENT` keyword outside of the persistence editor, instance paths are added there.

As of CODESYS version 3.3.0.1, a variable declaration with `PERSISTENT RETAIN` has the same effect as with `RETAIN PERSISTENT` or `PERSISTENT`.

Syntax of the declaration in the global persistent variable list `PersistentVars`:

```
VAR_GLOBAL PERSISTENT RETAIN
    <identifier> : <data type> (:= <initialization>)?;
 <instance path to POU variable>
END_VAR
```

Syntax of the declaration in POUs

```
<scope> PERSISTENT RETAIN
    <identifier> : <data type> ( := <initialization> )?; //
( ... )? : Optional
END_VAR
<scope> : VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | VAR_STAT |
VAR_GLOBAL
```

An assignment of inputs, outputs, or memory addresses with the `AT` keyword is not permitted.

Never use the `POINTER TO` data type in persistent variable lists. If the application is downloaded again, their addresses could change. The corresponding compiler warnings are shown in the message window.

If you frequently change the names or data types of remanent variables, then it is better to declare them as retain variables with the `RETAIN` keyword only.

NOTICE!
Avoid inserting instance paths because in this case twice as much memory is used and a higher cycle time can occur. Instead, declare variables in the list of persistent variables.

Example

Declaration in the persistent variable list `PersistentVars`:

```
{attribute 'qualified_only'}
VAR_GLOBAL PERSISTENT RETAIN
    g_iCounter : INT;
    // Generated instance path of persistent variable
    PLC_PRG.fb_A.iPersistentCounter_A: INT;
END_VAR
```

Declaration in the function block `FB_A`:

```
FUNCTION_BLOCK FB_A
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR PERSISTENT
    iPersistentCounter_A : INT;
END_VAR
```

Declaration in the program `PLC_PRG`:

```
VAR
    fb_A1 : FB_A;
END_VAR
```

Possible declaration locations

Directly in the persistent global variable list	The variable is persistent and lies in the protected memory area.
Locally in a program with an instance path in the persistent variable list Locally in a function block with an instance path in the persistent variable list	The variable is persistent and located in the protected memory area and in the memory (double allocation).
Only locally in a program Only locally in a function block	This variable is not persistent. A warning is shown in the message window. Hint: Click *"Declarations ➔ Add All Instance Paths"* to import the variables into the persistent variable list.
Locally in a function	This declaration does not have any effect. The variable is not persistent.

In the persistence editor, click “Declarations ➔ Add All Instance Paths” if local variables are marked with `PERSISTENT`*.*

Whenever possible, avoid marking variables, which are declared in a function block, with `PERSISTENT`*. This is because the function block instance is stored entirely in remanent memory and not just the marked variable.*

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.8.19.5 “Declaring VAR PERSISTENT Variables ” on page 308*
- *Chapter 1.4.1.20.3.17.4 “Command 'Add all instance paths'” on page 1124*
- *Chapter 1.4.1.8.19.1 “Preserving data with persistent variables” on page 304*

Retain Variable - RETAIN

Retain variables are declared by the keyword `RETAIN` is added in programming objects in the scope `VAR`, `VAR_INPUT`, `VAR_OUTPUT`, `VAR_IN_OUT`, `VAR_STAT`, or `VAR_GLOBAL`.

Syntax for the declaration

```
<scope> RETAIN
    <identifier>: <data type> ( := <initialization> )? // ( ... )? :
Optional
END_VAR
<scope> : VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | VAR_STAT |
VAR_GLOBAL
```

An assignment of inputs, outputs, or memory addresses with the `AT` keyword is not permitted.

Example

In a POU:

```
VAR RETAIN
    iVarRetain: INT;
END_VAR
```

In a GVL:

```
VAR_GLOBAL RETAIN
    g_iVarRetain: INT;
END_VAR
```

Possible declaration locations

Locally in a program	Only the variable is located in the retain memory area. Info: When using redundancy, the entire program with all of its data is located in the retain memory area.
Globally in a global variable list	Only the variable is located in the retain memory area. Info: When using redundancy, the entire global variable list with all of its data is located in the retain memory area.

Locally in a function block	The entire instance of the function block with all of its data is located in the retain memory area. Only the declared retain variable is protected.
Locally in a function	The variable is not located in the retain memory area. This declaration does not have any effect.
Locally and persistently in a function	The variable is not located in the retain memory area. This declaration does not have any effect.

Whenever possible, avoid using `RETAIN` to mark the variables of a function block.

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.8.19.5 “Declaring VAR PERSISTENT Variables ” on page 308*
- *Chapter 1.4.1.20.3.17.4 “Command 'Add all instance paths'” on page 1124*
- *Chapter 1.4.1.8.19.2 “Preserving data with retain variables” on page 306*

SUPER

`SUPER` is a special variable and is used for object-oriented programming.

`SUPER` is the pointer of a function block to the basic function block instance from which the function block was generated. The `SUPER` pointer thus also permits access to the implementation of the methods of the basic function block (basic class). A `SUPER` pointer is automatically available for each function block.

You can use `SUPER` only in methods and in the associated function block implementations.

Dereferencing of the pointer: `SUPER^`

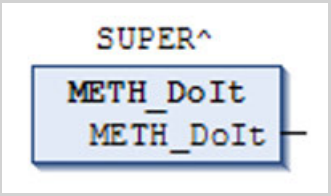
Use of the `SUPER` pointer: with the help of the keyword `SUPER` you call a method that is valid in the instance of the basic class or parent class.

Examples

ST:

```
SUPER^.METH_DoIt();
```

FBD/CFC/LD



`THIS` is not yet implemented for the instruction list (IL).

Examples

Use of `SUPER` and `THIS` pointers

```
FUNCTION_BLOCK FB_Base
VAR_OUTPUT
  iCnt : INT;
END_VAR

METHOD METH_DoIt : BOOL
 iCnt := -1;

METHOD METH_DoAlso : BOOL
 METH_DoAlso := TRUE;

FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_OUTPUT
 iBase : INT;
END_VAR

THIS^.METH_DoIt();  //Call of the methods of FB_1
THIS^.METH_DoAlso();

SUPER^.METH_DoIt();   //Call of the methods of FB_Base
SUPER^.METH_DoAlso();
iBase := SUPER^.iCnt;

METHOD METH_DoIt : BOOL
 iCnt := 1111;
 METH_DoIt := TRUE;

PROGRAM PLC_PRG
VAR
 myBase : FB_Base;
 myFB_1 : FB_1;
 iTHIS : INT;
 iBase : INT;
END_VAR

myBase();
iBase := myBase.iCnt;
myFB_1();
iTHIS := myFB_1.iCnt;
```

See also

- *Chapter 1.4.1.19.5 “Data Types” on page 646*
- *Chapter 1.4.1.19.2.15 “THIS” on page 539*

THIS

`THIS` is a special variable and is used for object-oriented programming.

`THIS` is the pointer of a function block to its own function block instance. A `THIS` pointer is automatically available for each function block.

You can use `THIS` only in methods and in function blocks. `THIS` is available for the implementation in the input assistant in the category *“Keywords”*.

Dereferencing of the pointer: `THIS^`

Use of the `THIS` pointer

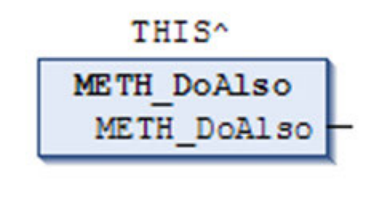
- If a local variable obscures a function block variable in a method, you can set the function block variable with the `THIS` pointer. See example below (1)
- If the pointer to the function block's own function block instance is referenced for use in a function. (See example below (2))

Examples

ST:

```
THIS^.METH_DoIt();
```

FBD/CFC/LD:



`THIS` is not yet implemented for the instruction list (IL).

Examples

(1) The local variable `iVarB` obscures the function block variable `iVarB`.

```
FUNCTION_BLOCK  fbA
VAR_INPUT
 iVarA: INT;
END_VAR
iVarA := 1;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
 iVarB: INT := 0;
END_VAR
iVarA := 11;
iVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
 iVarB: INT;
END_VAR
iVarB := 22;    // The local variable iVarB is set.
THIS^.iVarB := 222;    // The function block variable iVarB is set
even though iVarB is obscured.

PROGRAM PLC_PRG
VAR
 MyfbB: fbB;
END_VAR

MyfbB(iVarA:=0, iVarB:= 0);
MyfbB.DoIt();
```

(2) A function call requires the reference to its own instance.

```
FUNCTION funA
VAR_INPUT
 pFB: fbA;
END_VAR
...;

FUNCTION_BLOCK  fbA
VAR_INPUT
 iVarA: INT;
END_VAR
...;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
 iVarB: INT := 0;
END_VAR
iVarA := 11;
iVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
 iVarB: INT;
END_VAR
iVarB := 22;    //The local variable iVarB is set.
funA(pFB := THIS^);    //funA is called via THIS^.
PROGRAM PLC_PRG
```

```
VAR
 MyfbB: fbB;
END_VAR
MyfbB(iVarA:=0 , iVarB:= 0);
MyfbB.DoIt();
```

See also

- *Chapter 1.4.1.19.5.12 “Pointers” on page 656*
- *Chapter 1.4.1.19.2.14 “SUPER” on page 538*

1.4.1.19.3 Operators

CODESYS V3 supports all IEC-61131-3 operators. These operators are recognized implicitly throughout the project. In addition to these IEC operators, CODESYS also supports some non-IEC 61131-3 operators.

Operators are used in blocks, such as functions.

For information about the processing order (binding strength) of the ST operators, please refer to the section on ST expressions.

CAUTION!
For operations with floating-point data types, the computational result depends on the applied target system hardware.

CAUTION!
For operations with overflow or underflow in the data type, the computational result depends on the applied target system hardware.

Overflow/underflow in the data type

The CODESYS compiler generates code for the target device and computes temporary results always with the native size that is defined by the target device. For example, computation is performed at least with 32-bit temporary values on x86 and ARM systems and always with 64-bit temporary values on x64 systems. This provides considerable advantages in the computation speed and often also produces the desired result. But this also means that an overflow or underflow in the data type is not truncated in some cases.

Examples

Example 1

The result of this addition is not truncated and the result in `dwVar` is `65536`.

```
VAR
wVar : WORD;
dwVar: DWORD;
END_VAR

wVar := 65535;
dwVar := wVar + 1;
```

Example 2

The overflow and underflow in the data type is not truncated and the results (`bVar1, bVar2`) of both comparisons are `FALSE` on 32-bit and 64-bit hardware.

```
VAR
wVar1 : WORD;
wVar2 : WORD;
bVar1 : BOOL;
bVar2 : BOOL;
END_VAR

wVar1 := 65535;
wVar2 := 0;
bVar1 := (wVar1 + 1) = wVar2;
bVar2 := (wVar2 - 1) = wVar1;
```

Example 3

By the assignment to `wVar3`, the value is truncated to the target data type `WORD` and the result `bvar1` is `TRUE`.

```
VAR
wVar1 : WORD;
wVar2 : WORD;
wVar3 : WORD;
bVar1 : BOOL;
END_VAR

wVar1 := 65535;
wVar2 := 0;
wVar3 := (wVar1 + 1);
bVar1 := wVar3 = wVar2;
```

Example 4

In order to force the compiler to truncate the temporary results, a conversion can be inserted.

The type conversion makes sure that both comparisons are 16-bit only and the results (`bVar1, bVar2`) of both comparisons are each `TRUE`.

```
VAR
wVar1 : WORD;
wVar2 : WORD;
bVar1 : BOOL;
bVar2 : BOOL;
END_VAR

wVar1 := 65535;
wVar2 := 0;
bVar1 := TO_WORD(wVar1 + 1) = wVar2;
bVar2 := TO_WORD(wVar2 - 1) = wVar1;
```

Arithmetic operators

↳ *Chapter 1.4.1.19.3.1 “Operator 'ADD'” on page 546*
↳ *Chapter 1.4.1.19.3.3 “Operator 'SUB'” on page 548*
↳ *Chapter 1.4.1.19.3.2 “Operator 'MUL'” on page 547*
↳ *Chapter 1.4.1.19.3.4 “Operator 'DIV'” on page 549*
↳ *Chapter 1.4.1.19.3.5 “Operator 'MOD'” on page 550*
↳ *Chapter 1.4.1.19.3.6 “Operator 'MOVE'” on page 550*
↳ *Chapter 1.4.1.19.3.7 “Operator 'INDEXOF'” on page 550*
↳ *Chapter 1.4.1.19.3.8 “Operator 'SIZEOF'” on page 551*
↳ *Chapter 1.4.1.19.3.9 “Operator 'XSIZEOF'” on page 551*

Bitstring operators

↳ *Chapter 1.4.1.19.3.11 “Operator 'AND'” on page 552*
↳ *Chapter 1.4.1.19.3.12 “Operator 'OR' ” on page 552*
↳ *Chapter 1.4.1.19.3.13 “Operator 'XOR'” on page 553*
↳ *Chapter 1.4.1.19.3.10 “Operator 'NOT'” on page 552*
↳ *Chapter 1.4.1.19.3.14 “Operator 'AND_THEN'” on page 553*
↳ *Chapter 1.4.1.19.3.15 “Operator 'OR_ELSE'” on page 553*

Bitshift operators

↳ *Chapter 1.4.1.19.3.16 “Operator 'SHL'” on page 554*
↳ *Chapter 1.4.1.19.3.17 “Operator 'SHR'” on page 555*
↳ *Chapter 1.4.1.19.3.18 “Operator 'ROL'” on page 556*
↳ *Chapter 1.4.1.19.3.19 “Operator 'ROR'” on page 557*

Selection operators

↳ *Chapter 1.4.1.19.3.20 “Operator 'SEL'” on page 558*
↳ *Chapter 1.4.1.19.3.21 “Operator 'MAX'” on page 559*
↳ *Chapter 1.4.1.19.3.22 “Operator 'MIN'” on page 559*
↳ *Chapter 1.4.1.19.3.23 “Operator 'LIMIT'” on page 560*
↳ *Chapter 1.4.1.19.3.24 “Operator 'MUX'” on page 560*

Comparison operators

A comparison operator is a Boolean that compares two inputs (first and second operand).

↳ *Chapter 1.4.1.19.3.25 “Operator 'GT'” on page 561*
↳ *Chapter 1.4.1.19.3.26 “Operator 'LT'” on page 561*
↳ *Chapter 1.4.1.19.3.27 “Operator 'LE'” on page 561*
↳ *Chapter 1.4.1.19.3.28 “Operator 'GE'” on page 562*
↳ *Chapter 1.4.1.19.3.29 “Operator 'EQ'” on page 562*
↳ *Chapter 1.4.1.19.3.30 “Operator 'NE'” on page 562*

Address operators

↳ *Chapter 1.4.1.19.3.31 “Operator 'ADR'” on page 563*
↳ *Chapter 1.4.1.19.3.32 “Operator 'Content Operator'” on page 564*
↳ *Chapter 1.4.1.19.3.33 “Operator 'BITADR'” on page 564*

Call operators

↳ *Chapter 1.4.1.19.3.34 “Operator 'CAL'” on page 565*

Type conversion operators

You can explicitly call type conversion operators. The type conversion operators described below are available for typed conversions from one elementary type to another elementary type, as well as for overloading. Conversions from a larger type to a smaller type are also implicitly possible (for example, from `INT` to `BYTE` or from `DINT` to `WORD`).

Typed conversion: `<elementary data type> _TO_ <another elementary data type>`

Overloaded conversion: `TO_ <elementary data type>`

Elementary data types:

```
<elementary  data type> =
__UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DATE_AND_TIME
| DINT | DT | DWORD | INT | LDATE | LDATE_AND_TIME | LDT | LINT |
LREAL | LTIME | LTOD | LWORD | REAL | SINT | TIME | TOD | UDINT |
UINT | ULINT | USINT | WORD
```

The keywords `T`, `TIME_OF_DAY` and `DATE_AND_TIME` are alternative forms for the data types `TIME`, `TOD`, and `DT`. `T`, `TIME_OF_DAY` and `DATE_AND_TIME` are not represented as a type conversion command.

NOTICE!

If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

NOTICE!

String manipulation when converting to `STRING` or `WSTRING`

When converting the type to `STRING` or `WSTRING`, the typed value is left-aligned as a character string and truncated if it is too long. Therefore, declare the return variable for the type conversion operators `<>_TO_STRING` and `<>_TO_WSTRING` long enough that the character string has enough space without any manipulation.

See also

- *Chapter 1.4.1.19.3.38 “ Floating-Point Number Conversion” on page 584*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*
- *Chapter 1.4.1.19.3.41 “Date and Time Conversion” on page 600*
- *Chapter 1.4.1.19.3.39 “String Conversion” on page 587*
- *Chapter 1.4.1.19.3.42 “Operator 'TRUNC' ” on page 606*
- *Chapter 1.4.1.19.3.43 “Operator 'TRUNC_INT' ” on page 606*

Numeric Operators

Chapter 1.4.1.19.3.44 “Operator 'ABS'” on page 607

Chapter 1.4.1.19.3.45 “Operator 'SQRT'” on page 607

Chapter 1.4.1.19.3.46 “Operator 'LN'” on page 607

Chapter 1.4.1.19.3.47 “Operator 'LOG'” on page 608

Chapter 1.4.1.19.3.48 “Operator 'EXP'” on page 608

Chapter 1.4.1.19.3.49 “Operator 'EXPT'” on page 608

Chapter 1.4.1.19.3.50 “Operator 'SIN'” on page 609

Chapter 1.4.1.19.3.53 “Operator 'ASIN'” on page 610

Chapter 1.4.1.19.3.51 “Operator 'COS'” on page 609

↳ *Chapter 1.4.1.19.3.52 “Operator 'TAN'” on page 610*

↳ *Chapter 1.4.1.19.3.54 “Operator 'ACOS'” on page 611*

↳ *Chapter 1.4.1.19.3.55 “Operator 'ATAN'” on page 611*

Namespace operators

Namespace operators are extended from IEC 61131-3 operators. They make it possible for you to provided unique access to variables and modules, even when you use the same name multiple times for variables or modules in a project.

↳ *Chapter 1.4.1.19.3.69 “Operator - Global namespace” on page 629*

↳ *Chapter 1.4.1.19.3.70 “Operator - Namespace for global variables lists” on page 629*

↳ *Chapter 1.4.1.19.3.72 “Operator - Enumeration namespace” on page 630*

↳ *Chapter 1.4.1.19.3.71 “Operator - Library namespace” on page 630*

↳ *Chapter 1.4.1.19.3.73 “Operator '__POOL'” on page 630*

Multicore operators

Working with different tasks requires the synchronization of these tasks. This is especially true when working on multicore platforms. Some special operators are provided in CODESYS to support this synchronization.

These operators are extensions of IEC-61131-3. The operators `TEST_AND_SET` and `__COMPARE_AND_SWAP` are used for similar tasks.

↳ *Chapter 1.4.1.19.3.68 “Operator 'TEST_AND_SET'” on page 628*

↳ *Chapter 1.4.1.19.3.64 “Operator '__COMPARE_AND_SWAP” on page 625*

↳ *Chapter 1.4.1.19.3.65 “Operator '__XADD'” on page 626*

Other operators

↳ *Chapter 1.4.1.19.3.56 “Operator '__DELETE'” on page 611*

↳ *Chapter 1.4.1.19.3.57 “Operator '__ISVALIDREF'” on page 614*

↳ *Chapter 1.4.1.19.3.58 “Operator '__NEW'” on page 614*

↳ *Chapter 1.4.1.19.3.59 “Operator '__QUERYINTERFACE'” on page 617*

↳ *Chapter 1.4.1.19.3.60 “Operator '__QUERYPOINTER'” on page 618*

↳ *Chapter 1.4.1.19.3.74 “Operator 'INI'” on page 631*

↳ *Chapter 1.4.1.19.3.61 “Operators '__TRY', '__CATCH', '__FINALLY', '__ENDTRY'” on page 619*

↳ *Chapter 1.4.1.19.3.66 “Operator '__POSITION'” on page 627*

↳ *Chapter 1.4.1.19.3.67 “Operator '__POUNAME'” on page 627*

Operator 'ADD'

The IEC operator adds variables.

Permitted data types: `__UXINT | __XINT | __XWORD | BYTE | DATE | DATE_AND_TIME | DINT | DT | DWORD | INT | LDATE | LDATE_AND_TIME | LDT | LINT | LREAL | LTIME | LTOD | LWORD | REAL | SINT | TIME | TIME_OF_DAY | TOD | UDINT | UINT | ULINT | USINT | WORD`

Possible combinations for time data types:

- `TIME + TIME = TIME`
- `TIME + LTIME = LTIME`
- `LTIME + LTIME = LTIME`

Possible combinations for date and time data types:

- TOD + TIME = TOD
- DT + TIME = DT
- TOD + LTIME = LTOD
- DT + LTIME = LDT
- LTOD + TIME = LTOD
- LDT + LTIME = LDT
- LTOD +LTIME = LTOD
- LDT + LTIME = LDT

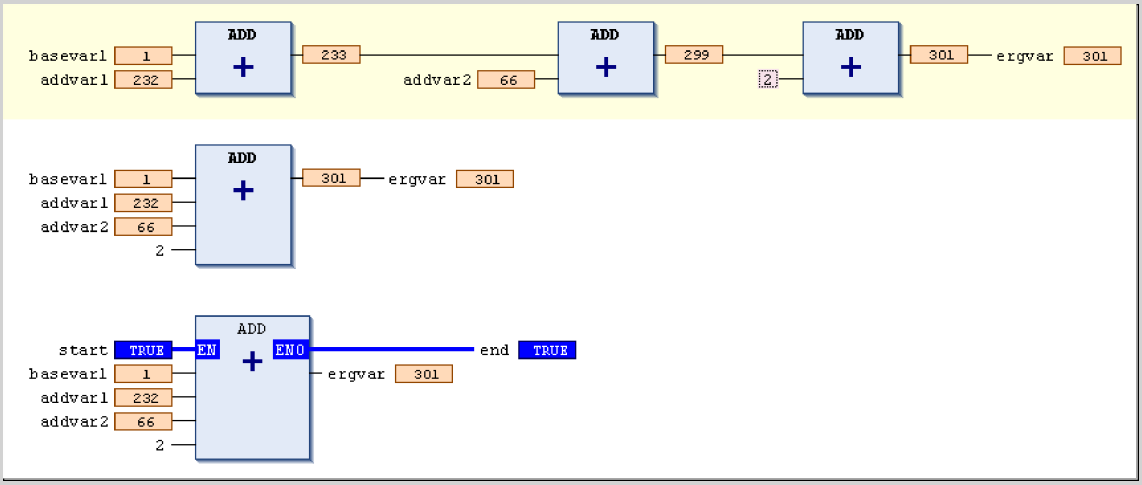
Feature in the FBD/LD editor: You can extend the ADD operator to function block inputs. The number of additional function block inputs is limited.

Examples

ST:

```
var1 := 7+2+4+7;
```

FBD:



Operator 'MUL'

This IEC operator is used for multiplying variables.

Permitted data types: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME

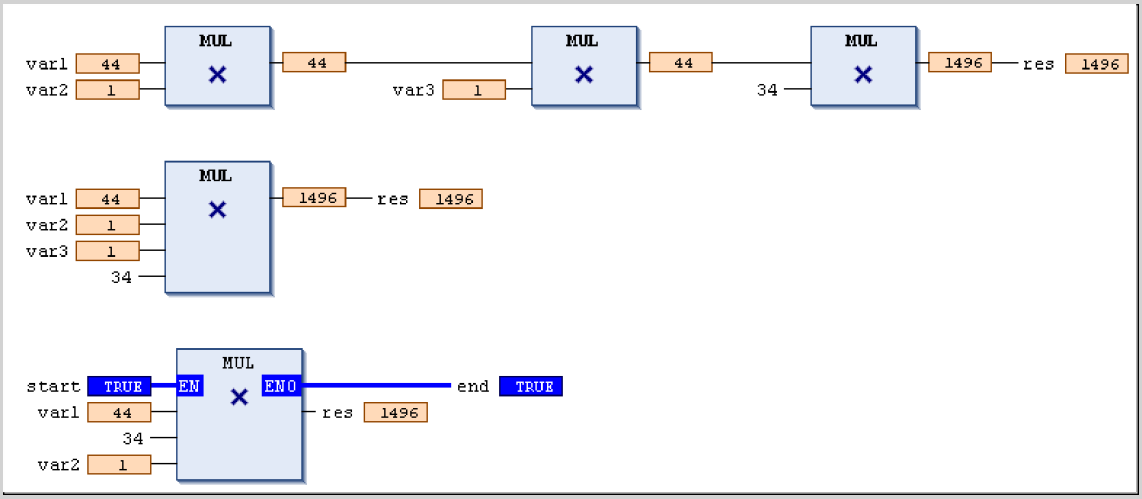
Feature in the FBD/LD editor: You can extend the MUL operator to additional function block inputs. The number of additional function block inputs is limited.

Examples

ST:

```
var1 := 7*2*4*7;
```

FBD:



Operator 'SUB'

The IEC operator subtracts variables.

Permitted data types: `BYTE`, `WORD`, `DWORD`, `LWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `LINT`, `ULINT`, `REAL`, `LREAL`, `TIME`, `LTIME`, `TIME_OF_DAY (TOD)`, `LTIME_OF_DAY (LTOD)`, `DATE`, `LDATE`, `DATE_AND_TIME (DT)` `LDATE_AND_TIME (DT)`

Possible combinations for time data types:

- `TIME - TIME = TIME`
- `LTIME - LTIME = LTIME`

Possible combinations for date and time data types:

- `DATE - DATE = TIME`
- `LDATE - LDATE = LTIME`
- `TOD - TIME = TOD`
- `LTOD - LTIME = LTOD`
- `TOD - TOD = TIME`
- `LTOD - LTOD = LTIME`
- `DT - TIME = DT`
- `LDT - LTIME = LDT`
- `DT - DT = TIME`
- `LDT - LDT = LTIME`

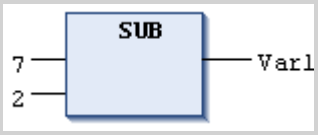
Negative `TIME/LTIME` values are undefined.

Examples

ST:

```
var1 := 7-2;
```

FBD:



Operator 'DIV'

This IEC operator is used for dividing variables.

Permitted data types: `BYTE`, `WORD`, `DWORD`, `LWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `LINT`, `ULINT`, `REAL`, `LREAL`, `TIME`

NOTICE!
Division by zero may have different results depending on the target system.

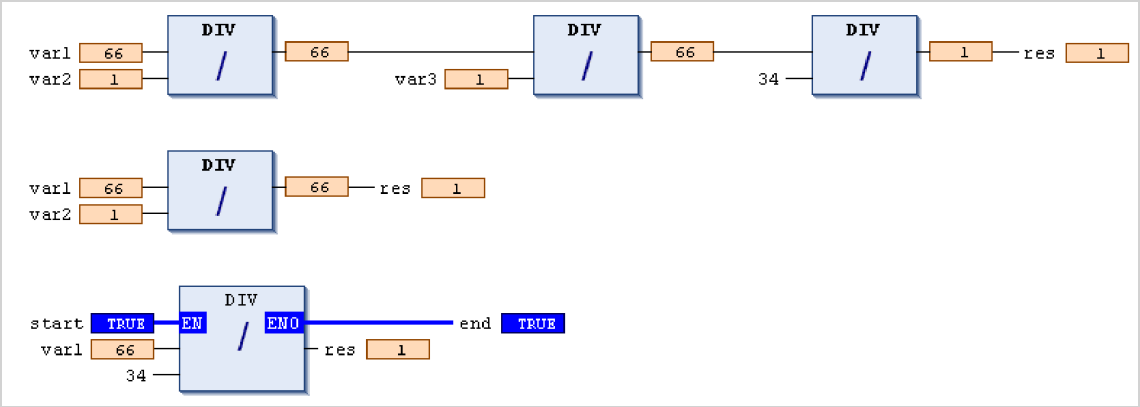
Examples

ST:

```
var1 := 8/2;
```

FBD:

1. Series of DIV blocks, 2. Single DIV block, 3. DIV blocks with EN/ENO parameters



Please note that it is possible to monitor division by zero at runtime by using the implicit monitoring functions `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`.

See also

- *Chapter 1.4.1.20.2.19.2 “POU 'CheckDivInt'” on page 909*
- *Chapter 1.4.1.20.2.19.3 “POU 'CheckDivLInt'” on page 909*
- *Chapter 1.4.1.20.2.19.4 “POU 'CheckDivReal'” on page 910*
- *Chapter 1.4.1.20.2.19.5 “POU 'CheckDivLReal'” on page 911*

Operator 'MOD'

This IEC operator is used for modulo division.

The result of the function is the integer remainder of division.

Permitted data types: `BYTE`, `WORD`, `DWORD`, `LWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `LINT`, `ULINT`

> **NOTICE!**
> Division by zero may have different results depending on the target system.

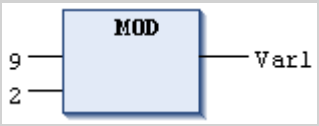
Examples

Result in Var1: 1

ST:

```
var1 := 9 MOD 2;
```

FBD:



Operator 'MOVE'

This IEC operator is used for assigning a variable to another variable of a corresponding type.

Because the `MOVE` block is available in the CFC, FBD, and LD editors, you can also use the EN/ENO functionality for variable assignment.

CFC with EN/ENO function:

CODESYS assigns the value of `var1` to `var2` only if "en_i" yields `TRUE`.

ST:

```
ivar2 := MOVE(ivar1);
```

This corresponds to:

```
ivar2 := ivar1;
```

Operator 'INDEXOF'

This operator is an extension of the IEC 61131-3 standard.

Instead of the `INDEXOF` operator, the `ADR` operator is provided in CODESYS V3 for obtaining a pointer at the index of a block.

See also

- *Chapter 1.4.1.19.3.31 “Operator 'ADR'” on page 563*

Operator 'SIZEOF'

The operator is an extension of the IEC 61131-3 standard.

The operator is used for defining the number of bytes that are required by the variable `x`. The operator `SIZEOF` always yields an unsigned value. The type of return variable adapts to the detected size of the variable `x`.

In compiler version 3.5.16.0 and higher, the operator `XSIZEOF` should be used instead of this operator.

Return value of `SIZEOF(x)`	Data type of the constant which CODESYS uses implicitly for the detected size.
0 <= size of x < 256	`USINT`
256 <= size of x < 65536	`UINT`
65536 <= size of x < 4294967296	`UDINT`
4294967296 <= size of x	`ULINT`

Examples

Result in `var1`: `10`.

ST:

```
arr1 : ARRAY[0..4] OF INT;
var1 : INT;

var1 := SIZEOF(arr1);  (* var1 := USINT#10; *)
```

See also

- *Chapter 1.4.1.19.3.9 “Operator 'XSIZEOF'” on page 551*

Operator 'XSIZEOF'

The operator is an extension of the IEC 61131-3 standard.

The operator is used for defining the number of bytes that are required by the variable x. The data type of the return value is `ULINT` on 64-bit platforms and `UDINT` on all other platforms.

The operator `XSIZEOF` should be used instead of the operator `SIZEOF`. Because the data type of the return value is fixed, problems do not occur for `XSIZEOF`, which do occur in the case of the operator `SIZEOF`.

Example

Variable udiVarX is

ST:

```
udiVarX : UDINT;  (* Data type for 64-bit platforms: ULINT *)

udiVarX := XSIZEOF(<variable>);
```

The variable udiVarX contains the number of bytes that the variable <variable> requires.

See also

- *Chapter 1.4.1.19.3.8 “Operator 'SIZEOF'” on page 551*

Operator 'NOT'

This IEC operator is used for the bitwise `NOT` of a bit operand.

When the respective input bit yields 0, the output bit also yields 1, and vice-versa.

Permitted data types: `BOOL`, `BYTE`, `WORD`, `DWORD`, `LWORD`

Examples

Result in `var1`: `2#0110_1100`

ST:

```
var1 := NOT 2#1001_0011;
```

FBD:

Operator 'AND'

This IEC operator is used for the bitwise `AND` of bit operands.

When the input bits all yield 1, the output bit also yields 1; otherwise 0.

Permitted data types: `BOOL`, `BYTE`, `WORD`, `DWORD`, `LWORD`

Examples

Result in `var1` ist `2#1000_0010`

ST:

```
var1 := 2#1001_0011 AND 2#1000_1010;
```

FBD:

Operator 'OR'

This IEC operator is used for the bitwise `OR` of bit operands.

When at least one of the input bits yields 1, the output bit also yields 1; otherwise 0.

Permitted data types: `BOOL`, `BYTE`, `WORD`, `DWORD`, `LWORD`

Examples

Result in `Var1` ist `2#1001_1011`

ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010;
```

FBD:

Operator 'XOR'

This IEC operator is used for the bitwise XOR of bit operands.

When only one of the two input bits yields 1, the output bit also yields 1. When both inputs yield 1 or 0, then the output yields 0.

Permitted data types: BOOL, BYTE, WORD, DWORD, LWORD

> **NOTICE!**
> Please note the following behavior of the XOR block in extended form (more than two inputs): CODESYS compares the inputs in pairs and then the corresponding results (according to the standard, but not necessarily according to expectations).

Examples

Result in var1: 2#0001_1001

ST:

```
var1 := 2#1001_0011 XOR 2#1000_1010;
```

FBD:

Operator 'AND_THEN'

This operator is an extension of the IEC 61131-3 standard.

The AND_THEN operator is permitted only for programming in structured text with the AND operation of BOOL and BIT operands with short-circuit evaluation. This means that:

When all operands yield TRUE, the result of the operands also yield TRUE; otherwise FALSE.

However, CODESYS also executes the expressions on other operands only if the first operand of the AND_THEN operator is TRUE. This can prevent problems with null pointers, for example in conditions such as `IF (ptr <> 0 AND_THEN ptr^ = 99) THEN...`.

In contrast, CODESYS always evaluates all operands when using the AND IEC operator.

See also

- ↳ *Chapter 1.4.1.19.3.11 “Operator 'AND'” on page 552*

Operator 'OR_ELSE'

This operator is an extension of the IEC 61131-3 standard.

The OR_ELSE operator is permitted only for programming in structured text: OR operation of BOOL and BIT operands; with short-circuit evaluation. This means:

When at least one of the operands yields TRUE, the result of the operation also yields TRUE; otherwise FALSE.

In contrast to using the OR IEC operator, for OR_ELSE the expressions on all other operators are not evaluated as soon as one of the operands is evaluated as TRUE.

Example

```
VAR
    bEver: BOOL;
    bX: BOOL;
    dw: DWORD := 16#000000FF;
END_VAR
bEver := FALSE;
bX := dw.8 OR_ELSE dw.1 OR_ELSE dw.1 OR_ELSE (bEver := TRUE);
```

`dw.8` is `FALSE` and `dw.1` is `TRUE`; therefore `bX` is the result of the operation `TRUE`. However, the expression at the third input is not executed, and `bEver` remains `FALSE`. On the other hand, if the standard OR operation was used, `bEver` would be set to TRUE.

See also

- *Chapter 1.4.1.19.3.12 “Operator 'OR' ” on page 552*

Operator 'SHL'

This IEC operator is used for bitwise shift of an operand to the left.

`erg := SHL (in, n)`

`in`: Operand that is shifted to the left

`n`: Number of bits to shift `in` to the left

NOTICE!
If `n` overwrites the data type width, then it depends on the target system how the `BYTE`, `WORD`, `DWORD`, and `LWORD` operands are padded. The target systems cause padding with zeros or `n MOD <tab width>`.

NOTICE!
Please note the number of bits that CODESYS uses for this operation as defined by the data type of the input variable `in`.

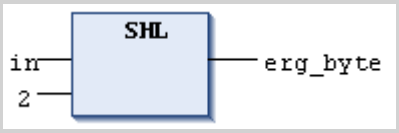
Examples

The results for `erg_byte` and `erg_word` are different, although the values of the `in_byte` and `in_word` input variables are the same and the data types of the input variables are different.

ST:

```
PROGRAM shl_st
VAR
 in_byte : BYTE := 16#45; (* 2#01000101 )
 in_word : WORD := 16#0045; (* 2#0000000001000101 )
 erg_byte : BYTE;
 erg_word : WORD;
 n: BYTE := 2;
END_VAR

erg_byte := SHL(in_byte,n); (* Result is 16#14, 2#00010100 *)
erg_word := SHL(in_word,n); (* Result is 16#0114,
2#0000000100010100 *)
```

FBD:



Operator 'SHR'

This IEC operator is used for bitwise shift of an operand to the right.

`erg := SHR (in, n)`

`in`: Operand that is shifted to the right

`n`: Number of bits for shifting `in` to the right

NOTICE!
If `n` overwrites the data type width, then it depends on the target system how the `BYTE`, `WORD`, `DWORD`, and `LWORD` operands are padded. The target systems cause padding with zeros or `n MOD <tab width>`.

Examples

ST:

```
PROGRAM shr_st
VAR
 in_byte : BYTE:=16#45; (* 2#01000101 )
 in_word : WORD:=16#0045; (* 2#0000000001000101 )
 erg_byte : BYTE;
 erg_word : WORD;
 n: BYTE :=2;
END_VAR

erg_byte := SHR(in_byte,n); (* Result is 16#11, 2#00010001 *)
erg_word := SHR(in_word,n); (* Result is 16#0011,
2#0000000000010001 *)
```

FBD:

Operator 'ROL'

This IEC operator is used for bitwise rotation of an operand to the left.

Permitted data types: `BYTE`, `WORD`, `DWORD`, `LWORD`

```
erg := ROL (in, n)
```

CODESYS moves `in` n-times one bit to the left and adds the bit to the leftmost position from the right.

NOTICE!
Please note the number of bits that CODESYS uses for this operation as defined by the data type of the input variable `in`. If this is a constant, then CODESYS uses the smallest possible data type. The data type of the output variables still does not influence this operation.

Examples

The results for `erg_byte` and `erg_word` are different depending on the data type of the input variables, although the values of the `in_byte` and `in_word` input variables are the same.

ST:

```
PROGRAM rol_st

VAR
 in_byte : BYTE := 16#45;
 in_word : WORD := 16#45;
 erg_byte : BYTE;
 erg_word : WORD;
 n: BYTE := 2;
END_VAR

erg_byte := ROL(in_byte,n); (* Result: 16#15 *)

erg_word := ROL(in_word,n); (* Result: 16#0114 *)
```

FBD:

IL:

```
LD    in_byte
ROL   n
ST    erg_byte
```

Operator 'ROR'

This IEC operator is used for bitwise rotation of an operand to the right.

Permitted data types: `BYTE`, `WORD`, `DWORD`, `LWORD`

```
erg := ROR(in,n)
```

CODESYS moves `in` n-times one bit to the right and adds the bit to the rightmost position from the left.

> *Please note the number of bits that CODESYS uses for this operation as defined by the data type of the input variable `in`. If this is a constant, then CODESYS uses the smallest possible data type. The data type of the output variables still does not influence this operation.*

Examples

The results for `erg_byte` and `erg_word` are different depending on the data type of the input variables, although the values of the `in_byte` and `in_word` input variables are the same.

ST:

```
PROGRAM ror_st

VAR
 in_byte : BYTE := 16#45;
 in_word : WORD := 16#45;
 erg_byte : BYTE;
 erg_word : WORD;
 n: BYTE := 2;
END_VAR

erg_byte := ROR(in_byte,n); (* Result: 16#51 *)

erg_word := ROR(in_word,n); (* Result: 16#4011 *)
```

FBD:

Operator 'SEL'

The IEC operator is used for bitwise selection.

`OUT := SEL(G, IN0, IN1)` means:

`OUT := IN0;` if `G = FALSE`

`OUT := IN1;` if `G = TRUE`

Permitted data types:

`IN0`, ..., `INn` and `OUT`: Any identical data type. Make sure that variables of the identical type are used at all three positions, especially when using user-defined data types. The compiler checks for type identity and returns any compile errors. The assignment of function block instances to interface variables is specifically not supported.

`G`: `BOOL`

NOTICE!

When `G` is TRUE, CODESYS does not compute an expression that precedes `IN0`.When `G` is `FALSE`, CODESYS does not compute an expression that precedes `IN1`.

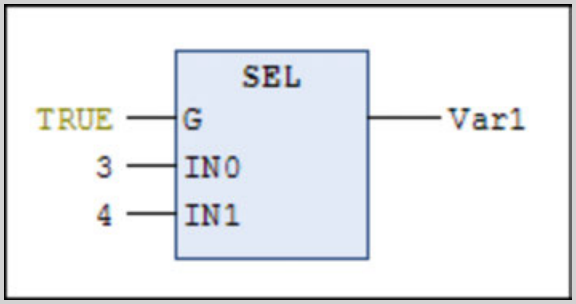
Caution: In the case of graphical programming languages, the expressions at `IN0` and `IN1` are computed independently of the `G` input when a *“Box”*, *“Jump”*, *“Return”*, *“Line Branch”*, or *“Edge Detection”* precedes.

Examples

ST:

```
Var1 := SEL(TRUE,3,4); (* Result: 4 *)
```

FBD:



Operator 'MAX'

This IEC operator is used for the maximum function. It yields the largest value of two values.

```
OUT := MAX(IN0, IN1)
```

Permitted data types: all

Examples

ST:

Result: 90

```
Var1 := MAX(30,40);
```

```
Var1 := MAX(40,MAX(90,30));
```

FBD:

Result: 90

Operator 'MIN'

This IEC operator is used for the minimum function. It yields the smallest value of two values.

```
OUT := MIN(IN0,IN1)
```

Permitted data types: all

Examples

Result: `30`

ST:

```
Var1:=MIN(90,30);

Var1 := MIN(MIN(90,30),40);
```

FBD:

Operator 'LIMIT'

This IEC selection operator is used for limiting.

`OUT := LIMIT(Min, IN, Max)`

Means: `OUT := MIN (MAX (IN, Min), Max)`

`Max` is the upper limit and `Min` is the lower limit for the result. If the `IN` value is above the `Max` upper limit, then LIMIT yields `Max`. If the value of `IN` is below the Min lower limit, then the result is `Min`.

Permitted data types for `IN` and `OUT`: all

Examples

Result in `Var1` is `80`

ST:

```
Var1 := LIMIT(30,90,80);
```

Operator 'MUX'

This IEC operator is used as a multiplexer.

`OUT := MUX(K, IN0,...,INn)`

Means: `OUT = IN_K`

Permitted data type for `K`: `BYTE`, `WORD`, `DWORD`, `LWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `LINT`, `ULINT`, o`UDINT`.

`IN0`, ..., `INn`, and `OUT`: Any identical data type. Make sure that variables of the identical type are used at all three positions, especially when using user-defined data types. The compiler checks for type identity and returns any compile errors. The assignment of function block instances to interface variables is specifically not supported.

`MUX` selects the K-th value from a set of values. The first value is K=0. If `K` is greater than the number of other inputs (n), then CODESYS passes on the last value (`INn`).

NOTICE!

For runtime optimization, CODESYS computes only the expression that precedes `IN_K`. However, CODESYS computes all branches in simulation mode.

Examples

Result in `Var1` is `30`.

ST:

```
Var1 := MUX(0,30,40,50,60,70,80);
```

Operator 'GT'

This IEC operator is used for the "greater than" function.

Permitted data types of the operands: any basic data type.

If the first operand is greater than the second operand, then the operator yields the result `TRUE`; otherwise `FALSE`.

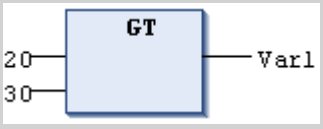
Examples

Result: FALSE

ST:

```
VAR1 := 20 > 30;
```

FBD:



Operator 'LT'

This IEC operator is used for the "less than" function.

Permitted data types of the operands: any basic data type.

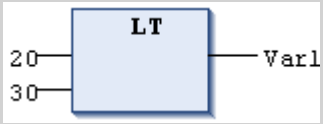
If the first operand is less than the second operand, then the operator yields the result `TRUE`; otherwise `FALSE`.

Examples

Result: `TRUE`

ST:

```
Var1 := 20 < 30;
```



Operator 'LE'

This IEC operator is used for the "less than or equal to" function.

Permitted data types of the operands: any basic data type.

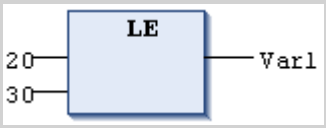
If the first operand is less than or equal to the second operand, then the operator yields the result `TRUE`; otherwise `FALSE`.

Examples

Result in `Var1`: `TRUE`

ST:

```
Var1 := 20 <= 30;
```



Operator 'GE'

This IEC operator is used for the "greater than or equal to" function.

Permitted data types of the operands: any basic data type.

If the first operand is greater than or equal to the second operand, then the operator yields the result `TRUE`; otherwise `FALSE`.

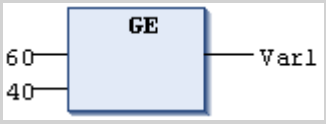
Examples

Result: `TRUE`

ST:

```
VAR1 := 60 >= 40;
```

FBD:



Operator 'EQ'

This IEC operator is used for the "equals" function.

Permitted data types of the operands: any basic data type, depending on target system and compiler version: structure data type.

If the operands are equal, then then the operator yields the result `TRUE`, otherwise `FALSE`.

Examples

Result: TRUE

ST:

```
VAR1 := 40 = 40;
```

FBD:

Operator 'NE'

This IEC operator is used for the "does not equal" function.

Permitted data types of the operands: any basic data type, depending on target system and compiler version: structure data type.

If the operands are not equal, then then the operator yields the result `TRUE`; otherwise `FALSE`.

If the target system supports the data type, then as from compiler version >= 3.5.7.0 also operands of type STRIUCT (structure) can be compared. Example: `IF (stStruct1 := stStruct2) THEN....`

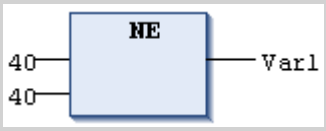
Examples

Result in `Var1` is `FALSE`

ST:

```
Var1 := 40 <> 40;
```

FBD:



Operator 'ADR'

The operator is an extension of the IEC 61131-3 standard.

`ADR` yields the 32-bit address (or the 64-bit address, if possible) of its argument. You can pass this address to the manufacturer functions or assign them to a pointer in the project.

Syntax

```
VAR
    <address name> : DWORD | LWORD | POINTER TO < basis data type>
END_VAR

<address name> := ADR( <variable name> );
```

Example

```
FUNCTION_BLOCK FB_Address
VAR
    piAddress1: POINTER TO INT;
    iNumber1: INT := 5;
    lwAddress2
    iNumber2: INT := 10;
END_VAR

piAddress1 := ADR(iNumber1); // piNumber is assigned to address of
iNumber1
lwAddress2 := ADR(iNumber2); // 64 bit runtime system
```

NOTICE!

In contrast to CoDeSys V2.3, you can use the `ADR` operator with function names, program names, function block names, and method names. Therefore, `ADR` replaces the `INDEXOF` operator.

When using function pointers, note that you can pass a function pointer to external libraries, but it is not possible to call a function pointer from within CODESYS. To enable a system call (runtime system), you must set the respective object property (*"Build"* tab) for the function object.

CAUTION!
When you use an online change, the contents of addresses can shift. As a result, `POINTER TO` variables could point to an invalid memory area. To avoid problems, you should make sure that the value of pointers is updated in every cycle.

CAUTION!
Do not return `Pointer-TO` variables of functions and methods to the caller or assign them to global variables.

See also

- *Chapter 1.4.1.19.5.12 “Pointers” on page 656*

Operator 'Content Operator'

This operator is an extension of the IEC 61131-3 standard.

You can use this operator to dereference pointers by appending the operator as `^` to the pointer identifier.

CAUTION!
When using pointers to addresses, please note that applying an online change can shift address contents.

Example

ST:

```
pt : POINTER TO INT;
var_int1 : INT;
var_int2 : INT;
pt := ADR(var_int1);
var_int2 := pt^;
```

Operator 'BITADR'

The operator is an extension of the IEC 61131-3 standard.

`BITADR` yields the bit offset within a segment in a `DWORD`.

NOTICE!
The offset depends on whether the "Byte addressing" option is selected or cleared in the target system settings.

The highest value nibble (4 bits) in this `DWORD` defines the memory range:

Marker `M`: `16#40000000`

Input `I`: `16#80000000`

Output `Q`: `16#C0000000`

CAUTION!
When using pointers to addresses, note that applying an online change can shift the contents of addresses.

Example **ST implementation language:**

```
VAR
    xVar AT %IX2.3 : BOOL;
    dwBitoffset : DWORD;
END_VAR

dwBitoffset := BITADR(xVar); (* If byte addressing = TRUE, result =
16#80000013; if byte addressing = FALSE, result =  16#80000023 *)
```

Operator 'CAL'

This IEC operator is used for calling function blocks.

In IL, `CAL` calls the instance of a function block.

```
CAL <function block> (<input variable1> := <value>,  <input
variableN> := <value>)
```

Example Call of the `Inst` instance of a function block with assignment of the input variables `Par1` and `Par2` with `0` or `TRUE`.

```
CAL Inst(Par1 := 0, Par2 := TRUE);
```

Overloading

NOTICE!
If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

NOTICE!
The rounding logic for borderline cases depends on the target system or the FPU (Floating Point Unit) of the target system. For example, a value of `-1.5` can be converted differently on different controllers.

Catch value ranges overflows across the application to program code-independent from the target system.

The IEC61131-3 specification does not provide for overloaded functions.

If you want to program strictly according to IEC61131-3, then you should use the operators of the syntax `<type> _TO_ <another type>` as described in the following sections.

The rules for typed conversions also apply here for overloading.

The operators convert values into other data types, explicitly specifying only a target data type and no initial data type (data type of the operands) ("overloaded conversion"). Overloading is not part of the IEC 61131-3 specification.

Call syntax

```
<variable name> := <TO operator> ( <operand> );
<operand> = <variable name> | <literal>
```

Operators

```
TO___UXINT
TO___XINT
TO___XWORD
TO_BIT
TO_BYTE
TO_BOOL
TO_DATE
TO_DINT
TO_DT
TO_DWORD
TO_INT
TO_LDATE
TO_LDT
TO_LINT
TO_LREAL
TO_LTIME
TO_LTOD
TO_LWORD
TO_REAL
TO_SINT
TO_STRING
TO_TIME
TO_TOD
TO_UDINT
TO_UINT
TO_ULINT
TO_USINT
TO_WORD
TO_WSTRING
```

Examples

ST implementation language:

```
VAR
    iNumber_1 : INT;
    rNumber_2 : REAL := 123.456;
    iNumber_2 : INT;
    xIsTrue : BOOL;
    sOutputText : STRING;
    sText : STRING := 'Hello World!';
    wsText: WSTRING;
    dateEvent : DATE := D#2019-9-3;
    uiEvent : UINT;
    uxiData : __UXINT;
END_VAR

iNumber_1 := TO_INT(4.22);            (* Result:  4 *)
iNumber_2 := TO_INT(rNumber_2);        (* Result:  123 *)
xIsTrue := TO_BOOL(1);                (* Result: TRUE *)
sOutputText := TO_STRING(342);        (* Result: '342' *)
wsText := TO_WSTRING(sText);        (* Result: "Hello World!" *)
uiEvent := TO_UINT(dateEvent);        (* Result:  44288 *)
uxiData := TO___UXINT(iNumber_2);    (* Result:  123 *)
```

See also

- *"Type conversion operators" on page 545*
- *Chapter 1.4.1.19.3.36 "Boolean Conversion" on page 567*
- *Chapter 1.4.1.19.3.37 "Integer Conversion" on page 572*
- *Chapter 1.4.1.19.3.38 " Floating-Point Number Conversion" on page 584*
- *Chapter 1.4.1.19.3.39 "String Conversion" on page 587*
- *Chapter 1.4.1.19.3.41 "Date and Time Conversion" on page 600*
- *Chapter 1.4.1.19.3.40 "Time Conversion" on page 595*

Boolean Conversion

NOTICE!

String manipulation when converting to `STRING` or `WSTRING`

When converting the type to `STRING` or `WSTRING`, the typed value is left-aligned as a character string and truncated if it is too long. Therefore, declare the return variable for the type conversion operators `<>_TO_STRING` and `<>_TO_WSTRING` long enough that the character string has enough space without any manipulation.

The operators convert a Boolean value into the specified data types and return a type-converted value.

Call syntax

```
<variable name> := <BOOL to operator> ( <operand> );
<operand> = <variable name> | <literal>
```

Operators

```
BOOL_TO___UXINT
BOOL_TO___XINT
BOOL_TO___XWORD
BOOL_TO_BIT
BOOL_TO_BYTE
BOOL_TO_DATE
BOOL_TO_DINT
BOOL_TO_DT
BOOL_TO_DWORD
BOOL_TO_INT
BOOL_TO_LDATE
BOOL_TO_LDT
BOOL_TO_LINT
BOOL_TO_LREAL
BOOL_TO_LTIME
BOOL_TO_LTOD
BOOL_TO_LWORD
BOOL_TO_REAL
BOOL_TO_SINT
BOOL_TO_STRING
BOOL_TO_TIME
BOOL_TO_TOD
BOOL_TO_UDINT
BOOL_TO_UINT
BOOL_TO_ULINT
BOOL_TO_USINT
BOOL_TO_WORD
BOOL_TO_WSTRING
```

When the operand value is `TRUE`, the following typed values are returned:

- `BOOL_TO_DATE: D#1970-1-1 // The zeroth bit is set, but does not effect the display.`
- `BOOL_TO_DT: DT#1970-01-01-0:0:1`
- `BOOL_TO_LTIME: LTIME#1NS`
- `BOOL_TO_REAL: '1'`
- `BOOL_TO_STRING: 'TRUE'`
- `BOOL_TO_TOD: TOD#0:0:0.001`
- `BOOL_TO_TIME: T#1MS`
- `BOOL_TO_WSTRING: "TRUE"`

When the operand value is `FALSE`, the following typed values are returned:

- `BOOL_TO_DATE: D#1970-1-1`
- `BOOL_TO_DT: DT#1970-01-01-00:00:00`
- `BOOL_TO_LTIME: LTIME#0NS`
- `BOOL_TO_REAL: '0.0'`
- `BOOL_TO_STRING: 'FALSE'`
- `BOOL_TO_TOD: TOD#0:0:0`
- `BOOL_TO_TIME: T#0MS`
- `BOOL_TO_WSTRING: "FALSE"`

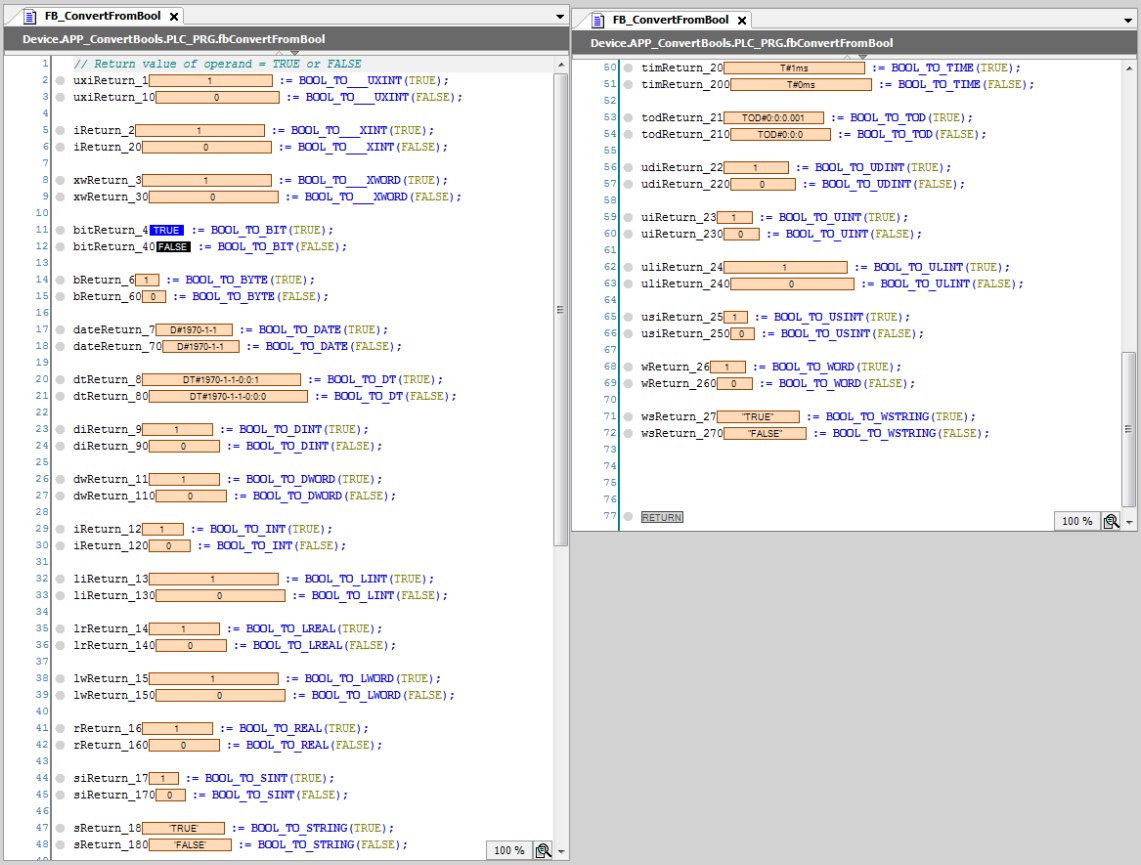
Examples

ST implementation language

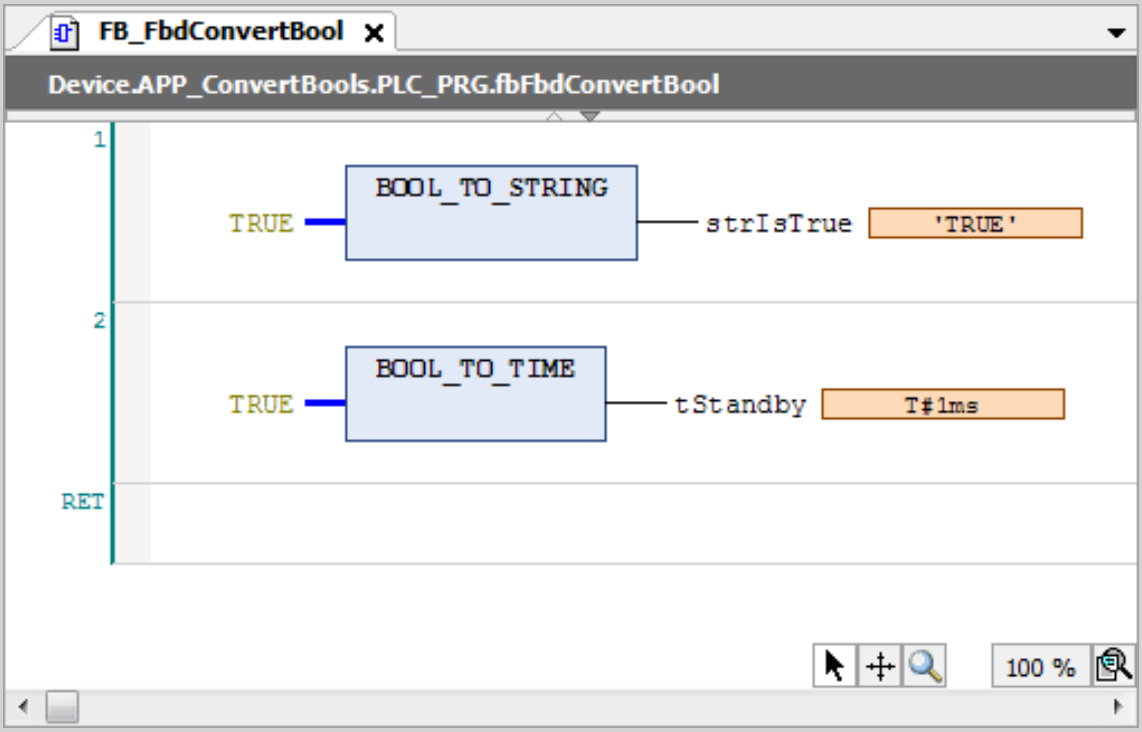
```
FUNCTION_BLOCK FB_ConvertFromBool
VAR
VAR
    uxiReturn_1: __UXINT;
    uxiReturn_10: __UXINT;
    iReturn_2: __XINT;
    iReturn_20: __XINT;
    xwReturn_3: __XWORD;
    xwReturn_30: __XWORD;
    bitReturn_4: BOOL;
    bitReturn_40: BOOL;
    bReturn_6: BYTE;
    bReturn_60: BYTE;
    dateReturn_7: DATE;
    dateReturn_70: DATE;
    dtReturn_8: DATE_AND_TIME;
    dtReturn_80: DATE_AND_TIME;
    diReturn_9: DINT;
    diReturn_90: DINT;
    dtReturn_10: DATE_AND_TIME;
    dtReturn_100: DATE_AND_TIME;
    dwReturn_11: DWORD;
    dwReturn_110: DWORD;
    iReturn_12: INT;
    iReturn_120: INT;
    liReturn_13: LINT;
    liReturn_130: LINT;
    lrReturn_14: LREAL;
    lrReturn_140: LREAL;
    lwReturn_15: LWORD;
    lwReturn_150: LWORD;
    rReturn_16: REAL;
    rReturn_160: REAL;
    siReturn_17: SINT;
    siReturn_170: SINT;
    sReturn_18: STRING;
    sReturn_180: STRING;
    todReturn_19: TIME_OF_DAY;
    todReturn_190: TIME_OF_DAY;
    timReturn_20: TIME;
    timReturn_200: TIME;
    todReturn_21: TIME_OF_DAY;
    todReturn_210: TIME_OF_DAY;
    udiReturn_22: UDINT;
    udiReturn_220: UDINT;
    uiReturn_23: UINT;
    uiReturn_230: UINT;
    uliReturn_24: ULINT;
    uliReturn_240: ULINT;
    usiReturn_25: USINT;
    usiReturn_250: USINT;
    wReturn_26: WORD;
    wReturn_260: WORD;
    wsReturn_27: WSTRING;
    wsReturn_270: WSTRING;
END_VAR

// Return value of operand = TRUE or FALSE
uxiReturn_1 := BOOL_TO___UXINT(TRUE);
```

```
            uxiReturn_10 := BOOL_TO___UXINT(FALSE);

            iReturn_2 := BOOL_TO___XINT(TRUE);
            iReturn_20 := BOOL_TO___XINT(FALSE);

            xwReturn_3 := BOOL_TO___XWORD(TRUE);
            xwReturn_30 := BOOL_TO___XWORD(FALSE);

            bitReturn_4 := BOOL_TO_BIT(TRUE);
            bitReturn_40 := BOOL_TO_BIT(FALSE);

            bReturn_6 := BOOL_TO_BYTE(TRUE);
            bReturn_60 := BOOL_TO_BYTE(FALSE);

            dateReturn_7 := BOOL_TO_DATE(TRUE);
            dateReturn_70 := BOOL_TO_DATE(FALSE);

            dtReturn_8 := BOOL_TO_DT(TRUE);
            dtReturn_80 := BOOL_TO_DT(FALSE);

            diReturn_9 := BOOL_TO_DINT(TRUE);
            diReturn_90 := BOOL_TO_DINT(FALSE);

            dwReturn_11 := BOOL_TO_DWORD(TRUE);
            dwReturn_110 := BOOL_TO_DWORD(FALSE);

            iReturn_12 := BOOL_TO_INT(TRUE);
            iReturn_120 := BOOL_TO_INT(FALSE);

            liReturn_13 := BOOL_TO_LINT(TRUE);
            liReturn_130 := BOOL_TO_LINT(FALSE);

            lrReturn_14 := BOOL_TO_LREAL(TRUE);
            lrReturn_140 := BOOL_TO_LREAL(FALSE);

            lwReturn_15 := BOOL_TO_LWORD(TRUE);
            lwReturn_150 := BOOL_TO_LWORD(FALSE);

            rReturn_16 := BOOL_TO_REAL(TRUE);
            rReturn_160 := BOOL_TO_REAL(FALSE);

            siReturn_17 := BOOL_TO_SINT(TRUE);
            siReturn_170 := BOOL_TO_SINT(FALSE);

            sReturn_18 := BOOL_TO_STRING(TRUE);
            sReturn_180 := BOOL_TO_STRING(FALSE);

            timReturn_20 := BOOL_TO_TIME(TRUE);
            timReturn_200 := BOOL_TO_TIME(FALSE);

            todReturn_21 := BOOL_TO_TOD(TRUE);
            todReturn_210 := BOOL_TO_TOD(FALSE);

            udiReturn_22 := BOOL_TO_UDINT(TRUE);
            udiReturn_220 := BOOL_TO_UDINT(FALSE);

            uiReturn_23 := BOOL_TO_UINT(TRUE);
            uiReturn_230 := BOOL_TO_UINT(FALSE);

            uliReturn_24 := BOOL_TO_ULINT(TRUE);
            uliReturn_240 := BOOL_TO_ULINT(FALSE);

            usiReturn_25 := BOOL_TO_USINT(TRUE);
            usiReturn_250 := BOOL_TO_USINT(FALSE);
```

```
wReturn_26 := BOOL_TO_WORD(TRUE);
wReturn_260 := BOOL_TO_WORD(FALSE);

wsReturn_27 := BOOL_TO_WSTRING(TRUE);
wsReturn_270 := BOOL_TO_WSTRING(FALSE);
```



FBD implementation language



See also

- *“Type conversion operators” on page 545*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*
- *Chapter 1.4.1.19.3.37 “Integer Conversion” on page 572*
- *Chapter 1.4.1.19.3.38 “ Floating-Point Number Conversion” on page 584*
- *Chapter 1.4.1.19.3.39 “String Conversion” on page 587*
- *Chapter 1.4.1.19.3.41 “Date and Time Conversion” on page 600*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*

Integer Conversion

NOTICE!

If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

The operators convert an integer value into the specified data types and return this type-converted value. If the number to be converted exceeds the range limit, then the first bytes of the number are ignored.

Call syntax

```
<variable name> := <integer conversion type operator> ( <integer
operand> );

<integer conversion type operator> = <integer data type> _TO_ <data
type>
<integer operand> = <variable name> | <literal>

<integer data type> =
__UXINT | __XINT | __XWORD | BIT | BYTE | DINT | DWORD | INT | LINT |
LWORD | SINT | UDINT | UINT | ULINT | USINT | WORD
```

Operators

```
__UXINT_TO___XINT
__UXINT_TO___XWORD
__UXINT_TO_BIT
__UXINT_TO_BOOL
__UXINT_TO_BYTE
__UXINT_TO_DATE
__UXINT_TO_DINT
__UXINT_TO_DT
__UXINT_TO_DWORD
__UXINT_TO_INT
__UXINT_TO_LDATE
__UXINT_TO_LDT
__UXINT_TO_LINT
__UXINT_TO_LREAL
__UXINT_TO_LTIME
__UXINT_TO_LTOD
__UXINT_TO_LWORD
__UXINT_TO_REAL
__UXINT_TO_SINT
__UXINT_TO_STRING
```

```
__UXINT_TO_TIME
__UXINT_TO_TOD
__UXINT_TO_UDINT
__UXINT_TO_UINT
__UXINT_TO_ULINT
__UXINT_TO_USINT
__UXINT_TO_WORD
__UXINT_TO_WSTRING

__XINT_TO___UXINT
__XINT_TO___XWORD
__XINT_TO_BIT
__XINT_TO_BOOL
__XINT_TO_BYTE
__XINT_TO_DATE
__XINT_TO_DINT
__XINT_TO_DT
__XINT_TO_DWORD
__XINT_TO_INT
__XINT_TO_LDATE
__XINT_TO_LDT
__XINT_TO_LINT
__XINT_TO_LREAL
__XINT_TO_LTIME
__XINT_TO_LTOD
__XINT_TO_LWORD
__XINT_TO_REAL
__XINT_TO_SINT
__XINT_TO_STRING
__XINT_TO_TIME
__XINT_TO_TOD
__XINT_TO_UDINT
__XINT_TO_UINT
__XINT_TO_ULINT
__XINT_TO_USINT
__XINT_TO_WORD
__XINT_TO_WSTRING

__XWORD_TO_UXINT
__XWORD_TO_XINT
__XWORD_TO_BIT
__XWORD_TO_BOOL
__XWORD_TO_BYTE
__XWORD_TO_DATE
__XWORD_TO_DINT
__XWORD_TO_DT
__XWORD_TO_DWORD
__XWORD_TO_INT
__XWORD_TO_LDATE
__XWORD_TO_LDT
__XWORD_TO_LINT
__XWORD_TO_LREAL
__XWORD_TO_LTIME
__XWORD_TO_LTOD
__XWORD_TO_LWORD
__XWORD_TO_REAL
__XWORD_TO_SINT
__XWORD_TO_STRING
__XWORD_TO_TIME
__XWORD_TO_TOD
__XWORD_TO_UDINT
__XWORD_TO_UINT
__XWORD_TO_ULINT
__XWORD_TO_USINT
__XWORD_TO_WORD
```

```
__XWORD_TO_WSTRING

BIT_TO___UXINT
BIT_TO___XINT
BIT_TO___XWORD
BIT_TO_BOOL
BIT_TO_BYTE
BIT_TO_DATE
BIT_TO_DINT
BIT_TO_DT
BIT_TO_DWORD
BIT_TO_INT
BIT_TO_LDATE
BIT_TO_LDT
BIT_TO_LINT
BIT_TO_LREAL
BIT_TO_LTIME
BIT_TO_LTOD
BIT_TO_LWORD
BIT_TO_REAL
BIT_TO_SINT
BIT_TO_STRING
BIT_TO_TIME
BIT_TO_TOD
BIT_TO_UDINT
BIT_TO_UINT
BIT_TO_ULINT
BIT_TO_USINT
BIT_TO_WORD
BIT_TO_WSTRING

BYTE_TO___UXINT
BYTE_TO___XINT
BYTE_TO___XWORD
BYTE_TO_BOOL
BYTE_TO_BIT
BYTE_TO_DATE
BYTE_TO_DINT
BYTE_TO_DT
BYTE_TO_DWORD
BYTE_TO_INT
BYTE_TO_LDATE
BYTE_TO_LDT
BYTE_TO_LINT
BYTE_TO_LREAL
BYTE_TO_LTIME
BYTE_TO_LTOD
BYTE_TO_LWORD
BYTE_TO_REAL
BYTE_TO_SINT
BYTE_TO_STRING
BYTE_TO_TIME
BYTE_TO_TOD
BYTE_TO_UDINT
BYTE_TO_UINT
BYTE_TO_ULINT
BYTE_TO_USINT
BYTE_TO_WORD
BYTE_TO_WSTRING

DINT_TO___UXINT
DINT_TO___XINT
DINT_TO___XWORD
DINT_TO_BOOL
DINT_TO_BIT
```

```
DINT_TO_BYTE
DINT_TO_DATE
DINT_TO_DT
DINT_TO_DWORD
DINT_TO_INT
DINT_TO_LDATE
DINT_TO_LDT
DINT_TO_LINT
DINT_TO_LREAL
DINT_TO_LTIME
DINT_TO_LTOD
DINT_TO_LWORD
DINT_TO_REAL
DINT_TO_SINT
DINT_TO_STRING
DINT_TO_TIME
DINT_TO_TOD
DINT_TO_UDINT
DINT_TO_UINT
DINT_TO_ULINT
DINT_TO_USINT
DINT_TO_WORD
DINT_TO_WSTRING

DWORD_TO___UXINT
DWORD_TO___XINT
DWORD_TO___XWORD
DWORD_TO_BIT
DWORD_TO_BOOL
DWORD_TO_BYTE
DWORD_TO_DATE
DWORD_TO_DINT
DWORD_TO_DT
DWORD_TO_INT
DWORD_TO_LDATE
DWORD_TO_LDT
DWORD_TO_LINT
DWORD_TO_LREAL
DWORD_TO_LTIME
DWORD_TO_LTOD
DWORD_TO_LWORD
DWORD_TO_REAL
DWORD_TO_SINT
DWORD_TO_STRING
DWORD_TO_TIME
DWORD_TO_TOD
DWORD_TO_UDINT
DWORD_TO_UINT
DWORD_TO_ULINT
DWORD_TO_USINT
DWORD_TO_WORD
DWORD_TO_WSTRING

INT_TO___UXINT
INT_TO___XINT
INT_TO___XWORD
INT_TO_BIT
INT_TO_BOOL
INT_TO_BYTE
INT_TO_DATE
INT_TO_DINT
INT_TO_DT
INT_TO_DWORD
INT_TO_LDATE
INT_TO_LDT
```

```
INT_TO_LINT
INT_TO_LREAL
INT_TO_LTIME
INT_TO_LTOD
INT_TO_LWORD
INT_TO_REAL
INT_TO_SINT
INT_TO_STRING
INT_TO_TIME
INT_TO_TOD
INT_TO_UDINT
INT_TO_UINT
INT_TO_ULINT
INT_TO_USINT
INT_TO_WORD
INT_TO_WSTRING

LINT_TO___UXINT
LINT_TO___XINT
LINT_TO___XWORD
LINT_TO_BIT
LINT_TO_BOOL
LINT_TO_BYTE
LINT_TO_DATE
LINT_TO_DINT
LINT_TO_DT
LINT_TO_DWORD
LINT_TO_INT
LINT_TO_LDATE
LINT_TO_LDT
LINT_TO_LREAL
LINT_TO_LTIME
LINT_TO_LTOD
LINT_TO_LWORD
LINT_TO_REAL
LINT_TO_SINT
LINT_TO_STRING
LINT_TO_TIME
LINT_TO_TOD
LINT_TO_UDINT
LINT_TO_UINT
LINT_TO_ULINT
LINT_TO_USINT
LINT_TO_WORD
LINT_TO_WSTRING

LWORD_TO___UXINT
LWORD_TO___XINT
LWORD_TO___XWORD
LWORD_TO_BIT
LWORD_TO_BOOL
LWORD_TO_BYTE
LWORD_TO_DATE
LWORD_TO_DINT
LWORD_TO_DT
LWORD_TO_DWORD
LWORD_TO_INT
LWORD_TO_LDATE
LWORD_TO_LDT
LWORD_TO_LINT
LWORD_TO_LREAL
LWORD_TO_LTIME
LWORD_TO_LTOD
LWORD_TO_REAL
LWORD_TO_SINT
```

```
LWORD_TO_STRING
LWORD_TO_TIME
LWORD_TO_TOD
LWORD_TO_UDINT
LWORD_TO_UINT
LWORD_TO_ULINT
LWORD_TO_USINT
LWORD_TO_WORD
LWORD_TO_WSTRING

SINT_TO___UXINT
SINT_TO___XINT
SINT_TO___XWORD
SINT_TO_BIT
SINT_TO_BOOL
SINT_TO_BYTE
SINT_TO_DATE
SINT_TO_DINT
SINT_TO_DT
SINT_TO_DWORD
SINT_TO_INT
SINT_TO_LDATE
SINT_TO_LDT
SINT_TO_LINT
SINT_TO_LREAL
SINT_TO_LTIME
SINT_TO_LTOD
SINT_TO_LWORD
SINT_TO_REAL
SINT_TO_STRING
SINT_TO_TIME
SINT_TO_TOD
SINT_TO_UDINT
SINT_TO_UINT
SINT_TO_ULINT
SINT_TO_USINT
SINT_TO_WORD
SINT_TO_WSTRING

UDINT_TO___UXINT
UDINT_TO___XINT
UDINT_TO___XWORD
UDINT_TO_BIT
UDINT_TO_BOOL
UDINT_TO_BYTE
UDINT_TO_DATE
UDINT_TO_DINT
UDINT_TO_DT
UDINT_TO_DWORD
UDINT_TO_INT
UDINT_TO_LDATE
UDINT_TO_LDT
UDINT_TO_LINT
UDINT_TO_LREAL
UDINT_TO_LTIME
UDINT_TO_LTOD
UDINT_TO_LWORD
UDINT_TO_REAL
UDINT_TO_SINT
UDINT_TO_STRING
UDINT_TO_TIME
UDINT_TO_TOD
UDINT_TO_UINT
UDINT_TO_ULINT
UDINT_TO_USINT
```

```
UDINT_TO_WORD
UDINT_TO_WSTRING

UINT_TO___UXINT
UINT_TO___XINT
UINT_TO___XWORD
UINT_TO_BIT
UINT_TO_BOOL
UINT_TO_BYTE
UINT_TO_DATE
UINT_TO_DINT
UINT_TO_DT
UINT_TO_DWORD
UINT_TO_INT
UINT_TO_LDATE
UINT_TO_LDT
UINT_TO_LINT
UINT_TO_LREAL
UINT_TO_LTIME
UINT_TO_LTOD
UINT_TO_LWORD
UINT_TO_REAL
UINT_TO_SINT
UINT_TO_STRING
UINT_TO_TIME
UINT_TO_TOD
UINT_TO_UDINT
UINT_TO_ULINT
UINT_TO_USINT
UINT_TO_WORD
UINT_TO_WSTRING

ULINT_TO___UXINT
ULINT_TO___XINT
ULINT_TO___XWORD
ULINT_TO_BIT
ULINT_TO_BOOL
ULINT_TO_BYTE
ULINT_TO_DATE
ULINT_TO_DINT
ULINT_TO_DT
ULINT_TO_DWORD
ULINT_TO_INT
ULINT_TO_LDATE
ULINT_TO_LDT
ULINT_TO_LINT
ULINT_TO_LREAL
ULINT_TO_LTIME
ULINT_TO_LTOD
ULINT_TO_LWORD
ULINT_TO_REAL
ULINT_TO_SINT
ULINT_TO_STRING
ULINT_TO_TIME
ULINT_TO_TOD
ULINT_TO_UDINT
ULINT_TO_UINT
ULINT_TO_USINT
ULINT_TO_WORD
ULINT_TO_WSTRING

USINT_TO___XINT
USINT_TO___XINT
USINT_TO___XWORD
USINT_TO_BIT
```

```
USINT_TO_BOOL
USINT_TO_BYTE
USINT_TO_DATE
USINT_TO_DINT
USINT_TO_DT
USINT_TO_DWORD
USINT_TO_INT
USINT_TO_LDATE
USINT_TO_LDT
USINT_TO_LINT
USINT_TO_LREAL
USINT_TO_LTIME
USINT_TO_LTOD
USINT_TO_LWORD
USINT_TO_REAL
USINT_TO_SINT
USINT_TO_STRING
USINT_TO_TIME
USINT_TO_TOD
USINT_TO_UDINT
USINT_TO_UINT
USINT_TO_ULINT
USINT_TO_WORD
USINT_TO_WSTRING

WORD_TO___XINT
WORD_TO___XINT
WORD_TO___XWORD
WORD_TO_BIT
WORD_TO_BOOL
WORD_TO_BYTE
WORD_TO_DATE
WORD_TO_DINT
WORD_TO_DT
WORD_TO_DWORD
WORD_TO_INT
WORD_TO_LDATE
WORD_TO_LDT
WORD_TO_LINT
WORD_TO_LREAL
WORD_TO_LTIME
WORD_TO_LTOD
WORD_TO_LWORD
WORD_TO_REAL
WORD_TO_SINT
WORD_TO_STRING
WORD_TO_TIME
WORD_TO_TOD
WORD_TO_UDINT
WORD_TO_UINT
WORD_TO_ULINT
WORD_TO_USINT
WORD_TO_WSTRING
```

Converting to a string

NOTICE!

String manipulation when converting to `STRING` or `WSTRING`

When converting the type to `STRING` or `WSTRING`, the typed value is left-aligned as a character string and truncated if it is too long. Therefore, declare the return variable for the type conversion operators `<>_TO_STRING` and `<>_TO_WSTRING` long enough that the character string has enough space without any manipulation.

The operators that convert a value into a character string of type `STRING` or `WSTRING` require an operand that matches the target data type.

Example

FB_ConvertToWstrings

Device.App_ConvertStrings.PLC_PRG.fbConvertToWstrings

```
1  wstrReturn "255" := BYTE_TO_WSTRING(2#1111_1111);
2  wstrReturn_1 "D#1970-01- := DATE_TO_WSTRING(D#1970-1-1);
3  wstrReturn_2 "D#1970-01- := DATE_TO_WSTRING(D#1970-1-2);
4  wstrReturn_3 "D#1970-01- := DATE_TO_WSTRING(D#1970-1-1);
5  wstrReturn_4 "D#1970-01- := DATE_TO_WSTRING(DATE#1970-1-1);
6  wstrReturn_5 "DT#1970-01 := DT_TO_WSTRING(DT#1970-1-2-0:0:0);
7  wstrReturn_6 "65535" := DINT_TO_WSTRING(65535);
8  wstrReturn_7 "4294967295" := DWORD_TO_WSTRING(16#FFFF_FFFF);
9  wstrReturn_8 "-3276" := INT_TO_WSTRING(-3276);
10 wstrReturn_9 "0" := LINT_TO_WSTRING(0);
11 wstrReturn_10 "1.0e309" := LREAL_TO_WSTRING(1.7E+308);
12 wstrReturn_11 "1844674407 := LWORD_TO_WSTRING(16#FFFF_FFFF_FFFF_FFFF);
13 wstrReturn_12 "1.234" := REAL_TO_WSTRING(1.234);
14 wstrReturn_13 "127" := SINT_TO_WSTRING(127);
15 wstrReturn_14 "Hello" := STRING_TO_WSTRING('Hello');
16 wstrReturn_15 "TOD#01:01: := TOD_TO_WSTRING(TOD#1:1:1);
17 wstrReturn_16 "T#5d" := TIME_TO_WSTRING(T#5D);
18 wstrReturn_17 "4294967295" := UDINT_TO_WSTRING(4294967295);
19 wstrReturn_18 "65535" := UINT_TO_WSTRING(65535);
20 wstrReturn_19 "1" := ULINT_TO_WSTRING(1);
21 wstrReturn_20 "255" := USINT_TO_WSTRING(255);
22 wstrReturn_21 "65535" := WORD_TO_WSTRING(16#FFFF);
23
24 RETURN
```

100 %

Examples

ST implementation language

When a larger data type is converted to a smaller data type, the more high-order (front) bytes are truncated. When a smaller data type is converted to a larger data type, the more high-order bytes filled with zeros.

```
FUNCTION_BLOCK FB_ConvertIntegersFromInt
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    uxiReturn: __UXINT;
    xiReturn: __XINT;
    xwReturn: __XWORD;
    bitReturn: BIT;
    xReturn: BOOL;
    bReturn: BYTE;
    dateReturn: DATE;
    diReturn: DINT;
    dtReturn: DATE_AND_TIME;
    dwReturn: DWORD;
    liReturn: LINT;
    lrReturn: LREAL;
    lwReturn: LWORD;
    siReturn: SINT;
    sReturn: STRING;
    timReturn: TIME;
    todReturn: TIME_OF_DAY;
    udiReturn: UDINT;
    uiReturn: UINT;
    usiReturn: USINT;
    wReturn: WORD;
    wsReturn: WSTRING;
    uliReturn: ULINT;
END_VAR

uxiReturn := INT_TO___UXINT(127);
xiReturn := INT_TO___XINT(127);
xwReturn := INT_TO___XWORD(127);
bitReturn := INT_TO_BIT(127);
xReturn := INT_TO_BOOL(127);
bReturn := INT_TO_BYTE(127);
dateReturn := INT_TO_DATE(127);
diReturn := INT_TO_DINT(127);
dtReturn := INT_TO_DT(127);
dwReturn := INT_TO_DWORD(127);
liReturn := INT_TO_LINT(127);
lrReturn := INT_TO_LREAL(127);
lwReturn := INT_TO_LWORD(127);
siReturn := INT_TO_SINT(127);
sReturn := INT_TO_STRING(127);
timReturn := INT_TO_TIME(127);
todReturn := INT_TO_TOD(127);
udiReturn := INT_TO_UDINT(127);
uiReturn := INT_TO_UINT(127);
uliReturn := INT_TO_ULINT(127);
usiReturn := INT_TO_USINT(127);
wReturn := INT_TO_WORD(127);
wsReturn := INT_TO_WSTRING(127);

FUNCTION_BLOCK FB_ConvertIntegersToInt
VAR_INPUT
END_VAR
VAR_OUTPUT
```

```
END_VAR
VAR
    iReturn_uxi: INT;
    iReturn_xi: INT;
    iReturn_xw: INT;
    iReturn_bit: INT;
    iReturn_bool: INT;
    iReturn_b: INT;
    iReturn_d: INT;
    iReturn_di: INT;
    iReturn_dt: INT;
    iReturn_dw: INT;
    iReturn_li: INT;
    iReturn_lr: INT;
    iReturn_lw: INT;
    iReturn_r: INT;
    iReturn_si: INT;
    iReturn_s: INT;
    iReturn_tim: INT;
    iReturn_tod: INT;
    iReturn_tod_0: INT;
    iReturn_udi: INT;
    iReturn_ui: INT;
    iReturn_uli: INT;
    iReturn_usi: INT;
    iReturn_w: INT;
    iReturn_ws: INT;
END_VAR

iReturn_uxi := __UXINT_TO_INT(18446744073709551615);
iReturn_xi := __XINT_TO_INT(9223372036854775807);
iReturn_xw := __XWORD_TO_INT(16#FFFF_FFFF_FFFF_FFFF);
iReturn_bit := BIT_TO_INT(1);
iReturn_bool := BOOL_TO_INT(TRUE);
iReturn_b := BYTE_TO_INT(2#1111_0000);
iReturn_d := DATE_TO_INT(DATE#2019-9-13);
iReturn_di := DINT_TO_INT(2147483647);
iReturn_dt := DT_TO_INT(DT#1979-1-1-00:00:00);
iReturn_dw := DWORD_TO_INT(16#FFFF_FFFF);
// iReturn_i := INT_TO_<>(iData_12);
iReturn_li := LINT_TO_INT(9223372036854775807);
iReturn_lr := LREAL_TO_INT(1.7976931348623157E+30);
iReturn_lw := LWORD_TO_INT(16#FFFF_FFFF_FFFF_FFFF);
iReturn_r := REAL_TO_INT(3.402823E+38);
iReturn_si := SINT_TO_INT(127);
iReturn_s := STRING_TO_INT('127');
iReturn_tim := TIME_TO_INT(T#49D17H2M47S295MS);
iReturn_tod := TOD_TO_INT(TOD#23:59:59.999);
iReturn_tod_0 := TOD_TO_INT(TOD#1:1:1.001);
iReturn_udi := UDINT_TO_INT(4294967295);
iReturn_ui := UINT_TO_INT(65535);
iReturn_uli := ULINT_TO_INT(18446744073709551615);
iReturn_usi := USINT_TO_INT(255);
iReturn_w := WORD_TO_INT(16#FFFF);
iReturn_ws := WSTRING_TO_INT("1234567890");

PROGRAM PLC_PRG
VAR
    fbConvertIntegersFromInt : FB_ConvertIntegersFromInt;
    fbConvertIntegersToInt : FB_ConvertIntegersToInt;
END_VAR

fbConvertIntegersFromInt();
fbConvertIntegersToInt();
```

FB_ConvertIntegersFromInt

Device.App_ConvertIntegers.PLC_PRG.fbConvertIntegersFromInt

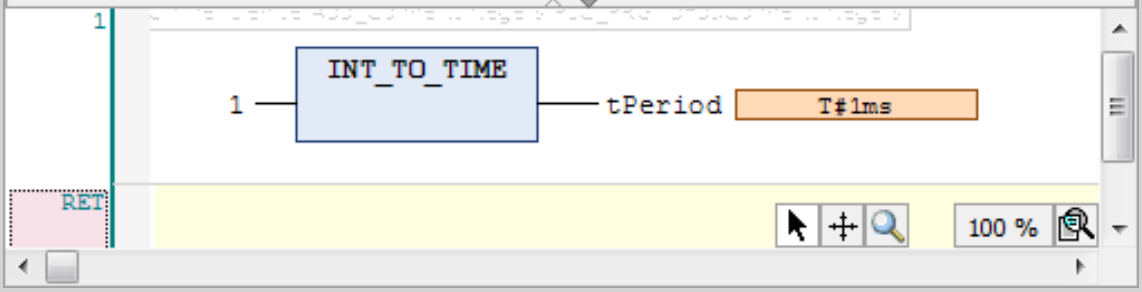
```
1  uxiReturn  16#000000000000007F  := INT_TO___UXINT(127);
2  xiReturn   16#000000000000007F  := INT_TO___XINT(127);
3  xwReturn   16#000000000000007F  := INT_TO___XWORD(127);
4  bitReturn  TRUE                 := INT_TO_BIT(127);
5  xReturn    TRUE                 := INT_TO_BOOL(127);
6  bReturn    16#7F                := INT_TO_BYTE(127);
7  dateReturn D#1970-1-1           := INT_TO_DATE(127);
8  diReturn   16#0000007F          := INT_TO_DINT(127);
9  dtReturn   DT#1970-1-1-0:2:7    := INT_TO_DT(127);
10 dwReturn   16#0000007F          := INT_TO_DWORD(127);
11 liReturn   16#000000000000007F  := INT_TO_LINT(127);
12 lrReturn   127                  := INT_TO_LREAL(127);
13 lwReturn   16#000000000000007F  := INT_TO_LWORD(127);
14 siReturn   16#7F                := INT_TO_SINT(127);
15 sReturn    '127'                := INT_TO_STRING(127);
16 timReturn  T#127ms              := INT_TO_TIME(127);
17 todReturn  TOD#0:0:0.127        := INT_TO_TOD(127);
18 udiReturn  16#0000007F          := INT_TO_UDINT(127);
19 uiReturn   16#007F              := INT_TO_UINT(127);
20 uliReturn  16#000000000000007F  := INT_TO_ULINT(127);
21 usiReturn  16#7F                := INT_TO_USINT(127);
22 wReturn    16#007F              := INT_TO_WORD(127);
23 wsReturn   "127"                := INT_TO_WSTRING(127);
24
25 RETURN
```

100 %

FBD implementation language

FB_FbdConvertIntegers

Device.App_ConvertIntegers.PLC_PRG.fbFbdConvertIntegers



See also

- *“Type conversion operators” on page 545*
- *Chapter 1.4.1.19.3.36 “Boolean Conversion” on page 567*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*

- *Chapter 1.4.1.19.3.38 “ Floating-Point Number Conversion” on page 584*
- *Chapter 1.4.1.19.3.39 “String Conversion” on page 587*
- *Chapter 1.4.1.19.3.41 “Date and Time Conversion” on page 600*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*

Floating-Point Number Conversion

NOTICE!
If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

NOTICE!
If the floating-point number is within the value range of the target data type, then the conversion operates the same way on all systems.

NOTICE!
If the floating-point number to be converted exceeds the range limit, then the first bytes of the number are ignored.

The operators convert a floating-point number into the specified data types and return a type-converted value. If applicable, the conversion is rounded.

Call

Syntax

```
<variable name> := <floating-point conversion operator> ( <floating-
point operand> );

<floating-point operand> = <variable name> | <literal>

<floating-point type> =
REAL |
LREAL
```

Operators

```
REAL_TO___UXINT
REAL_TO___XINT
REAL_TO___XWORD
REAL_TO_BIT
REAL_TO_BOOL
REAL_TO_BYTE
REAL_TO_DATE
REAL_TO_DINT
REAL_TO_DT
REAL_TO_DWORD
REAL_TO_INT
REAL_TO_LINT
REAL_TO_LREAL
REAL_TO_LTIME
REAL_TO_LWORD
REAL_TO_SINT
```

```
REAL_TO_STRING
REAL_TO_TIME
REAL_TO_TOD
REAL_TO_UDINT
REAL_TO_UINT
REAL_TO_ULINT
REAL_TO_USINT
REAL_TO_WORD
REAL_TO_WSTRING

LREAL_TO___UXINT
LREAL_TO___XINT
LREAL_TO___XWORD
LREAL_TO_BIT
LREAL_TO_BOOL
LREAL_TO_BYTE
LREAL_TO_DATE
LREAL_TO_DINT
LREAL_TO_DT
LREAL_TO_DWORD
LREAL_TO_INT
LREAL_TO_LINT
LREAL_TO_LTIME
LREAL_TO_LWORD
LREAL_TO_REAL
LREAL_TO_SINT
LREAL_TO_STRING
LREAL_TO_TIME
LREAL_TO_TOD
LREAL_TO_UDINT
LREAL_TO_UINT
LREAL_TO_ULINT
LREAL_TO_USINT
LREAL_TO_WORD
LREAL_TO_WSTRING
```

Rounding

When converting to an integer, the operand is rounded up or down to an integer value. For 1 to 4 after the decimal point, the number is rounded down. For 5 to 9, the number is rounded up. Then the rounded number is converted to the specified integer type. If the rounded value is outside of the integer value range, then an undefined, target system-dependent value is returned. An exception error is also possible then.

NOTICE!

The rounding logic for borderline cases depends on the target system or the FPU (Floating Point Unit) of the target system. For example, a value of `-1.5` can be converted differently on different controllers.

To program target system-independent code, you have to catch value range overflows across the application.

Converting to a string

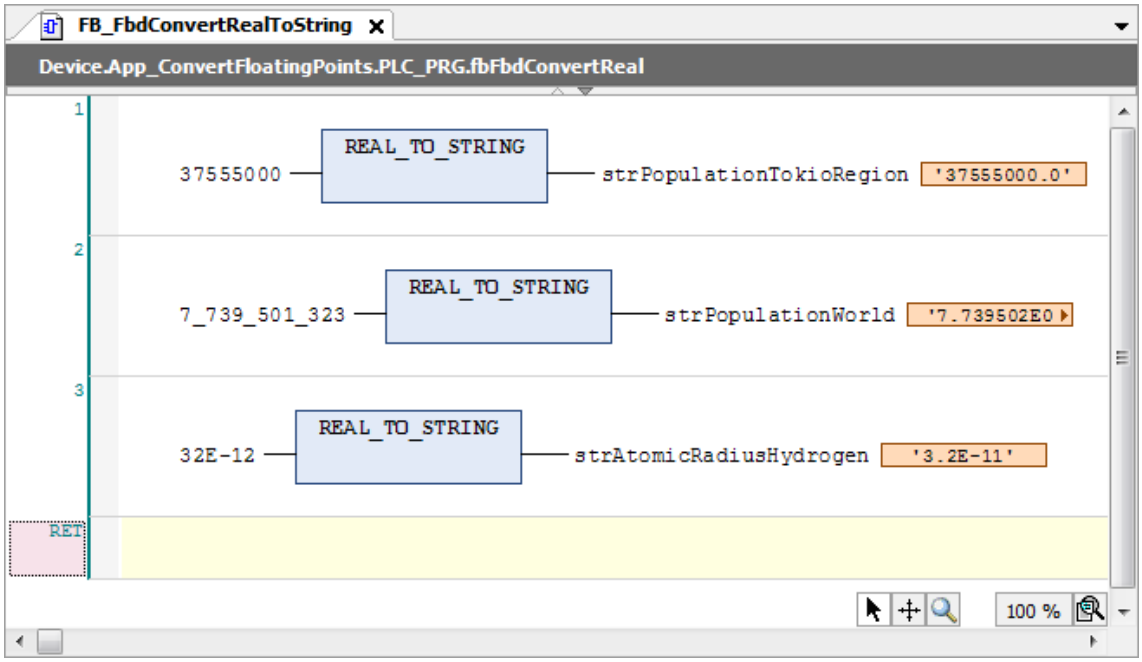
NOTICE!

String manipulation when converting to `STRING` or `WSTRING`

When converting the type to `STRING` or `WSTRING`, the typed value is left-aligned as a character string and truncated if it is too long. Therefore, declare the return variable for the type conversion operators `<>_TO_STRING` and `<>_TO_WSTRING` long enough that the character string has enough space without any manipulation.

For a floating-point number conversion to a string, the number of decimal places of the mantissa is limited to 6. If the number is < 1, then the mantissa is `1 <= m < 10`. If the mantissa has more digits after the comma, then it is rounded to the 6th digit and then converted.

The string variable may also be declared too short for the return value. In this case, the return string is truncated on the right.



Examples

ST implementation language

FB_ConvertFromReal

Device.App_ConvertFloatingPoints.PLC_PRG.fbConvertFromReal

```
3  xwReturn 9223372036854775808 := REAL_TO___XWORD(3.402823E+38);
4  xReturn TRUE := REAL_TO_BOOL(3.402823E+38);
5  bReturn 0 := REAL_TO_BYTE(3.402823E+38);
6  dateReturn D#1970-1-1 := REAL_TO_DATE(3.402823E+38);
7  diReturn 0 := REAL_TO_DINT(3.402823E+38);
8  dtReturn DT#1970-1-1-0:0:0 := REAL_TO_DT(3.402823E+38);
9  dwReturn 0 := REAL_TO_DWORD(3.402823E+38);
10 iReturn -2 := REAL_TO_INT(-1.5);
11 iReturn_0 -1 := REAL_TO_INT(-1.4);
12 iReturn_1 1 := REAL_TO_INT(1.4);
13 iReturn_2 2 := REAL_TO_INT(1.5);
14 iReturn_3 12346 := REAL_TO_INT(1.234567890123456789E4);
15 liReturn -9223372036854775808 := REAL_TO_LINT(3.402823E+38);
16 lrReturn 3.4E+38 := REAL_TO_LREAL(3.402823E+38);
17 lwReturn 9223372036854775808 := REAL_TO_LWORD(3.402823E+38);
18 siReturn 0 := REAL_TO_SINT(3.402823E+38);
19 sReturn '3.402823E3 := REAL_TO_STRING(3.402823E+38);
20 timReturn T#0ms := REAL_TO_TIME(3.402823E+38);
21 todReturn TOD#0:0:0 := REAL_TO_TOD(3.402823E+38);
22 udiReturn 0 := REAL_TO_UDINT(3.402823E+38);
23 uiReturn 0 := REAL_TO_UINT(3.402823E+38);
24 uliReturn 9223372036854775808 := REAL_TO_ULINT(3.402823E+38);
25 usiReturn 0 := REAL_TO_USINT(3.402823E+38);
26 wReturn 0 := REAL_TO_WORD(3.402823E+38);
27 wsReturn "3.402823E3 := REAL_TO_WSTRING(3.402823E+38);
28
```

100 %

See also

- *“Type conversion operators” on page 545*
- *Chapter 1.4.1.19.3.36 “Boolean Conversion” on page 567*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*
- *Chapter 1.4.1.19.3.37 “Integer Conversion” on page 572*
- *Chapter 1.4.1.19.3.39 “String Conversion” on page 587*
- *Chapter 1.4.1.19.3.41 “Date and Time Conversion” on page 600*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*

String Conversion

NOTICE!

If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

The operators convert a character string (`STRING` or `WSTRING`) into the specified target data type and return a type-converted value.

A conversion with a meaningful result is only possible when the operand matches the target data type according to the IEC 61131-3 standard. This is the case if the value of the operand corresponds to a valid constant (literal) of the target data type.

Convertible strings contain:

- Number with type prefix (example: `'16#FFFFFFFF'`)
- Number with grouping characters (example: `'2#1111_1111'`)
  Note: The international weight and measure grouping character (thin space) is not accepted. Only the underscore is accepted.
- Floating-point number, also in exponential notation (example: `'9.876'` or `'1.2E-34'`)
  Note: Floating-point numbers are not convertible. The comma is treated and truncated like a following character.
- Time, time of day, and date specification with prefix and size (example: `'T#2h'`, `'DT#2019-9-9-12:30:30.9'`)
- Infinite values (example: `'1.7E+400'`)
- Additional character **after** a number (example: `'2m'` or `'3.14'`). These are truncated. Additional characters **before** a number are not permitted.
- Spaces before (example: `' 3.14'`)

Call syntax

```
<variable name> := <string to operator> ( <operand> );

<operand> = <variable name> | <literal>
```

Operators

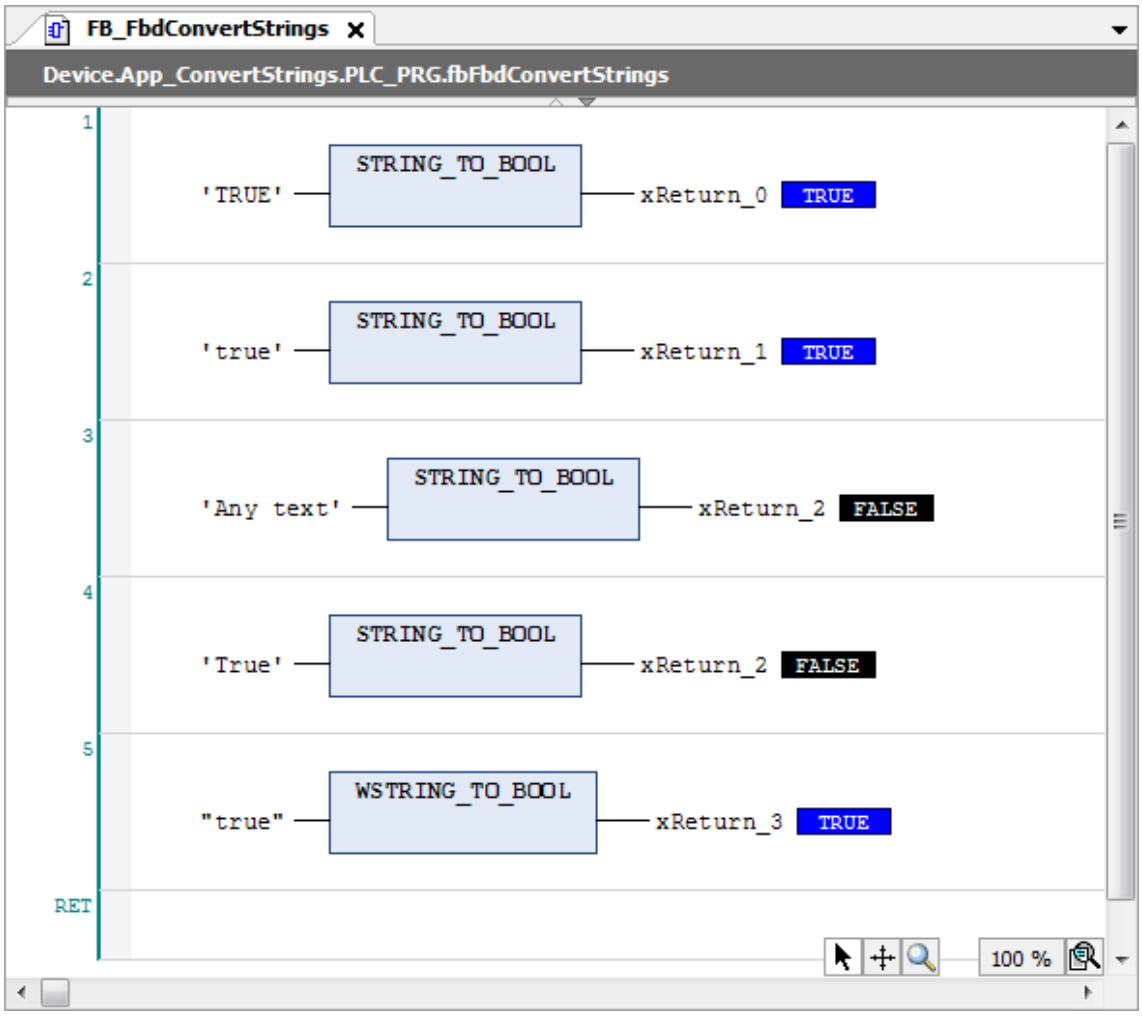
```
STRING_TO___UXINT
STRING_TO___XINT
STRING_TO___XWORD
STRING_TO_BIT
STRING_TO_BOOL
STRING_TO_BYTE
STRING_TO_DATE
STRING_TO_DINT
STRING_TO_DT
STRING_TO_DWORD
STRING_TO_INT
STRING_TO_LDATE
STRING_TO_LDT
STRING_TO_LINT
STRING_TO_LREAL
STRING_TO_LTIME
STRING_TO_LWORD
STRING_TO_LTIME
STRING_TO_LTOD
STRING_TO_REAL
STRING_TO_SINT
STRING_TO_TIME
STRING_TO_TOD
STRING_TO_UDINT
STRING_TO_UINT
STRING_TO_ULINT
STRING_TO_USINT
STRING_TO_WORD
STRING_TO_WSTRING

WSTRING_TO___UXINT
WSTRING_TO___XINT
WSTRING_TO___XWORD
```

```
WSTRING_TO_BIT
WSTRING_TO_BOOL
WSTRING_TO_BYTE
WSTRING_TO_DATE
WSTRING_TO_DINT
WSTRING_TO_DT
WSTRING_TO_DWORD
WSTRING_TO_INT
WSTRING_TO_LDATE
WSTRING_TO_LDT
WSTRING_TO_LINT
WSTRING_TO_LREAL
WSTRING_TO_LTIME
WSTRING_TO_LTOD
WSTRING_TO_LWORD
WSTRING_TO_LTIME
WSTRING_TO_REAL
WSTRING_TO_SINT
WSTRING_TO_STRING
WSTRING_TO_TIME
WSTRING_TO_TOD
WSTRING_TO_UDINT
WSTRING_TO_UINT
WSTRING_TO_ULINT
WSTRING_TO_USINT
WSTRING_TO_STRING
WSTRING_TO_WORD
```

Converting to a Boolean value

Operator `STRING_TO_BOOL`: A value of `TRUE` is returned only if the operand value is `'TRUE'` or `'true'`. On the other hand, `FALSE` is returned for `'True'`.

Operator `WSTRING_TO_BOOL`: A value of `TRUE` is returned only if the operand value is `"TRUE"` or `"true"`. On the other hand, `FALSE` is returned for `"True"`.


FB_FbdConvertStrings
Device.App_ConvertStrings.PLC_PRG.fbFbdConvertStrings
1
'TRUE'
STRING_TO_BOOL
xReturn_0
TRUE
2
'true'
STRING_TO_BOOL
xReturn_1
TRUE
3
'Any text'
STRING_TO_BOOL
xReturn_2
FALSE
4
'True'
STRING_TO_BOOL
xReturn_2
FALSE
5
"true"
WSTRING_TO_BOOL
xReturn_3
TRUE
RET
100 %

Examples

ST implementation language

```
FUNCTION_BLOCK FB_ConvertStrings
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    xReturn_0: BOOL;
    xReturn_1: BOOL;
    dateReturn: DATE;
    dtReturn: DATE_AND_TIME;
    iReturn: INT;
    lrReturn: LREAL;
    lrReturn_0: LREAL;
    lwReturn: LWORD;
    lwReturn_0: LWORD;
    lwReturn_1: LWORD;
    ltReturn: LTIME;
    ltReturn_0: LTIME;
    ltReturn_1: LTIME;
    ltReturn_2: LTIME;
    rReturn: REAL;
    rReturn_0: REAL;
    timReturn: TIME;
    timReturn0: TIME;
    timReturn1: TIME;
    timReturn2: TIME;
    todReturn: TIME_OF_DAY;
    todReturn0: TIME_OF_DAY;
    todReturn1: TIME_OF_DAY;
    todReturn2: TIME_OF_DAY;
    uliReurn: ULINT;
    uliReurn_0: ULINT;
    uliReurn_1: ULINT;
    wReturn: WORD;
    wReturn_0: WORD;
    wReturn_1: WORD;
    wstrReturn: WSTRING;
    wstrReturn_0: WSTRING;
END_VAR
xReturn_0 := STRING_TO_BOOL('FALSE');
xReturn_1 := STRING_TO_BOOL('TRUE');
dateReturn := STRING_TO_DATE('DATE#2019-9-9');
dtReturn := STRING_TO_DT('DT#2019-9-9-1:1:1.1');
iReturn := STRING_TO_INT('123abc');
lrReturn := STRING_TO_LREAL('4.94E-323');
lrReturn_0 := STRING_TO_LREAL('1.7E+308');
lwReturn := STRING_TO_LWORD('16#FFFF_FFFF_FFFF_FFFF');
lwReturn_0 := STRING_TO_LWORD('16#0123456789ABCDEF');
lwReturn_1 := STRING_TO_LWORD('16#0123456789ABCDEF');
ltReturn :=
STRING_TO_LTIME('LTIME#213503d23h34m33s709ms551us615ns');
ltReturn_0 := STRING_TO_LTIME('LTIME#0ns');
ltReturn_1 := STRING_TO_LTIME('LTIME#1ms');
ltReturn_2 := STRING_TO_LTIME('LTIME#2s');
rReturn := STRING_TO_REAL('6.543e21');
rReturn_0 := STRING_TO_REAL('1.234');
timReturn := STRING_TO_TIME('T#5d4h3m2s');
timReturn0 := STRING_TO_TIME('TIME#1s');
timReturn1 := STRING_TO_TIME('1s');
```

```
timReturn2 := STRING_TO_TIME('TIME#5s');
todReturn := STRING_TO_TOD('TOD#12:0:0.1');
todReturn0 := STRING_TO_TOD('TOD#0:0:0.0');
todReturn1 := STRING_TO_TOD('20:15');
todReturn2 := STRING_TO_TOD('TOD#20:15');
uliReurn := STRING_TO_ULINT('18446744073709551615');
uliReurn_0 := STRING_TO_ULINT('1');
uliReurn_1 := STRING_TO_ULINT('0');
wReturn := STRING_TO_WORD('16#FFFF_0000');
wReturn_0 := STRING_TO_WORD('34abc');
wReturn_1 := STRING_TO_WORD('16#34abc');
wstrReturn := STRING_TO_WSTRING('Hello World!');
wstrReturn_0 := STRING_TO_WSTRING('123456789');
```

Device.App_ConvertStrings.PLC_PRG.fbConvertStrings

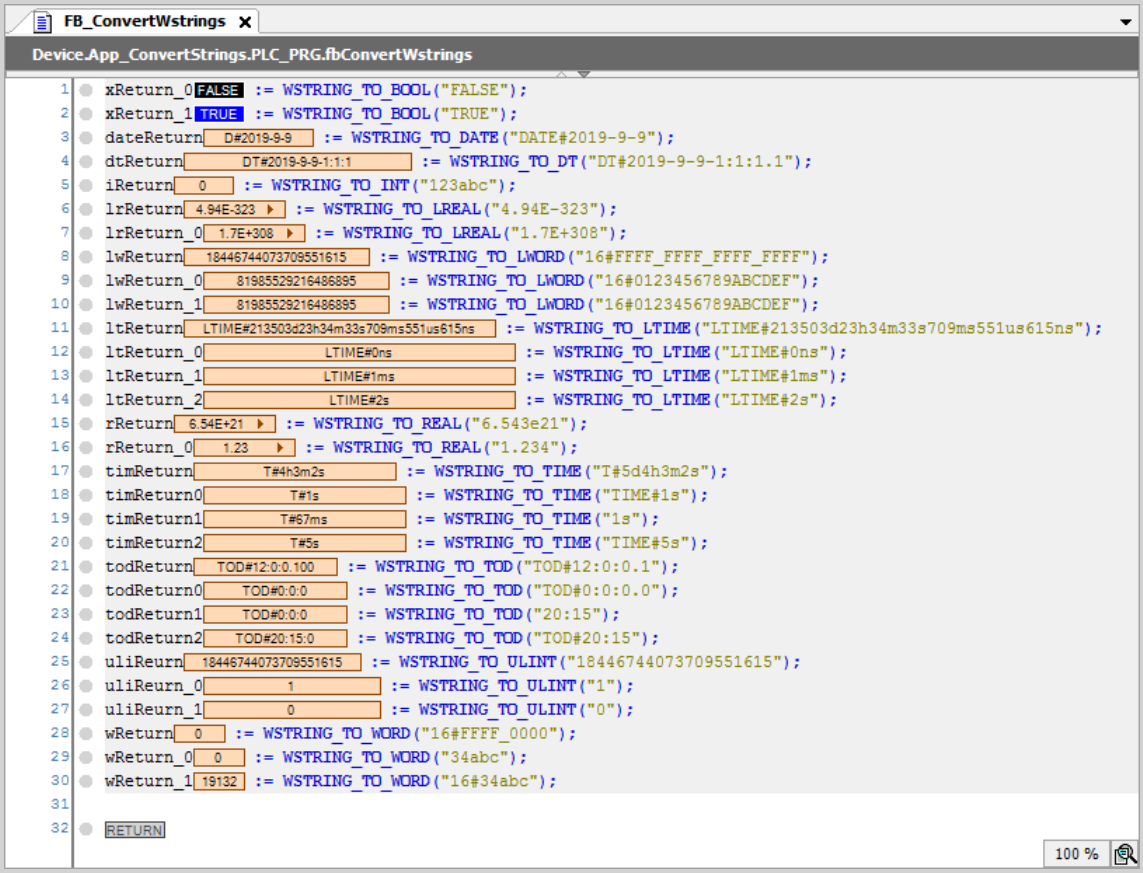
```
1  xReturn_0 FALSE := STRING_TO_BOOL('FALSE');
2  xReturn_1 TRUE := STRING_TO_BOOL('TRUE');
3  dateReturn D#2019-9-9 := STRING_TO_DATE('DATE#2019-9-9');
4  dtReturn DT#2019-9-9-1:1:1 := STRING_TO_DT('DT#2019-9-9-1:1:1.1');
5  iReturn 123 := STRING_TO_INT('123abc');
6  lrReturn 4.94E-323 := STRING_TO_LREAL('4.94E-323');
7  lrReturn_0 1.7E+308 := STRING_TO_LREAL('1.7E+308');
8  lwReturn 18446744073709551615 := STRING_TO_LWORD('16#FFFF_FFFF_FFFF_FFFF');
9  lwReturn_0 81985529216486895 := STRING_TO_LWORD('16#0123456789ABCDEF');
10 lwReturn_1 81985529216486895 := STRING_TO_LWORD('16#0123456789ABCDEF');
11 ltReturn LTIME#213503d23h34m33s709ms551us615ns := STRING_TO_LTIME('LTIME#213503d23h34m33s709ms551us615ns');
12 ltReturn_0 LTIME#0ns := STRING_TO_LTIME('LTIME#0ns');
13 ltReturn_1 LTIME#1ms := STRING_TO_LTIME('LTIME#1ms');
14 ltReturn_2 LTIME#2s := STRING_TO_LTIME('LTIME#2s');
15 rReturn 6.54E+21 := STRING_TO_REAL('6.543e21');
16 rReturn_0 1.23 := STRING_TO_REAL('1.234');
17 timReturn T#5d4h3m2s := STRING_TO_TIME('T#5d4h3m2s');
18 timReturn0 T#1s := STRING_TO_TIME('TIME#1s');
19 timReturn1 T#67ms := STRING_TO_TIME('1s');
20 timReturn2 T#5s := STRING_TO_TIME('TIME#5s');
21 todReturn TOD#12:0:0.100 := STRING_TO_TOD('TOD#12:0:0.1');
22 todReturn0 TOD#0:0:0 := STRING_TO_TOD('TOD#0:0:0.0');
23 todReturn1 TOD#0:0:0 := STRING_TO_TOD('20:15');
24 todReturn2 TOD#20:15:0 := STRING_TO_TOD('TOD#20:15');
25 uliReurn 18446744073709551615 := STRING_TO_ULINT('18446744073709551615');
26 uliReurn_0 1 := STRING_TO_ULINT('1');
27 uliReurn_1 0 := STRING_TO_ULINT('0');
28 wReturn 0 := STRING_TO_WORD('16#FFFF_0000');
29 wReturn_0 34 := STRING_TO_WORD('34abc');
30 wReturn_1 19132 := STRING_TO_WORD('16#34abc');
31 wstrReturn "Hello Worl := STRING_TO_WSTRING('Hello World!');
32 wstrReturn_0 "123456789" := STRING_TO_WSTRING('123456789');
33
34
```

100 %

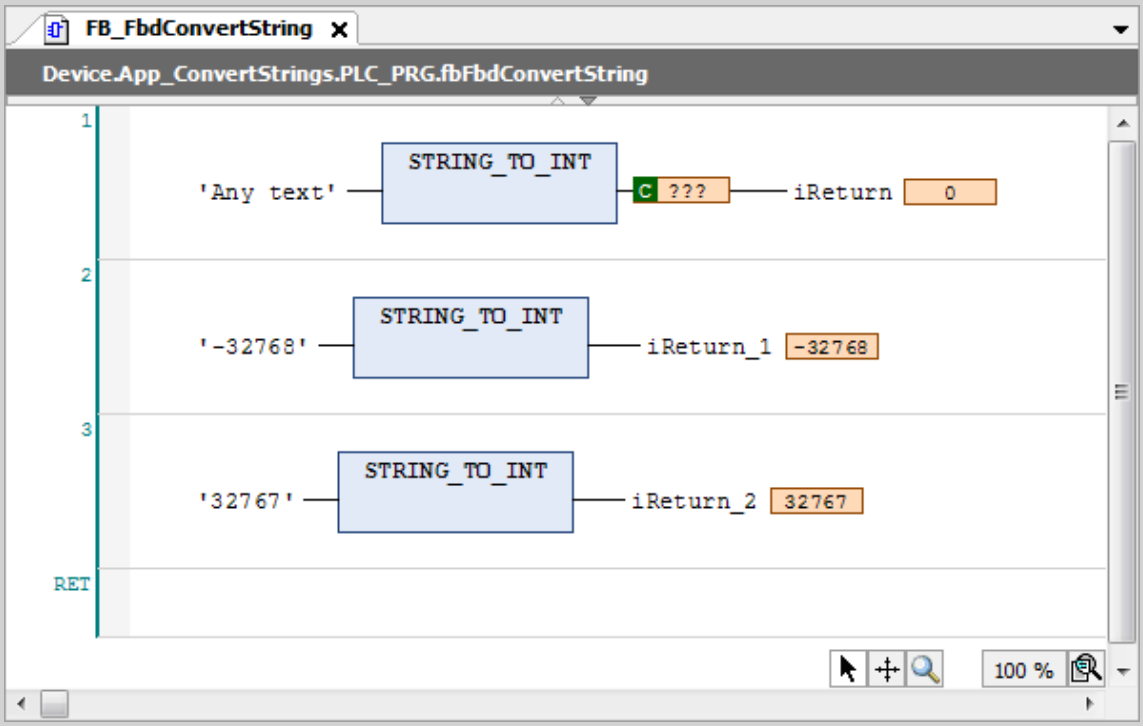
WSTRING conversion in ST

```
FUNCTION_BLOCK FB_ConvertWstrings
VAR
    xReturn_0: BOOL;
    xReturn_1: BOOL;
    dateReturn: DATE;
    dtReturn: DATE_AND_TIME;
    iReturn: INT;
    lrReturn: LREAL;
    lrReturn_0: LREAL;
    lwReturn: LWORD;
    lwReturn_0: LWORD;
    lwReturn_1: LWORD;
    ltReturn: LTIME;
    ltReturn_0: LTIME;
    ltReturn_1: LTIME;
    ltReturn_2: LTIME;
    rReturn: REAL;
    rReturn_0: REAL;
    timReturn: TIME;
    timReturn0: TIME;
    timReturn1: TIME;
    timReturn2: TIME;
    todReturn: TIME_OF_DAY;
    todReturn0: TIME_OF_DAY;
    todReturn1: TIME_OF_DAY;
    todReturn2: TIME_OF_DAY;
    uliReurn: ULINT;
    uliReurn_0: ULINT;
    uliReurn_1: ULINT;
    wReturn: WORD;
    wReturn_0: WORD;
    wReturn_1: WORD;
    wstrReturn: WSTRING;
    wstrReturn_0: WSTRING;
END_VAR

xReturn_0 := WSTRING_TO_BOOL("FALSE");
xReturn_1 := WSTRING_TO_BOOL("TRUE");
dateReturn := WSTRING_TO_DATE("DATE#2019-9-9");
dtReturn := WSTRING_TO_DT("DT#2019-9-9-1:1:1.1");
iReturn := WSTRING_TO_INT("123abc");
lrReturn := WSTRING_TO_LREAL("4.94E-323");
lrReturn_0 := WSTRING_TO_LREAL("1.7E+308");
lwReturn := WSTRING_TO_LWORD("16#FFFF_FFFF_FFFF_FFFF");
lwReturn_0 := WSTRING_TO_LWORD("16#0123456789ABCDEF");
lwReturn_1 := WSTRING_TO_LWORD("16#0123456789ABCDEF");
ltReturn :=
WSTRING_TO_LTIME("LTIME#213503d23h34m33s709ms551us615ns");
ltReturn_0 := WSTRING_TO_LTIME("LTIME#0ns");
ltReturn_1 := WSTRING_TO_LTIME("LTIME#1ms");
ltReturn_2 := WSTRING_TO_LTIME("LTIME#2s");
rReturn := WSTRING_TO_REAL("6.543e21");
rReturn_0 := WSTRING_TO_REAL("1.234");
timReturn := WSTRING_TO_TIME("T#5d4h3m2s");
timReturn0 := WSTRING_TO_TIME("TIME#1s");
timReturn1 := WSTRING_TO_TIME("1s");
timReturn2 := WSTRING_TO_TIME("TIME#5s");
todReturn := WSTRING_TO_TOD("TOD#12:0:0.1");
todReturn0 := WSTRING_TO_TOD("TOD#0:0:0.0");
todReturn1 := WSTRING_TO_TOD("20:15");
todReturn2 := WSTRING_TO_TOD("TOD#20:15");
```

```
uliReurn := WSTRING_TO_ULINT("18446744073709551615");
uliReurn_0 := WSTRING_TO_ULINT("1");
uliReurn_1 := WSTRING_TO_ULINT("0");
wReturn := WSTRING_TO_WORD("16#FFFF_0000");
wReturn_0 := WSTRING_TO_WORD("34abc");
wReturn_1 := WSTRING_TO_WORD("16#34abc");
```



FBD implementation language



See also

- *“Type conversion operators” on page 545*
- *Chapter 1.4.1.19.3.36 “Boolean Conversion” on page 567*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*
- *Chapter 1.4.1.19.3.37 “Integer Conversion” on page 572*
- *Chapter 1.4.1.19.3.38 “ Floating-Point Number Conversion” on page 584*
- *Chapter 1.4.1.19.3.41 “Date and Time Conversion” on page 600*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*

Time Conversion

NOTICE!

If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

The operators convert time values (`TIME` or `LIME`) into the specified data types and return this type-converted value.

Call syntax

```
<variable name> := <time conversion operator> ( <operand> );

<operand> = <variable name> | <literal>
```

Operators

```
TIME_TO___UXINT
TIME_TO___XINT
TIME_TO___XWORD
TIME_TO_BIT
TIME_TO_BOOL
TIME_TO_BYTE
TIME_TO_DATE
TIME_TO_DINT
TIME_TO_DT
TIME_TO_DWORD
TIME_TO_INT
TIME_TO_LDATE
TIME_TO_LDINT
TIME_TO_LINT
TIME_TO_LREAL
TIME_TO_LTIME
TIME_TO_LTOD
TIME_TO_LWORD
TIME_TO_REAL
TIME_TO_SINT
TIME_TO_STRING
TIME_TO_TOD
TIME_TO_UDINT
TIME_TO_UINT
TIME_TO_ULINT
TIME_TO_USINT
TIME_TO_WORD
TIME_TO_WSTRING
```

```
LTIME_TO___UXINT
LTIME_TO___XINT
LTIME_TO___XWORD
LTIME_TO_BIT
LTIME_TO_BOOL
LTIME_TO_BYTE
LTIME_TO_DATE
LTIME_TO_DINT
LTIME_TO_DT
LTIME_TO_DWORD
LTIME_TO_INT
LTIME_TO_LDATE
LTIME_TO_LDINT
LTIME_TO_LINT
LTIME_TO_LREAL
LTIME_TO_LTOD
LTIME_TO_LWORD
LTIME_TO_REAL
LTIME_TO_SINT
LTIME_TO_STRING
LTIME_TO_TIME
LTIME_TO_TOD
LTIME_TO_UDINT
LTIME_TO_UINT
LTIME_TO_ULINT
LTIME_TO_USINT
LTIME_TO_WORD
LTIME_TO_WSTRING
```

Converting to Boolean values

The operator returns `FALSE` if and only if the operand value can be interpreted as "0".

`xTime := TIME_TO_BOOL(T#0MS);`	`xTime = FALSE`
`xLongTime := TIME_TO_BOOL(T#0NS);`	`xLongTime = FALSE`
`xTime := TIME_TO_BOOL(T#1MS);`	`xDate = TRUE`
`xLongTime := TIME_TO_BOOL(T#1NS);`	`xLongTime = TRUE`

Converting to a string

NOTICE!

String manipulation when converting to `STRING` or `WSTRING`

When converting the type to `STRING` or `WSTRING`, the typed value is left-aligned as a character string and truncated if it is too long. Therefore, declare the return variable for the type conversion operators `<>_TO_STRING` and `<>_TO_WSTRING` long enough that the character string has enough space without any manipulation.

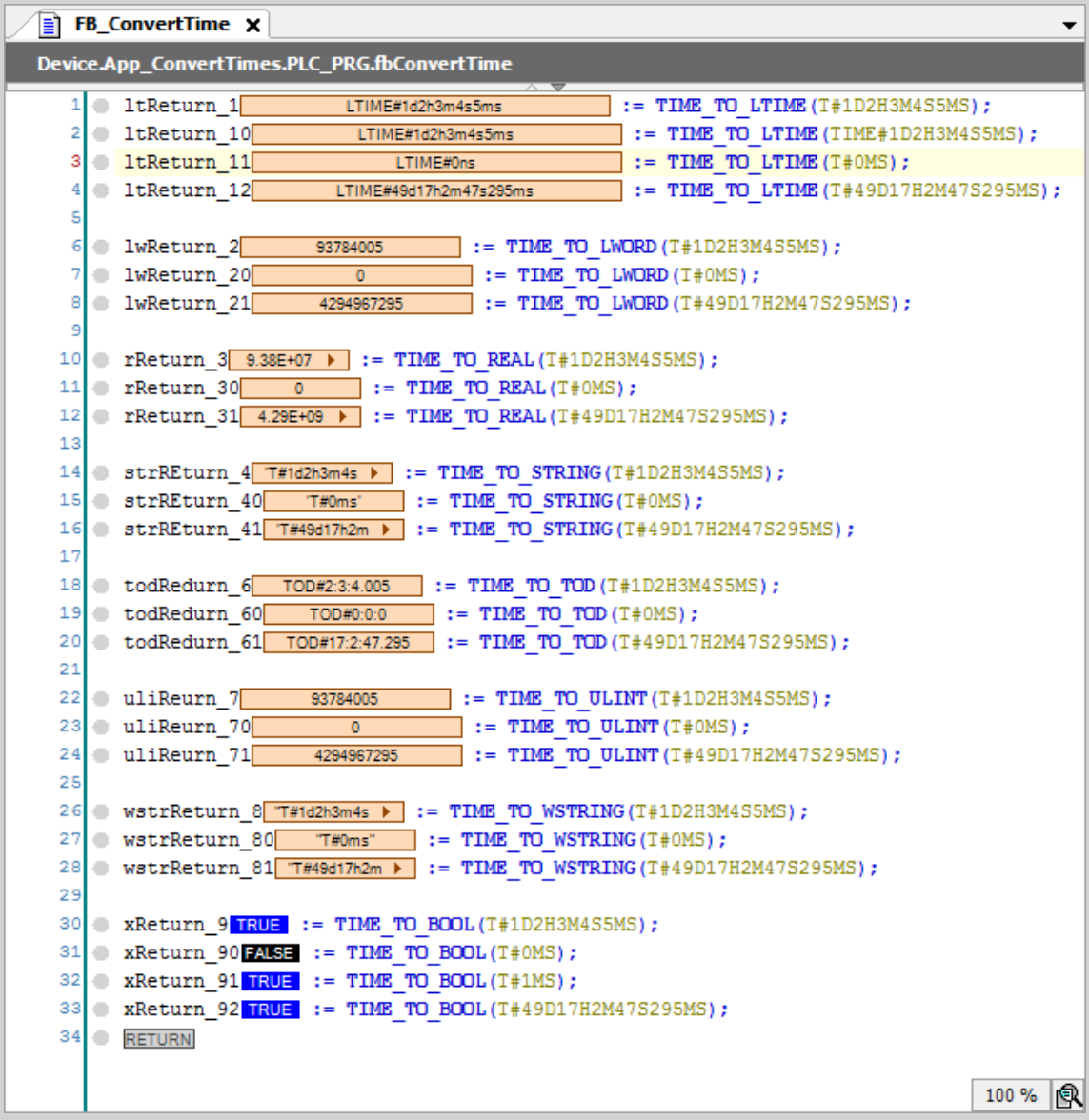
`sTime := TIME_TO_STRING(T#0MS);`	`sTime = 'T#0MS'`
`wsLongTime := LTIME_TO_WSTRING(T#0US);`	`wsLongTime = "T#0US"`

Examples

ST implementation language

```
FUNCTION_BLOCK FB_ConvertTimeAndDate
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    ltReturn_1: LTIME;
    lwReturn_2: LWORD;
    rReturn_3: REAL;
    strReturn_4: STRING;
    timReturn_5: TIME;
    todReturn_6: TIME_OF_DAY;
    uliReurn_7: ULINT;
    wstrReturn_8: WSTRING;
    wstrReturn_80: WSTRING;
    uliReurn_70: ULINT;
    todReturn_60: TIME_OF_DAY;
    timReturn_50: TIME;
    strReturn_40: STRING;
    rReturn_30: REAL;
    lwReturn_20: LWORD;
    ltReturn_10: LTIME;
    ltReturn_11: LTIME;
    lwReturn_21: LWORD;
    rReturn_31: REAL;
    strReturn_41: STRING;
    timReturn_51: TIME;
    todRedurn_61: TIME_OF_DAY;
    uliReurn_71: ULINT;
    wstrReturn_81: WSTRING;
    ltReturn_12: LTIME;
    xReturn_9: BOOL;
    xReturn_90: BOOL;
    xReturn_91: BOOL;
    xReturn_92: BOOL;
    dateReturn_6: DATE;
    timReturn_60: TIME;
    wReturn_61: WORD;
    todReturn_61: TIME_OF_DAY;
END_VAR

ltReturn_1 := DT_TO_LTIME(DT#2019-9-9-23:59:59);
ltReturn_10 := DT_TO_LTIME(DT#1970-1-1-0:0:0);
ltReturn_11 := DT_TO_LTIME(DT#1970-1-2-0:0:1);
ltReturn_12 := DT_TO_LTIME(DT#1970-1-3-12:30:30);

lwReturn_2 := TIME_TO_LWORD(T#5D4H2M3S2MS);
lwReturn_20 := TIME_TO_LWORD(T#0D0H0M0S0MS);

rReturn_3 := TIME_TO_REAL(T#5D4H2M3S2MS);
rReturn_30 := TIME_TO_REAL(T#0D0H0M0S0MS);

strREturn_4 := TIME_TO_STRING(T#5D4H2M3S2MS);
strREturn_40 := TIME_TO_STRING(T#0D0H0M0S0MS);

timReturn_5 := TOD_TO_TIME(TOD#23:59:59.999);
timReturn_50 := TOD_TO_TIME(TOD#0:0:0.000);
timReturn_51 := TOD_TO_TIME(TOD#0:0:0.001);
```

```
dateReturn_6 := TOD_TO_DATE(TOD#23:59:59.999);
timReturn_60 := TOD_TO_TIME(TOD#0:0:0.000);
wReturn_61 := TOD_TO_WORD(TOD#0:0:0.001);

uliReurn_7 := DATE_TO_ULINT(D#2019-9-9);
uliReurn_70 := DATE_TO_ULINT(D#1970-1-1);
uliReurn_71 := DATE_TO_ULINT(D#1970-1-2);

wstrReturn_8 := DATE_TO_WSTRING(D#2019-9-9);
wstrReturn_80 := DATE_TO_WSTRING(D#1970-1-1);
wstrReturn_81 := DATE_TO_WSTRING(D#1970-1-2);

xReturn_9 := DATE_TO_BOOL(D#2019-9-9);
xReturn_90 := DATE_TO_BOOL(D#1970-1-1);
xReturn_91 := DATE_TO_BOOL(D#1970-1-2);
xReturn_92 := DATE_TO_BOOL(D#1970-1-3);
```



FB_ConvertLongTime

Device.App_ConvertTimes.PLC_PRG.fbConvertLongTime

```
1  lwReturn_2  8645539203400204   := LTIME_TO_LWORD(LTIME#1000D15H23M12S34MS2US44NS);
2  lwReturn_20 18446744073709551615 := LTIME_TO_LWORD(LTIME#213503D23H34M33S709MS551US615NS);
3  lwReturn_21 0                  := LTIME_TO_LWORD(LTIME#0NS);
4
5  rReturn_3   1.84E+19           := LTIME_TO_REAL(LTIME#213503D23H34M33S709MS551US615NS);
6  rReturn_30  0                  := LTIME_TO_REAL(LTIME#0NS);
7  rReturn_31  1                  := LTIME_TO_REAL(LTIME#1NS);
8
9  strREturn_4  'LTIME#2135'      := LTIME_TO_STRING(LTIME#213503D23H34M33S709MS551US615NS);
10 strREturn_40 'LTIME#0ns'       := LTIME_TO_STRING(LTIME#0NS);
11 strREturn_41 'LTIME#1ns'       := LTIME_TO_STRING(LTIME#1NS);
12
13 todRedurn_6  TOD#17:2:47.295   := LTIME_TO_TOD(LTIME#213503D23H34M33S709MS551US615NS);
14 todRedurn_60 TOD#0:0:0         := LTIME_TO_TOD(LTIME#0NS);
15 todRedurn_61 TOD#0:0:0         := LTIME_TO_TOD(LTIME#0NS);
16
17 uliReurn_7  18446744073709551615 := LTIME_TO_ULINT(LTIME#213503D23H34M33S709MS551US615NS);
18 uliReurn_70 0                  := LTIME_TO_ULINT(LTIME#0NS);
19 uliReurn_71 1                  := LTIME_TO_ULINT(LTIME#1NS);
20
21 wstrReturn_8  'LTIME#2135'     := LTIME_TO_WSTRING(LTIME#213503D23H34M33S709MS551US615NS);
22 wstrReturn_80 'LTIME#0ns'      := LTIME_TO_WSTRING(LTIME#0NS);
23 wstrReturn_81 'LTIME#1ns'      := LTIME_TO_WSTRING(LTIME#1NS);
24
25 xReturn_9  TRUE  := LTIME_TO_BOOL(LTIME#213503D23H34M33S709MS551US615NS);
26 xReturn_90 FALSE := LTIME_TO_BOOL(LTIME#0NS);
27 xReturn_91 TRUE  := LTIME_TO_BOOL(LTIME#1NS);
28 xReturn_92 TRUE  := LTIME_TO_BOOL(LTIME#2NS);
29 RETURN
```

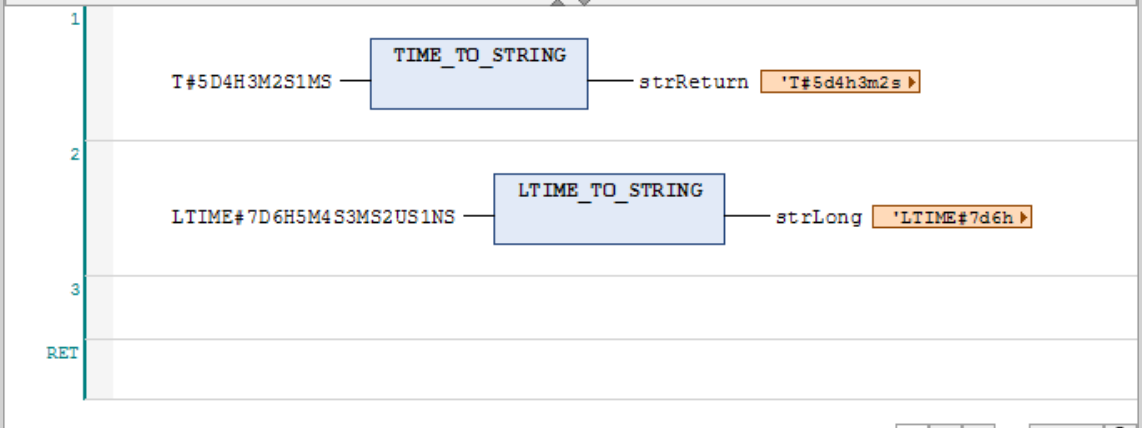
100 %

FBD implementation language

FB_FbdConvertTimes

Device.App_ConvertTimes.PLC_PRG.fbFbdConvertTimes

Expression	Type	Value
strReturn	STRING	'T#5d4h3m2s1ms'
strLong	STRING	'LTIME#7d6h5m4s3ms2us1ns'



100 %

See also

- *“Type conversion operators” on page 545*
- *Chapter 1.4.1.19.3.36 “Boolean Conversion” on page 567*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*
- *Chapter 1.4.1.19.3.37 “Integer Conversion” on page 572*
- *Chapter 1.4.1.19.3.38 “ Floating-Point Number Conversion” on page 584*
- *Chapter 1.4.1.19.3.39 “String Conversion” on page 587*
- *Chapter 1.4.1.19.3.41 “Date and Time Conversion” on page 600*

Date and Time Conversion

NOTICE!

If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.

Information can be lost when converting from larger data types to smaller data types.

The operators convert a date and time value into the specified data type and return a type-converted value.

Call syntax

```
<variable name> := <date and time conversion operator> ( <operand> );

<operand> = <variable name> | <literal>
```

The data types `DATE` and `DT` use the same memory format internally and are stored as `DWORD`. The resolution for `DATE` is 1 day. The resolution for `DT` is 1 second. Both begin at January 1, 1970.`TOD` is stored as `DWORD` with a resolution of 1 millisecond.

Operators

```
DATE_TO___UXINT
DATE_TO___XINT
DATE_TO___XWORD
DATE_TO_BIT
DATE_TO_BOOL
DATE_TO_BYTE
DATE_TO_DINT
DATE_TO_DT
DATE_TO_DWORD
DATE_TO_INT
DATE_TO_LINT
DATE_TO_LREAL
DATE_TO_LTIME
DATE_TO_LWORD
DATE_TO_REAL
DATE_TO_SINT
DATE_TO_STRING
DATE_TO_TIME
DATE_TO_TOD
DATE_TO_UDINT
DATE_TO_UINT
DATE_TO_ULINT
DATE_TO_USINT
DATE_TO_WORD
```

```
DATE_TO_WSTRING

DT_TO___UXINT
DT_TO___XINT
DT_TO___XWORD
DT_TO_BIT
DT_TO_BOOL
DT_TO_BYTE
DT_TO_DATE
DT_TO_DINT
DT_TO_DWORD
DT_TO_INT
DT_TO_LINT
DT_TO_LREAL
DT_TO_LTIME
DT_TO_LWORD
DT_TO_REAL
DT_TO_SINT
DT_TO_STRING
DT_TO_TIME
DT_TO_TOD
DT_TO_UDINT
DT_TO_UINT
DT_TO_ULINT
DT_TO_USINT
DT_TO_WORD
DT_TO_WSTRING

TOD_TO___UXINT
TOD_TO___XINT
TOD_TO___XWORD
TOD_TO_BOOL
TOD_TO_BIT
TOD_TO_BYTE
TOD_TO_DATE
TOD_TO_DINT
TOD_TO_DT
TOD_TO_DWORD
TOD_TO_INT
TOD_TO_LINT
TOD_TO_LREAL
TOD_TO_LTIME
TOD_TO_LWORD
TOD_TO_REAL
TOD_TO_SINT
TOD_TO_STRING
TOD_TO_TIME
TOD_TO_UDINT
TOD_TO_UINT
TOD_TO_ULINT
TOD_TO_USINT
TOD_TO_WORD
TOD_TO_WSTRING
```

Long operators

```
LDATE_TO___UXINT
LDATE_TO___XINT
LDATE_TO___XWORD
LDATE_TO_BIT
LDATE_TO_BOOL
LDATE_TO_BYTE
LDATE_TO_DATE
LDATE_TO_DINT
LDATE_TO_DT
LDATE_TO_DWORD
```

```
LDATE_TO_INT
LDATE_TO_LDT
LDATE_TO_LINT
LDATE_TO_LREAL
LDATE_TO_LTIME
LDATE_TO_LTOD
LDATE_TO_LWORD
LDATE_TO_REAL
LDATE_TO_SINT
LDATE_TO_STRING
LDATE_TO_TIME
LDATE_TO_TOD
LDATE_TO_UDINT
LDATE_TO_UINT
LDATE_TO_ULINT
LDATE_TO_USINT
LDATE_TO_WORD
LDATE_TO_WSTRING

LDT_TO___UXINT
LDT_TO___XINT
LDT_TO___XWORD
LDT_TO_BIT
LDT_TO_BOOL
LDT_TO_BYTE
LDT_TO_DATE
LDT_TO_DINT
LDT_TO_DWORD
LDT_TO_INT
LDT_TO_LDATE
LDT_TO_LINT
LDT_TO_LREAL
LDT_TO_LTIME
LDT_TO_LTOD
LDT_TO_LWORD
LDT_TO_REAL
LDT_TO_SINT
LDT_TO_STRING
LDT_TO_TIME
LDT_TO_TOD
LDT_TO_UDINT
LDT_TO_UINT
LDT_TO_ULINT
LDT_TO_USINT
LDT_TO_WORD
LDT_TO_WSTRING

LTOD_TO___UXINT
LTOD_TO___XINT
LTOD_TO___XWORD
LTOD_TO_BOOL
LTOD_TO_BIT
LTOD_TO_BYTE
LTOD_TO_DATE
LTOD_TO_DINT
LTOD_TO_DT
LTOD_TO_DWORD
LTOD_TO_INT
LTOD_TO_LDATE
LTOD_TO_LDT
LTOD_TO_LINT
LTOD_TO_LREAL
LTOD_TO_LTIME
LTOD_TO_LWORD
LTOD_TO_REAL
```

```
LTOD_TO_SINT
LTOD_TO_STRING
LTOD_TO_TIME
LTOD_TO_UDINT
LTOD_TO_UINT
LTOD_TO_ULINT
LTOD_TO_USINT
LTOD_TO_WORD
LTOD_TO_WSTRING
```

Converting to a Boolean value

The operator returns `FALSE` if and only if the operand value can be interpreted as "0".

`xDate := DATE_TO_BOOL(D#1970-1-1);`	`xDate = FALSE`
`xDateAndTime := DT_TO_BOOL(DT#1970-1-1-0:0:0);`	`xDateAndTime = FALSE`
`xTimeOfDay := TOD_TO_BOOL(TOD#0:0:0);`	`xTimeOfDay = FALSE`
`xDate := DATE_TO_BOOL(D#2019-9-1);`	`xDate = TRUE`
`xDateAndTime := DT_TO_BOOL(DT#2019-9-1-12:0:0);`	`xDateAndTime = TRUE`
`xTimeOfDay := TOD_TO_BOOL(TOD#12:0:0);`	`xTimeOfDay = TRUE`

Converting to an integer

The data types `DATE` and `DT` use the same memory format internally, namely a `DWORD`. The resolution for `DATE` is 1 day. The resolution for `DT` is 1 second. Both begin at January 1, 1970.

`TOD` is stored as `DWORD` with a resolution of 1 millisecond.

`diReturn_0 := DT_TO_DINT(DT#1970-1-1-0:0:0);`	`diReturn_0 = 0`
`diReturn_1 := DATE_TO_DINT(D#1970-1-1);`	`diReturn_1 = 0`
`diReturn_2 := TOD_TO_DINT(TOD#0:0:0);`	`diReturn_2 = 0`
`diReturn_1 := DT_TO_DINT(DT#1970-1-1-0:0:1);`	`diReturn_3 = 1`
`diReturn_3 := DATE_TO_DINT(D#1970-1-2);`	`diReturn_4 = 86400`
`diReturn_5 := DT_TO_DINT(DT#2019-9-1-12:0:0.0);`	`diReturn_5 = 1567339200`
`diReturn_6 := DATE_TO_DINT(D#2019-9-1);`	`diReturn_6 = 1567339200`
`diReturn_7 := TOD_TO_DINT(TOD#12:0:0);`	`diReturn_7 = 43200000`

Converting to a string

NOTICE!

String manipulation when converting to `STRING` or `WSTRING`

When converting the type to `STRING` or `WSTRING`, the typed value is left-aligned as a character string and truncated if it is too long. Therefore, declare the return variable for the type conversion operators `<>_TO_STRING` and `<>_TO_WSTRING` long enough that the character string has enough space without any manipulation.

The operands of type `DATE`, `DATE_AND_TIME`, `TIME_OF_DAY`, `DT`, or `TOD`, which are passed to an operator for a data and time conversion, are converted to their constant syntax (literal syntax). The generated string contains the keyword `D#`, `DT#` or `TOD#` and then the size with its data and time unit, as indicated in the IEC 61131-3 specification.

Examples

FB_ConvertToWstrings

Device.App_ConvertStrings.PLC_PRG.fbConvertToWstrings

```
1  wstrReturn "255" := BYTE_TO_WSTRING(2#1111_1111);
2  wstrReturn_1 "D#1970-01-" := DATE_TO_WSTRING(D#1970-1-1);
3  wstrReturn_2 "D#1970-01-" := DATE_TO_WSTRING(D#1970-1-2);
4  wstrReturn_3 "D#1970-01-" := DATE_TO_WSTRING(D#1970-1-1);
5  wstrReturn_4 "D#1970-01-" := DATE_TO_WSTRING(DATE#1970-1-1);
6  wstrReturn_5 "DT#1970-01" := DT_TO_WSTRING(DT#1970-1-2-0:0:0);
7  wstrReturn_6 "65535" := DINT_TO_WSTRING(65535);
8  wstrReturn_7 "4294967295" := DWORD_TO_WSTRING(16#FFFF_FFFF);
9  wstrReturn_8 "-3276" := INT_TO_WSTRING(-3276);
10 wstrReturn_9 "0" := LINT_TO_WSTRING(0);
11 wstrReturn_10 "1.0e309" := LREAL_TO_WSTRING(1.7E+308);
12 wstrReturn_11 "1844674407" := LWORD_TO_WSTRING(16#FFFF_FFFF_FFFF_FFFF);
13 wstrReturn_12 "1.234" := REAL_TO_WSTRING(1.234);
14 wstrReturn_13 "127" := SINT_TO_WSTRING(127);
15 wstrReturn_14 "Hello" := STRING_TO_WSTRING('Hello');
16 wstrReturn_15 "TOD#01:01:" := TOD_TO_WSTRING(TOD#1:1:1);
17 wstrReturn_16 "T#5d" := TIME_TO_WSTRING(T#5D);
18 wstrReturn_17 "4294967295" := UDINT_TO_WSTRING(4294967295);
19 wstrReturn_18 "65535" := UINT_TO_WSTRING(65535);
20 wstrReturn_19 "1" := ULINT_TO_WSTRING(1);
21 wstrReturn_20 "255" := USINT_TO_WSTRING(255);
22 wstrReturn_21 "65535" := WORD_TO_WSTRING(16#FFFF);
23
24 RETURN
```

100 %

Examples

FBD implementation language

The controller is in online mode in order to monitor the variables.

See also

- *“Type conversion operators” on page 545*
- *Chapter 1.4.1.19.3.36 “Boolean Conversion” on page 567*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*
- *Chapter 1.4.1.19.3.37 “Integer Conversion” on page 572*
- *Chapter 1.4.1.19.3.38 “ Floating-Point Number Conversion” on page 584*
- *Chapter 1.4.1.19.3.39 “String Conversion” on page 587*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*

Operator 'TRUNC'

The IEC operator is used for converting the `REAL` data type into the `DINT` data type. CODESYS takes only the integer part of the number.

> *In CoDeSys V2.3, the `TRUNC` operator converts `REAL` into `INT`. If you import a V2.3 project, then CODESYS automatically replaces `TRUNC` with `TRUNC_INT`.*

If CODESYS cannot represent the input value by a `DINT` or `INT`, then the result of this function is undefined. The behavior of such input values is platform-dependent.

> **NOTICE!**
> If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.
>
> Information can be lost when converting from larger data types to smaller data types.

Examples

Result in `diVar`: `1`

ST

```
diVar := TRUNC(1.9); (* Result: 1 *)

diVar := TRUNC(-1.4); (* Result: -1 *)
```

See also

- ↳ *"Type conversion operators" on page 545*

Operator 'TRUNC_INT'

The IEC operator is used for converting the `REAL` data type into the `INT` data type. CODESYS takes only the integer part of the number.

> *`TRUNC_INT` corresponds to the `TRUNC` operator in CoDeSys V2.3, and it is used automatically at this point when importing V2.3 projects. Note the change function of `TRUNC`.*

If CODESYS cannot represent the input value by a `DINT` or `INT`, then the result of this function is undefined. The behavior of such input values is platform-dependent.

> **NOTICE!**
> If the operand value for a type conversion operator is outside of the value range of the target data type, then the result output depends on the processor type and is therefore undefined. This is the case, for example, when a negative operand value is converted from `LREAL` to the target data type `UINT`.
>
> Information can be lost when converting from larger data types to smaller data types.

Examples

Result in `iVAR`: `1`

ST:

```
iVar := TRUNC_INT(1.9); (* Result: 1 *)

iVar := TRUNC_INT(-1.4); (* Result: -1 *)
```

See also

- *“Type conversion operators” on page 545*

Operator 'ABS'

This IEC operator yields the absolute value of a number.

Permitted data types for input and output variables and numeric constants: any numeric basic data type

Examples

Result in `i`: `2`

ST:

```
i := ABS(-2);
```

FBD:

Operator 'SQRT'

This IEC of course yields the square root of a number.

Permitted data types for input variables: any numeric basic data type

Permitted data types for output variables: `REAL` or `LREAL`

Examples

Result in `q`: `4`

ST:

```
q := SQRT(16);
```

FBD:

Operator 'LN'

This IEC operator yields the natural logarithm of a number.

Permitted data types for input variables: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result: `3.80666`

ST:

```
q := LN(45);
```

FBD:

Operator 'LOG'

This IEC operator yields the base-10 logarithm of a number.

The input variable can be any numeric basic data type, but the output variable must be the data type `REAL` or `LREAL`.

Examples

Result in `q`: `2.49762`

ST:

```
q := LOG(314.5);
```

FBD:

Operator 'EXP'

This IEC operator yields the exponential function.

Permitted data types for input variables: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in `q`: `7.389056099`

ST:

```
q := EXP(2);
```

FBD:

Operator 'EXPT'

This IEC operator raises a number to a higher power and returns the power of the base raised to the exponent: `power = base`$^{\texttt{exponent}}$. The input values (parameters) are the base and the exponent. The power function is undefined if the base is zero and the exponent is negative. However, the behavior depends on the platform in this case.

Syntax:

```
EXPT(<base>,<exponent>)
```

Permitted data types for the input values: Numeric base data types (`SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `LINT`, `ULINT`, `REAL`, `LREAL`, `BYTE`, `WORD`, `DWORD`, and `LWORD`)

Permitted data types for the return value: Floating-point number types (`REAL` and `LREAL`)

Example

Power function with literals

```
Var1 := EXPT(7,2);
```

FBD:

Return value: `Var1 = 49`

Example

Power function with variables

```
PROGRAM PLC_PRG
VAR
    lrPow : LREAL;
    iBase : INT := 2;
    iExponent : INT := 7;
END_VAR

lrPow := EXPT(iBase, iExponent);
```

Return value: `lrPow = 128`

Operator 'SIN'

The IEC operator yields the sine value of a number.

Permitted data types for input variables that measure the angle in radians: any numeric basic data type

Permitted data types for output variable: `REAL` and `LREAL`

The permitted range for the input value is -2^{63} to $+2^{63}$. On x86 and x64 systems: If the input value is outside of the permitted range, the function returns the input value

Examples

Result in `q`: `0.479426`

ST:

```
q := SIN (0.5);
```

FBD:

Operator 'COS'

The IEC operator yields the cosine value of a number.

Permitted data types for input variables that measure the angle in radians: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

> *The permitted range for the input value is -2^{63} to $+2^{63}$. On x86 and x64 systems: If the input value is outside of the permitted range, the function returns the input value*

Examples

Result in `q`: `0.877583`

ST:

```
q := COS(0.5);
```

FBD:

Operator 'TAN'

This IEC operator yields the tangent value of a number.

Permitted data types for input variables that measure the angle in radians: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in `q`: `0.546302`

ST:

```
q := TAN(0.5);
```

FBD:

Operator 'ASIN'

This IEC operator yields the arcsine value of a number.

Permitted data types for input variables: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in q: `0.523599`

ST:

```
q := ASIN(0,5);
```

FBD:

Operator 'ACOS'

This IEC operator yields the arccosine value of a number. The value is computed in radians.

Permitted data types for input variables that measure the angle in radians: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in q: 1.0472

ST:

```
q := ACOS(0.5);
```

FBD:

Operator 'ATAN'

This IEC operator yields the arctangent value of a number. The value is computed in radians.

Permitted data types for input variables that measure the angle in radians: any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in `q`: `0.463648`

ST:

```
q := ATAN(0.5);
```

FBD:

Operator '__DELETE'

This operator is an extension of the IEC 61131-3 standard.

> **NOTICE!**
> For compatibility, the compiler version must be >= 3.3.2.0.

The operator releases the memory of instances that the "__NEW" operator generated dynamically. The `__DELETE` operator does not have a return value and the operand is set to zero after this operation.

Requirement: In the properties dialog of the application, the *"Use dynamic memory allocation"* check box is selected in the *"Application Build Options"* tab.

```
__DELETE (<pointer>)
```

NOTICE!
Two tasks should not call __DELETE simultaneously. Either you use a semaphore (SysSemEnter) or comparable method to prevent any concurrent calling of __DELETE , or you use __DELETE in one tasks only (recommended).

You can use a semaphore (SysSemEnter) to prevent two tasks from allocating memory at the same time. As a consequence, the extensive use of `__DELETE` causes higher jitter.

If `Pointer` references a function block, then CODESYS calls the associated `FB_EXIT` method before the pointer is set to zero.

Examples

```
FUNCTION_BLOCK FBDynamic

VAR_INPUT
  in1, in2 : INT;
END_VAR

VAR_OUTPUT
  out : INT;
END_VAR

VAR
  test1 : INT := 1234;
  _inc : INT := 0;
  _dut : POINTER TO DUT;
  neu : BOOL;
END_VAR

out := in1 + in2;


METHOD FB_Exit : BOOL

VAR_INPUT
  bInCopyCode : BOOL;
END_VAR

__Delete(_dut);



METHOD FB_Init : BOOL

VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode : BOOL;
END_VAR

_dut := __NEW(DUT);


METHOD INC : INT

VAR_INPUT
END_VAR

_inc := _inc + 1;
INC := _inc;


PLC_PRG(PRG)

VAR
    pFB : POINTER TO FBDynamic;
    bInit: BOOL := TRUE;
    bDelete: BOOL;
    loc : INT;
END_VAR

IF (bInit) THEN
    pFB := __NEW(FBDynamic);
    bInit := FALSE;
END_IF
```

```
IF (pFB <> 0) THEN
    pFB^(in1 := 1, in2 := loc, out => loc);
    pFB^.INC();
END_IF

IF (bDelete) THEN
    __DELETE(pFB);
END_IF
```

Operator '__ISVALIDREF'

This operator is an extension of the IEC 61131-3 standard.

The operator is used for checking whether a reference refers to a valid value. For a description of use and an example, refer to the description for the `REFERENCE` data type.

See also

- ↳ *Chapter 1.4.1.19.5.13 “Reference” on page 658*

Operator '__NEW'

The operator is an extension of the IEC 61131-3 standard.

The `__NEW` operator reserves dynamic memory to instantiate function blocks, user-defined data types, or arrays of standard types. The operator returns a matching typed pointer.

Requirement: In the properties dialog of the parent application, on the *“Application Build Options”* tab, the *“Use dynamic memory allocation”* option is selected.

Syntax

```
<pointer name> := __NEW( <type> ( , <size> )? );
__DELETE( <pointer name> );

<type> : <function block> | <data unit type> | <standard data type>
```

The operator generates an instance of the type `<type>` and returns a pointer to this instance. Then the initialization of the instance is called. If `<type>` is a scalar standard data type, then the optional operand `<size>` is also evaluated. Then the operator generates an array of type `<standard data type>` and size `<size>`. If the attempt to allocate memory fails, then `__NEW` returns the value `0`.

Use the operator within the assignment "`:=`". Otherwise an error message is displayed.

A function block or a user-defined data type whose instance is created dynamically with `__NEW` uses a fixed memory area. Here it is required that you mark the objects with the pragma `{attribute 'enable_dynamic_creation'}`. It is not required for function blocks that are part of a library.

If you change the data layout of the function block in online mode, then you cannot execute a login with an online change afterwards. This is because the memory area of the function block instance has been invalidated. You change the data layout when you add new variables to the function block, delete existing variables, or change the data types of variables.

Example

Array (DWORD):

```
PROGRAM PLC_PRG
VAR
    pdwScalar : POINTER TO DWORD; //Typed pointer
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pdwScalar := __NEW(DWORD, 16); // Allocates memory (16 dwords)
and assigns them to pointer pdwScalar
END_IF
IF (xDelete) THEN
    __DELETE(pdwScalar); // Frees memory of pointer
END_IF
```

Function block:

```
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FBComputeGamma
VAR_INPUT
    iAlpha : INT;
    iBeta : INT;
END_VAR
VAR_OUTPUT
    iGamma : INT;
END_VAR
VAR
END_VAR

iGamma := iAlpha + iBeta;

PROGRAM PLC_PRG
VAR
    pComputeGamma : POINTER TO FBComputeGamma;    // Typed pointer
    xInit : BOOL := TRUE;
    xDelete : BOOL;
    iResult : INT;
END_VAR

IF (xInit) THEN
    pComputeGamma := __NEW(FBComputeGamma); // Allocates memory
    xInit := FALSE;
END_IF
pComputeGamma^.iAlpha := (pComputeGamma^.iAlpha + 1)MOD 100; //
Sets first input of pComputeGamma
pComputeGamma^.iBeta := 10; // Sets second input of pComputeGamma
pComputeGamma^(); // Calls the FB pComputeGamma is pointing to
iResult := pComputeGamma^.iGamma; // Reads output of pComputeGamma
IF (xDelete) THEN
    __DELETE(pComputeGamma); // Frees memory
END_IF
```

User-defined data type (DUT):

```
{attribute 'enable_dynamic_creation'}
TYPE ABCDATA :
STRUCT
        iA, iB, iC, iD : INT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    pABCData : POINTER TO ABCDATA; // Typed pointer
    xInit : BOOL := TRUE;
```

```
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pABCData := __NEW(ABCDATA); // Allocates memory
    xInit := FALSE;
END_IF
IF (xDelete) THEN
    __DELETE(pABCData); // Frees memory
END_IF
```

Array (BYTE):

```
PROGRAM PLC_PRG
VAR
    pbDataAlpha : POINTER TO BYTE;
    pbDataBeta : POINTER TO BYTE;
    xInit : BOOL := TRUE;
    xDelete : BOOL;
    usiCnt : USINT;
    bTestC: BYTE;
END_VAR

IF (xInit) THEN
    pbDataAlpha := __NEW(BYTE, 16); // Allocates 16 bytes for
pbDataAlpha
    pbDataBeta := __NEW(BYTE);  // Allocates memory for pbDataBeta
    xInit := FALSE;

    FOR usiCnt := 0 TO 15 DO
        pbDataAlpha[usiCnt] := usiCnt; // Writes to new array
    END_FOR
    pbDataBeta^:= 16#FF; // Writes to new data
END_IF

bTestC := pbDataAlpha[12];  // Reads new array by index access

IF (xDelete) THEN // Frees memory
    __DELETE(pbDataAlpha);
    __DELETE(pbDataBeta);
END_IF
```

NOTICE!

We do not recommend the simultaneous execution of two tasks that both call the `__NEW` operator. You use either a semaphore (`SysSemEnter`) or a comparable technique to prevent a concurrent call of `__NEW`. However, this results in a higher jitter when `__NEW` is applied extensively.

We recommend that you call `__NEW` operators in one task only.

See also

- *Chapter 1.4.1.20.4.10.9 “Dialog 'Properties - Application Build Options'” on page 1162*
- *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835*
- *Chapter 1.4.1.20.2.18.2 “Object 'Function Block'” on page 883*
- *Chapter 1.4.1.19.6.2.12 “Attribute 'enable_dynamic_creation'” on page 695*
- *Chapter 1.4.1.19.11.159 “Compiler error C0509” on page 815*

Operator '__QUERYINTERFACE'

This operator is an extension of the IEC 61131-3 standard.

At runtime, the operator executes a type conversion of an interface reference into another type. The operator returns a `BOOL` result. `TRUE` means that CODESYS has performed the conversion successfully.

```
__QUERYINTERFACE(<ITF_Source>,<ITF_Dest>);
```

1.Operand: Interface reference or FB interface

2.Operand: Interface reference with required target type

The requirement for the explicit conversion is that both the ITF_Source and ITF_Dest are derived from `Interface __System.IQueryInterface`. This interface is implicitly available does not require a library.

Example

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD

INTERFACE ItfDerived1 EXTENDS ItfBase
METHOD mderived1 : BOOL
END_METHOD

INTERFACE ItfDerived2 EXTENDS ItfBase
METHOD mderived2 : BOOL
END_METHOD

FUNCTION_BLOCK FB1 IMPLEMENTS ItfDerived1
METHOD mbase : BOOL
    mbase := TRUE;
END_METHOD
METHOD mderived1 : BOOL
    mderived1 := TRUE;
END_METHOD
END_FUNCTION_BLOCK

FUNCTION_BLOCK FB2 IMPLEMENTS ItfDerived2
METHOD mbase : BOOL
    mbase := FALSE;
END_METHOD
METHOD mderived2 : BOOL
    mderived2 := TRUE;
END_METHOD
END_FUNCTION_BLOCK

PROGRAMM POU
VAR
    inst1 : FB1;
    inst2 : FB2;
    itfbase1 : ItfBase := inst1;
    itfbase2 : ItfBase := inst2;
    itfderived1 : ItfDerived1 := 0;
    itfderived2 : ItfDerived2 := 0;
    xResult1, xResult2, xResult3, xResult4: BOOL;
END_VAR


xResult1 := __QUERYINTERFACE(itfbase1, itfderived1); // xResult =
TRUE, itfderivedi1 <>0
                                                     // references
the instance inst1
xResult2 := __QUERYINTERFACE(itfbase1, itfderived2); // xResult =
FALSE, itfderived2 = 0
xResult3 := __QUERYINTERFACE(itfbase2, itfderived1); // xResult =
FALSE, itfderived1 = 0
xResult4 := __QUERYINTERFACE(itfbase2, itfderived2); // xResult =
TRUE, itfderived2 <> 0
                                                     // references
the instance inst2
```

Operator '__QUERYPOINTER'

This operator is an extension of the IEC 61131-3 standard.

At runtime, the operator makes it possible to convert the type of an interface reference of a function block to a pointer. The operator returns a `BOOL` result. `TRUE` means that CODESYS has performed the conversion successfully.

> **NOTICE!**
> For compatibility, the definition of the pointer to be converted must be an extension of the base interface "__SYSTEM.IQueryInterface".

```
__QUERYPOINTER (<ITF_Source>, <Pointer_Dest>)
```

The operator receives an interface reference or a FB instance with the required target types as the first operand and a pointer as the second operand. After processing `__QUERYPOINTER`, Pointer_Dest receives the pointer to the reference or instance of a function block that the ITF_Source interface reference currently refers to. Pointer_Dest is not typed and can be cast to any type. You have to make sure of the type. For example, the interface could offer a method that returns a type code.

Operators '__TRY', '__CATCH', '__FINALLY', '__ENDTRY'

These operators are extended from the IEC 61131-3 standard and they are used for specific exception handling in IEC code.

Syntax

```
__TRY
    <statements_try>
__CATCH(exec)
    <statements_catch>
__FINALLY
    <statements_finally>
__ENDTRY
    <statements_next>
```

When a statement in the `__Try` operator throws an exception, the application does not stop. Instead, the application executes the statements in `__Catch`, starts the exception handling, and then executes the statements in `__FINALLY`. The exception handling ends with `__ENDTRY`, and the application executes the subsequent statements.

An IEC variable for an exception has the data type `__System.ExceptionCode`.

Example

If the statement in `__TRY` throws an exception, then program execution is not stopped. Instead, the statement in `__CATCH` is executed. Therefore, in this example, the application executes the `exc` function, then the statement in `__FINALLY`, and finally the statement in `__ENDTRY`.

```
FUNCTION Tester : UDINT
VAR_INPUT
    count : UDINT;
END_VAR
VAR_OUTPUT
    strExceptionText : STRING;
END_VAR
VAR
    exc : __SYSTEM.ExceptionCode;
END_VAR

__TRY
Tester := tryFun(count := count, testcase := g_testcase);  // This
statement is tested. If it throws an exception, then the statement
in __CATCH is executed first, and then the statement in __FINALLY.
__CATCH(exc)
HandleException(exc, strExceptionText => strExceptionText);
__FINALLY
GVL.g_count := GVL.g_count + 2;
__ENDTRY
```

See also

- *Chapter 1.4.1.20.3.6.19 “Command 'Stop Execution on Handled Exceptions'” on page 1043*

Data Type '__System.ExceptionCode'

```
TYPE ExceptionCode :
(
RTSEXCPT_UNKNOWN  := 16#FFFFFFFF,
RTSEXCPT_NOEXCEPTION  := 16#00000000,
RTSEXCPT_WATCHDOG  := 16#00000010,
RTSEXCPT_HARDWAREWATCHDOG  := 16#00000011,
RTSEXCPT_IO_CONFIG_ERROR  := 16#00000012,
RTSEXCPT_PROGRAMCHECKSUM  := 16#00000013,
RTSEXCPT_FIELDBUS_ERROR  := 16#00000014,
RTSEXCPT_IOUPDATE_ERROR  := 16#00000015,
RTSEXCPT_CYCLE_TIME_EXCEED  := 16#00000016,
RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED  := 16#00000017,
RTSEXCPT_UNRESOLVED_EXTREFS   := 16#00000018,
RTSEXCPT_DOWNLOAD_REJECTED                := 16#00000019,
RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR        := 16#0000001A,
RTSEXCPT_LOADBOOTPROJECT_FAILED           := 16#0000001B,
RTSEXCPT_OUT_OF_MEMORY                   := 16#0000001C,
RTSEXCPT_RETAIN_MEMORY_ERROR            := 16#0000001D,
RTSEXCPT_BOOTPROJECT_CRASH                := 16#0000001E,
RTSEXCPT_BOOTPROJECTTARGETMISMATCH        := 16#00000021,
RTSEXCPT_SCHEDULEERROR                    := 16#00000022,
RTSEXCPT_FILE_CHECKSUM_ERR                := 16#00000023,
RTSEXCPT_RETAIN_IDENTITY_MISMATCH        := 16#00000024,
RTSEXCPT_IEC_TASK_CONFIG_ERROR            := 16#00000025,
RTSEXCPT_APP_TARGET_MISMATCH             := 16#00000026,
RTSEXCPT_ILLEGAL_INSTRUCTION             := 16#00000050,
RTSEXCPT_ACCESS_VIOLATION                 := 16#00000051,
```

```
RTSEXCPT_PRIV_INSTRUCTION               := 16#00000052,
RTSEXCPT_IN_PAGE_ERROR                   := 16#00000053,
RTSEXCPT_STACK_OVERFLOW                   := 16#00000054,
RTSEXCPT_INVALID_DISPOSITION            := 16#00000055,
RTSEXCPT_INVALID_HANDLE                    := 16#00000056,
RTSEXCPT_GUARD_PAGE                        := 16#00000057,
RTSEXCPT_DOUBLE_FAULT                    := 16#00000058,
RTSEXCPT_INVALID_OPCODE                    := 16#00000059,
RTSEXCPT_MISALIGNMENT                    := 16#00000100,
RTSEXCPT_ARRAYBOUNDS                    := 16#00000101,
RTSEXCPT_DIVIDEBYZERO                    := 16#00000102,
RTSEXCPT_OVERFLOW                        := 16#00000103,
RTSEXCPT_NONCONTINUABLE                    := 16#00000104,
RTSEXCPT_PROCESSORLOAD_WATCHDOG            := 16#00000105,
RTSEXCPT_FPU_ERROR                        := 16#00000150,
RTSEXCPT_FPU_DENORMAL_OPERAND            := 16#00000151,
RTSEXCPT_FPU_DIVIDEBYZERO                := 16#00000152,
RTSEXCPT_FPU_INEXACT_RESULT                := 16#00000153,
RTSEXCPT_FPU_INVALID_OPERATION            := 16#00000154,
RTSEXCPT_FPU_OVERFLOW                    := 16#00000155,
RTSEXCPT_FPU_STACK_CHECK                := 16#00000156,
RTSEXCPT_FPU_UNDERFLOW                    := 16#00000157,
RTSEXCPT_VENDOR_EXCEPTION_BASE            := 16#00002000
RTSEXCPT_USER_EXCEPTION_BASE            := 16#00010000
) UDINT ;
END_TYPE
```

Operator '__VARINFO'

This operator is an extension of the IEC 61131-3 standard.

The operator yields information about a variable. You can save the information as data structure in a variable of data type `__SYSTEM.VAR_INFO`.

Syntax in the declaration:

```
<name of the info variable> : __SYSTEM.VAR_INFO; // Data structure
for info variable
```

Syntax for the call:

```
<name of the info variable> := __VARINFO( <variable name> ); // Call
of the operator
```

Example

```
FUNCTION_BLOCK FB_Velocity
VAR_INPUT
    rVelocity: REAL := 1.2;
END_VAR
VAR_OUTPUT
END_VAR
VAR
    infoVelocity: __SYSTEM.VAR_INFO; //Info of Velocity
END_VAR

infoVelocity := __VARINFO(rVelocity); // Gets the info of Velocity
locally


PROGRAM PLC_PRG
VAR
    iCounter : INT := 0; // Counts the calls
    infoCounter : __SYSTEM.VAR_INFO; //Info of Counter
    arrA : ARRAY [1..2, 1..2, 1..2] OF INT := [0, 1, 2, 3, 4, 5, 6,
7]; // Stores the A data
    infoA : __SYSTEM.VAR_INFO; //Info of A
    fbVel : FB_Velocity;
END_VAR

iCounter := iCounter + 1;
infoCounter := __VARINFO(iCounter);
infoA := __VARINFO(arrA);
fbVel();
```

The `iCounter` and `arrA` variables are recognized in the application code. The variable information is saved in the `infoCounter` and `infoA` variables. Moreover, the `FB_Velocity` function block is instantiated.

Data type `__SYSTEM.VAR_INFO`

Name	Data type	Initialization	Description
`ByteAddress`	`DWORD`	0	Address of the variable Example: `16#072E35EC` Note: For bit access of a variable `<variable name>.<bit index>`, the address of the variable that contains the bit is given.
`ByteOffset`	`DWORD`	0	Offset of the variable address (in bytes). Example: `13936` bytes. Note: If the variable is global, then the offset is relative to the beginning of the area. If the variable is a local variable in a function or method, then the offset is relative to the current stack frame. If the variable is a local variable in a function block, then the offset is relative to the function block instance.

Name	Data type	Initialization	Description
Area	DINT	0	Memory area number Area in the runtime system. Example: -1: Means that the variable is not global in the memory, but relative to an instance or on the stack. Note: The memory areas are device-dependent.
BitNr	INT	0	Number of bits (in bytes) Example: 16#00FF bytes Note: If the variable is **not** an integer data type, then: BitNr = -1 = 16#FFFF.
BitSize	INT	0	Memory size of the variable (in bits) Example: 16 bits
BitAddress	UDINT	0	Bit address of the variable Requirement: The variable is located in the input memory area I, output memory area Q, or marker memory area M. Otherwise the value is undefined.
TypeClass	TYPE_CLASS	TYPE_BOOL	Data type class of the variable Example: TYPE_INT, TYPE_ARRAY Note: For user-defined data types or function block instances, TYPE_USERDEF is output as the data type class.
TypeName	STRING(79)	''	Date type name of the variable as STRING(79) Note: For user-defined data types, the function block name or the DUT name is output. Example: 'INT', 'ARRAY'
NumElements	UDINT	0	Number of array elements Requirement: The variable has the data type ARRAY. Example: 8
BaseTypeClass	TYPE_CLASS	TYPE_BOOL	Elementary basic data type of the array elements. Requirement: The variable has the data type ARRAY. Example: TYPE_INT for arrA : ARRAY [1..2,1..2,1..2] OF INT;
ElemBitSize	UDINT	0	Memory size of the array element (in bits) Requirement: The variable has the data type ARRAY. Example: 16 bits for arrA : ARRAY [1..2,1..2,1..2] OF INT;

Name	Data type	Initialization	Description
MemoryArea	MEMORY_AREA	MEM_MEMORY	Information about the memory area ● MEM_GLOBAL: Global memory area For example in Area 0 ● MEM_LOCAL: Local memory area in Area -1 ● MEM_MEMORY: Marker memory area %M For example in 16#10 in Area 1 ● MEM_INPUT: Input memory area %I For example in 16#04 in Area 2 ● MEM_OUTPUT: Output memory area %Q For example in 16#08 in Area 3 ● MEM_RETAIN: Retain memory area For example in 16#20 in Area 0 Example: MEM_GLOBAL Note: The memory area configuration is device-dependent.
Symbol	STRING(39)	"	Variable name as STRING(39) Example: 'iCounter', 'arrA'
Comment	STRING(79)	"	Comment of the variable declaration Example: 'Counts the calls' or 'Stores the A data'

Operator '__CURRENTTASK'

This operator is an extension of the IEC 61131-3 standard.

In runtime mode, the operator provides information about the IEC task that is currently running.

The operator is supported only on target systems in which the target system setting `memory-layout\max-stack-size` is set to a value > 0.

The operator allows for access to a structure with two variables:

- TaskIndex: Zero-based index that identifies the task
- pTaskInfo: Detailed information about the currently running task. It can be assigned to a POINTER TO Task_Info2 from the library CmpIecTask.

The operator cannot be used in the declaration of a POU. This would result in an error message. If the current task cannot be determined, then the TaskIndex -1 and the pTaskInfo are zero.

Example

```
//Declaration
VAR
    idx : INT;
    pInfo : POINTER TO Task_Info2;
END_VAR

//Program code

idx := __CURRENTTASK.TaskIndex;
pInfo := __CURRENTTASK.pTaskInfo;
```

Operator '__COMPARE_AND_SWAP

The multicore operator is an extension of the IEC 61131-3 standard.

The operator can be used for implementing a semaphore, for example to guarantee exclusive access to a variable written to by different tasks.

`__COMPARE_AND_SWAP` gets a pointer to a data type `__XWORD` variable, an old value, and a new value as its input (example: `bMutex := __COMPARE_AND_SWAP(ADR(dwSynch), dwOld, dwNew);`). The old and new values can also be data type `__XWORD` variables. The referenced `__XWORD` variable is compared with the old value and if both are equal, then the new value is written. The result of the function is `TRUE` when the new value could be written.

The compiler automatically replaces the data type `__XWORD` with `DWORD` on 32-bit systems and `LWORD` on 64-bit systems.

This operation is atomic, so it cannot be interrupted by another task, even on multicore platforms.

Example

The following example shows a typical usage. Exclusive access to a type `STRING` variable, which is addressed via the `pstrOutput` pointer, should be implemented.

The access to a string is not atomic. If multiple tasks write to the same string at the same time, then the contents may be inconsistent.

With this function, it is now possible to write the same STRING variable in different tasks.

```
FUNCTION ExclusiveStringWrite : BOOL
VAR_INPUT
    strToWrite : STRING;
    pstrOutput : POINTER TO STRING;
END_VAR
VAR_STAT
  dwSynch : __XWORD;
END_VAR
VAR
    bMutex: BOOL;
END_VAR

bMutex:= __COMPARE_AND_SWAP(ADR(dwSynch), 0, 1);
(* compare with 0 and write 1 as atomic operation *)
IF bMutex THEN                          // bMutex is TRUE if write
could be done
    pstrOutput^ := strToWrite;          // Now you can write safely
on the string
    dwSynch := 0;                       // The __XWORD variable must
be reset.
    ExclusiveStringWrite := TRUE;          // Writing was successful
ELSE
    ExclusiveStringWrite := FALSE;      // Writing was not successful
END_IF
```

See also

- *“Multicore operators” on page 546*
- *Chapter 1.4.1.19.3.68 “Operator 'TEST_AND_SET'” on page 628*

Operator '__XADD'

The multicore operator is an extension of the IEC 61131-3 standard.

The operator can be used for implementing an atomic counter. If an integer variable is incremented by means of ordinary addition, for example `iTest := iTest + 1;`, then this operation is not executed atomically. Another access to the variable could take place between reading and writing the variable.

If the counter is incremented in multiple tasks, then the counter result can be less than the number of counting cycles. So if two tasks execute the above code one time and the variable previously had the value `0`, then the variable can then have the value `1`. This is especially problematic if arrays are being processed in multiple tasks and a unique index is required for the array in each processing cycle.

When the `__XADD` operator is called, it gets a pointer to a type `DINT` variable as the first summand and a type `DINT` value as the second summand. `__XADD` returns the old value of the first summand and in the same step adds the second summand to the first summand.

For example, the function call can look like this: `diOld := __XADD(ADR(diVar), deAdd);`

Example

The following example shows a typical usage. An array should be populated from two tasks. In the process, all positions in the array should be used and no position should be overwritten.

With this function, multiple tasks can populate a Boolean array.

```
FUNCTION WriteToNextArrayPosition : BOOL
VAR_EXTERNAL
    g_diIndex : DINT;  // Index and array are globally defined and
used by multiple tasks
    g_boolArray : ARRAY [0..1000] OF BOOL;
END_VAR
VAR_INPUT
    bToWrite : BOOL;
END_VAR
VAR
    diIndex : DWORD;
END_VAR

diIndex := __XADD(ADR(g_diIndex), 1);        // Returns a unique
index
WriteToNextArrayPosition := FALSE;
IF (diIndex >= 0 AND diIndex <= 1000) THEN
    g_boolArray[diIndex] := bToWrite;         //Writes to unique
index
    WriteToNextArrayPosition := TRUE;         // TRUE: Array was
not full yet
END_IF
```

See also

- *“Multicore operators” on page 546*

Operator '__POSITION'

The operator is an extension of the IEC 61131-1 standard.

At runtime, the operator yields the position of a variable in the declaration part or in the implementation part of a POU. The operator has to be assigned the variables of type `STRING` in the declaration part or in the implementation part.

Result of `__POSITION`

- Declaration part: `'Line <line number> (Decl)'`
- Implementation part: `'Line <line number>, Column <column number> (Impl)'`

Example

```
PROGRAM PROG1
VAR
    strPOS : STRING := __POSITION();  //Yields the line number of
this declaration
    strlocalPOS : STRING;
END_VAR

    strlocalPOS := __POSITION();      //Yields the line and column
number of this assignment
```

Operator '__POUNAME'

The operator is an extension of the IEC 61131-1 standard.

At runtime, the operator yields the name of the POU that contains the operator `__POUNAME`. The result is of type `STRING`.

The result of `__POUNAME` depends where it is used:

- In a program: program name
- In a function name: function name
- In a function block: function block name
- In a method: the method name qualified with the FB name
- In a Get/Set accessor of a property: the property name + Get/Set qualified with the FB name
- In a GVL: GVL name
- In a structure: structure name
- In a data structure `UNION`: union name

Example

```
PROGRAM PROG1
VAR
    strPOU : STRING := __POUNAME();  //Yields 'PROG1'
    strlocalPOU : STRING;
END_VAR

    strlocalPOU := __POUNAME();     //Yields 'PROG1'
```

Operator 'TEST_AND_SET'

The multicore operator is an extension of the IEC 61131-3 standard.

The operator can be used for implementing a semaphore, for example to guarantee exclusive access to a variable written to by different tasks.

`TEST_AND_SET` gets a type `DWORD` variable as its input. Write access to this variable must be possible. The variable is set to `1` and the previous value is returned as the result.

The operation is atomic, which means that it cannot be interrupted by another task. This also applies to multicore platforms.

For example, the call in the program is `dwOldValue := TEST_AND_SET(dw);`, in which the variables `dwOldValue` and `dw` must be of data type `DWORD`.

Example

The following example shows a typical usage. Exclusive access to a type `STRING` variable, which is addressed via the `pstrOutput` pointer, should be implemented. The access to a string is not atomic. If multiple tasks write to the same string at the same time, then the contents may be inconsistent. With the `TEST_AND_SET` function, it is now possible to write the same `STRING` variable in different tasks.

```
FUNCTION ExclusiveStringWrite : BOOL
VAR_INPUT
    strToWrite : STRING;
    pstrOutput : POINTER TO STRING;
END_VAR
VAR_STAT
    dwSynch : DWORD;
END_VAR
VAR
    dwOldValue: DWORD;
END_VAR

dwOldValue := TEST_AND_SET(dwSynch);  // Write the 1 and read the
old value at the same time
IF dwOldValue = 0 THEN                // 0 means: no other task is
currently writing
   pstrOutput^ := strToWrite;         // Now you can write safely
on the string
   dwSynch := 0;                      // The DWORD must be reset
   ExclusiveStringWrite := TRUE;      // Writing was successful
ELSE
   ExclusiveStringWrite := FALSE;     // Writing was not successful
```

See also

- *Chapter 1.4.1.19.3.64 “Operator '__COMPARE_AND_SWAP” on page 625*
- *“Multicore operators” on page 546*

Operator - Global namespace

This operator is an extension of the IEC 61131-3 standard.

An instance path that begins with a dot (`.`) always opens a global namespace. If there is a local variable that has the same name `<varname>` as a global variable, then you refer to the global variable as `.<varname>`.

Operator - Namespace for global variables lists

This operator is an extension of the IEC 61131-3 standard.

You can use the name of a global variables list (GVL) as a namespace identifier for the variables that are defined in the list. This makes is possible to use variables with the same name in different global variables lists and still access specific variables uniquely. You use a dot (`.`) to prepend the name of the global variables list to the variable name.

```
<global variable list name>.<variable>
```

Example

```
globlist1.varx := globlist2.varx;
```

The `globlist1` and `globlist2` global variables lists each contain a `varx` variable. CODESYS copies the `varx` global variable from the `globlist2` list to `varx` in the `globlist1` list.

If you reference a variable that is declared in several global variables lists without referencing the prepended list name, then an error message is printed.

Operator - Library namespace

This operator is an extension of the IEC 61131-3 standard.

Syntax: `<library namespace>.<library identifier>`

Example: `LIB_A.FB_A`

A library module identifier is appended with the library namespace (as a prefix separated by a dot) for unique and qualified access to the library module. The namespace usually coincides with the library name.

Example

A library is included in a project and contains the module `FB_A`. However, the function block with the same name is already available locally in the project. Identify the library module as `LIB_A.FB_A` in order to access the library module, not the local function block.

```
var1 := FB_A(in := 12); // Call of the project function FB_A
var2 := LIB_A.FB_A(in := 22); // Call of the library function FB_A
```

You can define another identifier for the namespace. To do this, specify a namespace in the project information (library developers: when creating a library project). As an alternative, you can specify a specific namespace for a library in the library manager in the *“Properties”* dialog box (application developers: when creating an application).

See also

- *Chapter 1.4.1.16 “Using Libraries” on page 448*
- *Chapter 1.4.1.20.3.14.4 “Command 'Placeholders'” on page 1120*
- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*

Operator - Enumeration namespace

This operator is an extension of the IEC 61131-3 standard.

You can use the TYPE name of an enumeration for unique access to an enumeration constant. In this way, you can use the same constant names in different enumerations.

The enumeration name is prepended to the constant name with a dot (`.`).

`<enumeration name>.<constant name>`

Example

The constant `Blue` is a component of both the enumeration `Colors` and the enumeration `Feelings`.

```
color := Colors.Blue;  // Access to component blue in enumeration
Colors

feeling := Feelings.Blue; // Access to component blue in
enumeration Feelings
```

Operator '__POOL'

The operator is an extension of the IEC 61131-3 standard.

The operator is used to reference objects which are managed in the global POU pool (in the *“POUs”* view). The operator directly accesses objects in the *“POUs”* view.

Example

```
PROGRAM PLC_PRG
VAR
    svar_pou : STRING;
END_VAR

svar_pou := __POOL.POU();
```

See also

- *Chapter 1.4.1.19.8 “Shadowing Rules” on page 745*

Operator 'INI'

> *The `INI` operator is a CoDeSys V2.3 operator. In CODESYS V3, the `FB_init` method replaces the `INI` operator. You can still use this operator in projects that are imported from CoDeSys V2.3.*

The `INI` operator is used for initializing retain variables of a function block instance used in a POU.

Assign the operator to a Boolean variable.

Syntax:

```
<Boolean variable name> := INI <FB instance name> , <Boolean value> );
<Boolean value> : TRUE | FALSE
```

If the second parameter of the operator yields `TRUE`, then CODESYS initializes all retain variables that are defined in the function block `<FB instance name>`.

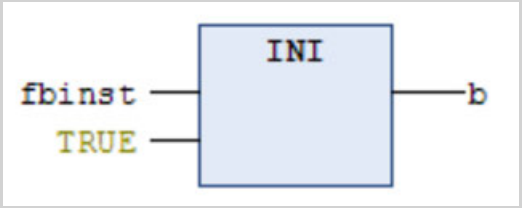
Examples

`fbinst` is the instance of the function block `fb1`, where the retain variable `retvar` is defined.

ST:

```
Declaration in the block:

VAR
    fbinst : fb1;
    b : BOOL;
END_VAR


Program part:

b := INI(fbinst, TRUE);
ivar := fbinst.retvar; (* => retvar is initialized *)
```

FBD



See also

- *Chapter 1.4.1.19.10 “Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'” on page 748*
- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

1.4.1.19.4 Operands

Constants and literals

Constants are identifiers for unchangeable values. You can declare constants locally within a POU or globally within a global variable list. The declaration segment is extended with the keyword `CONSTANT`.

Constants are also character strings that represent the value of a base type, such as integers or floating-point numbers (for example, `16#FFFF_FFFF`, `T#5s`, or `-1.234 E-5`). To distinguish between them, these constants are also called literals, literal constants, or unnamed constants. There are logical (`TRUE`, `FALSE`) or numeric literals (`3.1415`, `T#5s`), but also character string literals (`'Hello world!'`, `"black"`).

Syntax declaration

```
<scope> CONSTANT
    <identifier> : <data type> := <initial value> ;
END_VAR

<scope> : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
<data type>: <elementary data type | user defined data type |
function block >
<initial value> : literal value | identifier | expression
```

Allowed initial values:

- Literal (example: `TRUE`, `FALSE`, `16#FFFF_FFFF`)
- Named constant that was declared at another location
- Simple expression composed of literals, also combined with simple operators, such as + - *

Inputs or function calls cannot be specified as an initial value.

Example

```
VAR_GLOBAL CONSTANT
    g_ciMAX_A : INT := 100;
    g_ciSPECIAL : INT := g_ciMAX_A - 10;
END_VAR
```

Constants are defined only for the declaration. The assignment of an initial value is required. Within an implementation, constants are only read and therefore always appear on the right of the assignment operator in a statement.

The constants are replaced with the initial value when the code is compiled. It also has to be possible to calculate the initial value at compile time.

Constants of structured or user-defined types are calculated not until runtime. Structured constants in programs or GVLs are calculated one time at program start. Structured constants in functions or methods are calculated every time the function or method is called. Therefore, the initialization of structured constants can depend on inputs or execute function calls.

See also

- *Chapter 1.4.1.19.4.1 “BOOL constants” on page 633*
- *Chapter 1.4.1.19.4.2 “Numeric constants” on page 633*
- *Chapter 1.4.1.19.4.3 “REAL/LREAL constants” on page 634*
- *Chapter 1.4.1.19.4.4 “String Constants” on page 634*
- *Chapter 1.4.1.19.4.6 “Date and Time Constants” on page 637*
- *Chapter 1.4.1.19.4.5 “TIME/LTIME Constant” on page 635*
- *Chapter 1.4.1.19.4.7 “Typed literals” on page 640*

Variables

You can declare variables as either local in the declaration part of a POU or in a global variable list. The allowed location of a variable depends on its data type.

See also

- *Chapter 1.4.1.19.4.8 “Access to Variables in Arrays, Structures, and Blocks” on page 641*
- *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*

Other

- *Chapter 1.4.1.19.4.10 “Addresses” on page 643*
- *Chapter 1.4.1.19.4.11 “Functions” on page 645*

See also

- *Chapter 1.4.1.8.2 “Declaration of Variables ” on page 222*
- *Chapter 1.4.1.8.5 “Using input assistance” on page 260*
- *Chapter 1.4.1.19.2.11 “Constant Variables - 'CONSTANT'” on page 534*
- *Chapter 1.4.1.19.1.3.3 “ST expressions” on page 464*

BOOL constants

BOOL constants are the truth values `TRUE` (1) and `FALSE` (0).

See also

- *Chapter 1.4.1.19.5.1 “Data type 'BOOL'” on page 647*

Numeric constants

Numeric values can be binary, octal, decimal, and hexadecimal numbers. If an integer value is not a decimal number, then you must write its base followed by the number sign (#) before the integer constant. You enter the hexadecimal digit values for the numbers 10 to 15 as usual with the letters A-F.

You can use an underscore within a numeric value.

Examples:

`14`	decimal number
`2#1001_0011`	binary number
`8#67`	octal number
`16#A`	hexadecimal number
`DINT#16#A1`	typed data type `DINT#` and base `16#` combined

This type of numeric value can be `BYTE`, `WORD`, `DWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `REAL`, or `LREAL`.

Implicit conversions from "larger" to "smaller" types are not permitted. You cannot simply use a DINT variable as an INT variable. For this, you have to use a type conversion function.

See also

- *Chapter 1.4.1.19.3 “Operators” on page 542*
- *Chapter 1.4.1.19.4.7 “Typed literals” on page 640*

As number constants basically are treated as integers, in divisions you must enter a constant in the format of a floating-point number in order not to loose the remainder. For example: Division `1/10` results in `0`, division `1.0/10` results in `0.1`.

REAL/LREAL constants

You can specify floating-point numbers as `REAL` and `LREAL` constants either in decimal notation or exponential notation with mantissa and exponent The decimal point serves as the decimal separator according to the International System of Units (English).

Syntax of exponential notation

```
<significand> e | E <exponent>

exponent : -44..38 // REAL
exponent : -324..308 // LREAL
```

Example

Table 38: `REAL` literal

7.4	Decimal number. 7,4 with a comma returns a compiler error
1/3.0	Decimal fraction for 0.333333343 Note: In the case of division of integer types, the result remains an integer type. In this case, the value is rounded. For example, 1/3 yields 0 as the result.
1.64e+009	Exponential notation
-3.402823e+38	Smallest number
-1E-44	Largest negative number
1.0E-44	Smallest positive number
3.402823e+38	Largest number

Table 39: `LREAL` literal

-1.7976931348623157E+308	Smallest number
-4.94065645841247E-324	Largest negative number
4.94065645841247E-324	Smallest positive number
1.7976931348623157E+308	Largest number

See also

- *Chapter 1.4.1.19.5.3 “Data type 'REAL' / 'LREAL'” on page 648*

String Constants

A string constant is a character string enclosed in single straight quotation marks. The characters are coded according to the character set specified in ISO/IEC 8859-1. Therefore, a string constant can include spaces and accented characters, as these belong to this character set. This is also referred to as a string literal, or simply a string.

Example: `'Hello world!'`

When a dollar sign ($) is in a string constant, the following two characters are interpreted as a hexadecimal code according to the coding in ISO/IEC 8859-1. The code also corresponds to ASCII code. In addition, please note the special cases.

Table 40: Hexadecimal code

String with $ code	Interpretation
`'$<8-bit code>'`	8-bit code: Two-digit hexadecimal number that is interpreted according to ISO/IEC 8859-1.
`'$41'`	A
`'$A9'`	©
`'$40'`	@
`'$0D'`	Control character: Line break (corresponds to '$R')
`'$0A'`	Control character: New line (corresponds to '$L' and '$N')

Table 41: Special cases

String with $ code	Interpretation
`'$L','$l'`	Control character: Line feed (corresponds to `'$0A'`)
`'$N','$n'`	Control character: New line (corresponds to `'$0A'`)
`'$P','$p'`	Control character: Form feed
`'$R','$r'`	Control character: Line break (corresponds to `'$0D'`)
`'$T','$t'`	Control character: Tab
`'$$'`	Dollar sign: $
`'$''`	Single straight quotation mark: '

Example Constant declaration

```
VAR CONSTANT
    constA : STRING := 'Hello world';
    constB : STRING := 'Hello world $21'; // Hello world!
END_VAR
```

TIME/LTIME Constant

You can use `TIME` constants to operate the standard timer modules. The constant has a size of 32 bits and a resolution in milliseconds.

In addition, the time constant `LTIME` is available as a time basis for high-resolution timers. The `LTIME` constant has a dimension of 64 bits and a resolution in nanoseconds.

TIME constant

Syntax

```
<time keyword> # <length of time>

<time keyword> : TIME | time | T | t
<length of time> : ( <number of days>d )? ( <number of hours>h )?
( <number of minutes>m )? ( <number of seconds>s )? (<number of
milliseconds>ms)? // ( ...)? Optional
```

The order of time units must not be changed. However, it is not required to specify all units. It is permitted to specify the units in uppercase.

Time units

- `D | d`: Days
- `H | h`: Hours
- `M | m`: Minutes
- `s | s`: Seconds
- `MS | ms`: Milliseconds

Examples

Correct time constants of an ST assignment

```
VAR
    timLength : TIME := T#14ms;
    timLength1 : TIME := T#100s12ms; // Overflow in the highest
unit is allowed.
    timLength2 : TIME := T#12h34m15s;
    timCompare : TIME;
    xIsOK: BOOL;

    timLongest := T#49D17H2M47S295MS; // 4294967295
END_VAR

IF timLength < T#15MS THEN
    IF timCompare < timLength1 THEN
        xIsOK := TRUE;
    END_IF;
END_IF
```

Table 42: Incorrect usage:

`timIncorrect := t#5m68s;`	Overflow at a lower position
`timIncorrect1 := 15ms;`	Time marker `T#` missing
`timIncorrect2 := t#4ms13d;`	Incorrect order of time units

LTIME constant

Syntax

```
<long time keyword> # <length of high resolution time>

<long time keyword> : LTIME | ltime
<length of high resolution time> : <length of time> ( <number of
microseconds>us )?  ( <number of nanoseconds>ns )? // ( ...)? Optional
```

You can use the same units for `LTIME` constants as for `TIME` constants. You can also specify microseconds and nanoseconds because the specified time is calculated in higher time resolution. `LTIME` literals are treated internally as data type `LWORD` and therefore the value resolved in nanoseconds.

Additional time units

- `US | us`: Microseconds
- `NS | ns`: Nanoseconds

Examples of correct usage of an ST assignment:

```
PROGRAM PLC_PRG
VAR
    ltimLength := LTIME#1000d15h23m12s34ms2us44ns;
    ltimLength1 := LTIME#3445343m3424732874823ns;
END_VAR
```

See also

- *Chapter 1.4.1.19.5.5 “Data Type 'TIME'” on page 649*
- *Chapter 1.4.1.19.4.6 “Date and Time Constants” on page 637*

Date and Time Constants

32-bit date specifications 'DATE'

Use the keyword `DATE` (`D`) to specify a date.

Syntax

```
<date keyword>#<year>-<month>-<day>

<date keyword> : DATE | date | D | d
<year> : 1970-2106
<month> : 1-12
<day> : 1-31
```

`DATE` literals are treated internally as data type `DWORD`, which corresponds to an upper limit of `DATE#2106-2-7`.

Example

```
PROGRAM PRG_Date
VAR
    dateStart : DATE := DATE#2018-8-8;
    dateEnd : DATE := D#2018-8-31;
    dateCompare: DATE := date#1996-05-06;
    xIsDuringTheTime: BOOL;

    dateEarliest : DATE := d#1970-1-1; // = 0
    dateLatest : DATE := DATE#2106-2-7; // = 4294967295
END_VAR

IF dateStart < dateCompare THEN
    IF dateCompare < dateEnd THEN
        xIsDuringTheTime := TRUE;
    END_IF;
END_IF
```

64-bit date specifications 'LDATE'

Use the keyword `LDATE` (`LD`) to specify a date.

Syntax

```
<date keyword>#<year>-<month>-<day>

<date keyword> : LDATE | ldate | LD | ld
<year> : 1970-2262
<month> : 1-12
<day> : 1-31
```

`LDATE` literals are treated internally as data type `LWORD`, which corresponds to an upper limit of `DATE#2554-7-21`.

Example

```
PROGRAM PRG_Ldate
VAR
    ldateStart : LDATE := LDATE#2018-8-8;
    ldateEnd : LDATE := ldate#2018-8-31;
    ldateCompare: LDATE := LD#1996-05-06;
    xIsDuringTheTime: BOOL;

    ldateEarliest : LDATE := ld#1970-1-1; // = 0
    ldateLatest : LDATE := LDATE#2262-4-10; // = 16#7FFF63888C620000

    lwValue: LWORD;
END_VAR

IF ldateStart < ldateCompare THEN
    IF ldateCompare < ldateEnd THEN
        xIsDuringTheTime := TRUE;
    END_IF;
END_IF
lwValue := LDATE_TO_LWORD(ldateCompare);
```

32-bit date and time specifications 'DATE_AND_TIME'

Use the keyword `DATE_AND_TIME` (`DT`) to specify a date and time.

Syntax

```
<date and time keyword>#<date and time value>

<date and time keyword> : DATE_AND_TIME | date_and_time | DT | dt
<date and time value> : <year>-<month>-<day>-<hour>:<minute>:<second>
<year> : 1970-2106
<month> : 1-12
<day> : 1-31
<hour> : 0-24
<minute> : 0-59
<second> : 0-59
```

`DATE_AND_TIME` literals are treated internally as data type `DWORD`. The time is processed in seconds and as a result can take on values from January 1, 1970 00:00 to February 7, 2106 06:28:15.

Example

```
PROGRAM PLC_PRG
VAR
    dtDate : DATE_AND_TIME := DATE_AND_TIME#1996-05-06-15:36:30;
    dtDate1: DATE_AND_TIME := DT#1972-03-29-00:00:00;
    dtDate2: DATE_AND_TIME := DT#2018-08-08-13:33:20.5;

    dtEarliest : DATE_AND_TIME :=
DATE_AND_TIME#1979-1-1-00:00:00; // 0
    dtLatest : DATE_AND_TIME := DATE_AND_TIME#2106-2-7-6:28:15; //
4294967295
END_VAR
```

64-bit date and time specifications 'LDATE_AND_TIME'

Use the keyword `LDATE_AND_TIME` (`LDT`) to specify a date and time.

Syntax

```
<date and time keyword>#<long date and time value>

<date and time keyword> : LDATE_AND_TIME | ldate_and_time | LDT | ldt
<date and time value> : <year>-<month>-<day>-<hour>:<minute>:<second>
<year> : 1970-2106
<month> : 1-12
<day> : 1-31
<hour> : 0-24
<minute> : 0-59
<second> : 0-59 LDATE_AND_TIME#2262-4-10-23:59:59.99999999
```

`DATE_AND_TIME` literals are treated internally as data type `LWORD`. The time is processed in seconds and as a result can take on values from January 1, 1970 00:00 to July 21, 2554 23:59:59.999999999.

Example

```
PROGRAM PLC_PRG
VAR
    ldtDate : LDATE_AND_TIME := LDATE_AND_TIME#1996-05-06-15:36:30;
    ldtDate1: LDATE_AND_TIME := LDT#1972-03-29-00:00:00;
    ldtDate2: LDATE_AND_TIME := LDT#2018-08-08-13:33:20.5;

    dtEarliest : LDT := LDT#1979-1-1-00:00:00; // 0
    dtLatest : LDT := LDT#2266-4-10-23:59:59; // =
16#7FFF63888C620000
END_VAR
```

32-bit time specifications 'TIME_OF_DAY'

Use the keyword `TIME_OF_DAY` (`TOD`) to specify a time.

Syntax

```
<time keyword>#<time value>

<time keyword> : TIME_OF_DAY | time_of_day | TOD | tod
<time value> : <hour>:<minute>:<second>
<hour> : 0-23
<minute> : 0-59
<second> : 0.000-59.999
```

You can also specify fractions of a second. `TIME_OF_DAY` literals are treated internally as `DWORD` and the value is resolved in milliseconds.

Examples

```
PROGRAM POU
VAR
    todClockTime : TIME_OF_DAY :=  TIME_OF_DAY#15:36:30.123;
    todEarliest : TIME_OF_DAY := TIME_OF_DAY#0:0:0.000;
    todLatest : TOD := TOD#23:59:59.999;
END_VAR
```

64-bit time specifications 'LTIME_OF_DAY'

Use the keyword `LTIME_OF_DAY` (`LTOD`) to specify a time.

Syntax

```
<time keyword>#<time value>

<time keyword> : LTIME_OF_DAY | ltime_of_day | LTOD | ltod
<time value> : <hour>:<minute>:<second>
<hour> : 0-23
<minute> : 0-59
<second> : 0.000-59.999999999
```

You can also specify fractions of a second. `LTIME_OF_DAY` literals are treated internally as `LWORD` and the value is resolved in nanoseconds.

Examples

```
PROGRAM POU
VAR
    ltodClockTime : LTIME_OF_DAY :=  TIME_OF_DAY#15:36:30.123456789;
    todEarliest : TIME_OF_DAY := TIME_OF_DAY#0:0:0;
    todLatest : TOD := TOD#23:59:59.999999999;
END_VAR
```

See also

- *Chapter 1.4.1.19.5.7 “Date and Time Data Types” on page 650*

Typed literals

With the exception of REAL/LREAL constants (LREAL is always used here), CODESYS uses the smallest possible data type when calculating with IEC constants. If you want to use another data type, then you can use typed literals without having to declare the constants explicitly. When doing this, provide the constants with a prefix that indicates the type.

Syntax:

```
<type>#<literal>
```

<type> defines the desired data type; possible values: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL. You must capitalize the entire type name.

<literal> defines the constants. The entry must match the data type defined in <Type>.

Example:

```
var1 := DINT#34;
```

If CODESYS cannot convert the constant into the target type without data loss, then an error message is issued.

You can use typed constants wherever you can use normal constants.

Access to Variables in Arrays, Structures, and Blocks

Syntax for access to

- Two-dimensional array component: <array name> [<1st dimension> , <2nd dimension>]
- Structural variable: <structure name> . <component name>
- Function block and program variable: <function block name> | <program name> . <variable name>

See also

- *Chapter 1.4.1.19.5.14 “Data Type 'ARRAY'” on page 660*
- *Chapter 1.4.1.19.5.16 “Structure” on page 674*
- *Chapter 1.4.1.20.2.18.2 “Object 'Function Block'” on page 883*
- *Chapter 1.4.1.20.2.18.1 “Object 'Program'” on page 882*

Bit Access in Variables

NOTICE!

Implement concurrent bit access by two tasks only if the processor can execute bit access directly on the memory. All x86 and x64 systems have commands for bit access in memory. Systems such as ARM and PPC cannot access bits directly in the memory.

If two tasks execute bit access simultaneously, even though the processor cannot perform bit access directly in the memory, then proceed as follows. Use a semaphore (SysSemEnter) or a similar technique to prevent competing bit access. However, it is best to execute the bit access within a task.

With index access, individual bits can be addressed in integer variables. Using a structure variable or a function block instance, individual bits can be addressed symbolically.

Index access to bits integer variables

You can address individual bits in integer variables. To do this, append the variable with a dot and the index of the addressed bit. The bit-index can be given by any constant. Indexing is 0-based.

Syntax

```
<integer variable name> . <index>
<integer data type> = BYTE | WORD | DWORD | LWORD | SINT | USINT |
INT | UINT | DINT | UDINT | LINT | ULINT
```

Example

In the program, the third bit of the variable `wA` is set to the value of variable `xB`. The constant `c_usiENABLE` acts as an index to access the third bit of the variable `iX`.

Index access

```
PROGRAM PLC_PRG
VAR
    wA : WORD := 16#FFFF;
    xB : BOOL := 0;
END_VAR

// Index access in an integer variable
wA.2 := xB;
```

Result: `wA = 2#1111_1111_1111_1011 = 16#FFFB`

Constant as index

```
// GVL declaration
VAR_GLOBAL CONSTANT
    gc_usiENABLE : USINT := 2;
END_VAR

PROGRAM PLC_PRG
VAR
    iX : INT := 0;
END_VAR

// Constant as index
iX.gc_usiENABLE :=  TRUE;     // Third bit in iX is set TRUE
```

Result: `iX = 4`

Symbolic bit access in structure variables

With the `BIT` data type, you can combine individual bits into a structure and then access them individually. Then the bit is addressed with the component name.

Example

Type declaration of the structure:

```
TYPE S_CONTROLLER :
STRUCT
    bitOperationEnabled : BIT;
    bitSwitchOnActive : BIT;
    bitEnableOperation : BIT;
    bitError : BIT;
    bitVoltageEnabled : BIT;
    bitQuickStop : BIT;
    bitSwitchOnLocked : BIT;
    bitWarning : BIT;
END_STRUCT
END_TYPE
```

Declaration and write access to a bit:

```
PROGRAM PLC_PRG
VAR
    ControlDriveA : S_CONTROLLER;
END_VAR

// Symbolic bit access to bitEnableOperation
ControlDriveA.bitEnableOperation := TRUE;
```

Symbolic bit access in function block instances

In function blocks, you can declare variables for individual bits.

Example

```
FUNCTION_BLOCK FB_Controller
VAR_INPUT
    bitSwitchOnActive : BIT;
    bitEnableOperation : BIT;
    bitVoltageEnabled : BIT;
    bitQuickStop : BIT;
    bitSwitchOnLocked : BIT;
END_VAR
VAR_OUTPUT
    bitOperationEnabled : BIT;
    bitError : BIT;
    bitWarning : BIT;
END_VAR
VAR
END_VAR
;

PROGRAM PLC_PRG
VAR
    fbController : FB_Controller;
END_VAR
// Symbolic bit access to bitSwitchOnActive
fbController(bitSwitchOnActive := TRUE);
```

See also

- *Chapter 1.4.1.19.5.2 “Integer data types” on page 647*
- *“Symbolic bit access in structure variables” on page 675*
- *Chapter 1.4.1.19.5.10 “Data Type 'BIT'” on page 656*

Addresses

CAUTION!
If you use pointers to addresses, then the contents of addresses can be moved during an online change. If you use absolute addresses, then the contents of addresses does not change during an online change.

Syntax:

```
%<memory area prefix> ( <size prefix> )? <memory position>

<memory area prefix> : I | Q | M
<size prefix> : X | B | W | D
<memory position> : <number> ( .<number> )* // Depends on the target
system
```

When defining an address, you use specific character strings to express memory position and size. An address is marked with the percent sign (%), followed by the memory range prefix, the optional size prefix, and the memory range position. The numbering that you use for addressing the memory position depends on the target system.

Memory Range Prefix	
I	Input memory range for "Inputs" For physical inputs via input drivers, "Sensors"
Q	Output memory range for "Outputs" For physical outputs via output drivers, "Actuators"
M	Flag memory range

Size Prefix	Data Type	Data Width
No size prefix		Single bit
X		Single bit
B	BYTE	8 bits
W	WORD	16 bits
D	DWORD	32 bits

Examples

%QX7.5 %Q7.5	Single bit address of the output bit 7.5
%IW215	Word address of the input word 215
%QB7	Byte address of the output byte 7
%MD48	Address of a double word at memory position 48 in flag memory
%IW2.5.7.1	Word address of an input word; interpretation dependent on the current controller configuration
VAR wVar AT %IW0 : WORD; END_VAR	Variable declaration with address information of an input word
VAR xActuator AT %QW0 : BOOL; END_VAR	Boolean variable declaration Note: For Boolean variables, one byte is allocated internally if a single bit address is not specified. A change in the value of xActuator affects the range from QX0.0 to QX0.7.
VAR xSensor AT IX7.5 : BOOL; END_VAR	Boolean variable declaration with explicit specification of a single bit address. On access, only the input bit 7.5 is read.

Memory position

Make sure that the address is valid as follows:

To map a valid address in an application, you must know the required position (applicable memory range) in the process image: input memory range (I), output memory range (Q), and flag memory range (M) — see above. Furthermore, you have to specify the required size prefix: bit, BYTE, WORD, DWord (see above: X, B, W, D)

The current device configuration and device settings (hardware structure, device description, I/O settings) play a decisive part. Note specifically the differences in the interpretation of bit addresses for devices with "byte addressing mode" and devices with "word-oriented IEC addressing mode". For example, in a byte addressing device, the number before the point of bit address `%IX5.5` addresses byte 5. On the other hand, in a word-addressed device, it addresses word 5. In contrast, addressing with a word or byte address is independent of the device type: with `%IW5` always word 5 is addressed and with byte address `%IB5` always byte 5. Regardless of size and addressing mode, you can address different memory cells therefore with the same address information.

The following table shows the comparison of byte addressing and word-oriented IEC addressing for bits, bytes, words, and double words. It also shows the overlapping memory ranges that are present in the case of byte addressing (see also the example below the table).

Regarding syntax, note that the IEC addressing mode is always word-oriented. In this case, the word number is located before the point and the bit number ofter the point.

DWords / **W**ords				**B**ytes	**X** (bits)					
byte addressing		word oriented IEC addressing			byte addressing			word oriented IEC addressing		
D0	W0	D0	W0	B0	X0.7	...	X0.0	X0.7	...	X0.0
D1	W1			B1	X1.7	...	X1.0	X0.15	...	X0.8
...	W2		W1	B2	...			X1.7	...	X1.0
	W3			B3				X1.15	...	X1.8
	W4	D1	W2	B4						
	...			B5						
			W3	B6						
				B7						
		D2	W4	B8						
		...		...						
		...		...						
		...		...						
D(n-3)		D(n/4)	...							
	W(n-1)		W(n/2)							
				B**n**	Xn.7	...	Xn.0	X(n/2).15	...	X(n/2).8

n = byte number

Example of memory range overlapping in the case of the byte addressing mode

D0 contains B0 - B3, W0 contains B0 and B1, W1 contains B1 and B2, and W2 contains B2 and B3. Consequently, in order to avoid overlap, you must not use W1 (also D1, D2, and D3) for addressing.

See also

- *Chapter 1.4.1.8.11.2 “AT declaration” on page 281*

Functions

In ST, you can use a function call as an operand.

Example:

```
Result := Fct(7) + 3;
```

TIME() function

This function yields the time (in milliseconds) that has elapsed since system boot.

The data type is TIME.

Example in ST:

```
systime := TIME();
```

See also

- *Chapter 1.4.1.20.2.18.3 “Object 'Function'” on page 886*

1.4.1.19.5 Data Types

In the programming, a variable is identified by its name and has an address in the memory of the target system. Accordingly, variable names are identifiers under which the allocated memory is addressed. The size of the variable is determined by its data type. This determines how much memory is reserved for the variable and how the values in memory are to be interpreted. The data type also determines which operators are allowed.

In CODESYS, there is also the capability of instantiating function blocks. Function block instances then use memory like variables do. The memory requirement is determined by the function block.

The following groups of data types are available.

Standard data types

A standard data type (or standard data type) is an elementary data type or a string data type.

```
<standard data type> : __UXINT | __XINT | __XWORD | BIT | BOOL |
BYTE | DATE | DATE_AND_TIME | DINT | DT | DWORD | INT | LDATE |
LDATE_AND_TIME | LDT | LINT | LREAL | LTIME | LTOD | LWORD | REAL |
SINT |STRING | TIME | TOD | TIME_OF_DAY | UDINT | UINT | ULINT | USINT
| WORD | WSTRING
```

See also

- *Chapter 1.4.1.19.5.10 “Data Type 'BIT'” on page 656*
- *Chapter 1.4.1.19.5.1 “Data type 'BOOL'” on page 647*
- *Chapter 1.4.1.19.5.2 “Integer data types” on page 647*
- *Chapter 1.4.1.19.5.11 “Special Data Types '__UXINT', __XINT, and '__XWORD'” on page 656*
- *Chapter 1.4.1.19.5.3 “Data type 'REAL' / 'LREAL'” on page 648*
- *Chapter 1.4.1.19.5.4 “Data Type 'STRING'” on page 649*
- *Chapter 1.4.1.19.5.9 “Data type 'WSTRING'” on page 655*
- *Chapter 1.4.1.19.5.5 “Data Type 'TIME'” on page 649*
- *Chapter 1.4.1.19.5.6 “Data Type 'LTIME'” on page 650*
- *Chapter 1.4.1.19.5.7 “Date and Time Data Types” on page 650*

Extensions of the IEC 61131-3 standard

See also

- *Chapter 1.4.1.19.5.10 “Data Type 'BIT'” on page 656*
- *Chapter 1.4.1.19.5.12 “Pointers” on page 656*
- *Chapter 1.4.1.19.5.19 “Data type 'UNION'” on page 681*
- *Chapter 1.4.1.19.5.15 “Data Type '__VECTOR'” on page 666*

User-defined data types

You can declare your own data types which are based on the default predefined data types or existing data types.

These kinds of data types are called user-defined or user-specific. The data types are either organized as its own DUT object or declared within the declaration part of a programming object. Moreover, they are differentiated according to their purpose and syntax.

User-Defined Data Type	Declaration	See also
Alias	DUT object	*Chapter 1.4.1.19.5.18 “Alias” on page 680*
Arrays	Programming object	*Chapter 1.4.1.19.5.14 “Data Type 'ARRAY'” on page 660*
Enumeration	DUT object, programming object	*Chapter 1.4.1.19.5.17 “Enumerations” on page 676*
Reference	Programming object	*Chapter 1.4.1.19.5.13 “Reference” on page 658*
Pointer	Programming object	*Chapter 1.4.1.19.5.12 “Pointers” on page 656*
Structure	DUT object	*Chapter 1.4.1.19.5.16 “Structure” on page 674*
Subrange type	Programming object	*Chapter 1.4.1.19.5.20 “Subrange types” on page 681*
Union	DUT object	*Chapter 1.4.1.19.5.19 “Data type 'UNION'” on page 681*
Vector	DUT object	*Chapter 1.4.1.19.5.15 “Data Type '__VECTOR'” on page 666*

NOTICE!
Note the recommendations for naming an identifier.

See also

- *Chapter 1.4.1.19.7 “Identifiers” on page 740*

Data type 'BOOL'

Data Type	Values	Memory
BOOL	TRUE (1), FALSE (0)	8 bit

See also

- *Chapter 1.4.1.19.4.1 “BOOL constants” on page 633*

Integer data types

CODESYS provides the following integer data types.

Data Type	Lower Limit	Upper Limit	Memory
BYTE	0	255	8 bit
WORD	0	65535	16 bit
DWORD	0	4294967295	32 bit
LWORD	0	2^{64}-1	64 bit
SINT	-128	127	8 bit
USINT	0	255	8 bit
INT	-32768	32767	16 bit

Data Type	Lower Limit	Upper Limit	Memory
UINT	0	65535	16 bit
DINT	-2147483648	2147483647	32 bit
UDINT	0	4294967295	32 bit
LINT	-2^{63}	$2^{63}-1$	64 bit
ULINT	0	$2^{64}-1$	64 bit

NOTICE!
Information can be lost when converting from larger to smaller types.

See also

- *Chapter 1.4.1.19.4.2 “Numeric constants” on page 633*

Data type 'REAL' / 'LREAL'

The data types `REAL` and `LREAL` are floating-point types according to IEEE 754. They are necessary when using decimal numbers and floating-point numbers in decimal notation or exponential notation.

Table 43: Target system: CODESYS Control Win V3

Data type	Smallest value number	Largest value number	Storage space
`REAL`	1.0E-44	3.402823E+38	32 bit
`LREAL`	4.94065645841247E-324	1.7976931348623157E+308	64 bit

Example

```
PROGRAM PLC_PRG
VAR
    rMax: REAL := 3.402823E+38; // Largest number
    rPosMin : REAL := 1.0E-44; // Smallest positive number
    rNegMax: REAL := -1.0E-44; // Largest negative number
    rMin: REAL := -3.402823E+38; // Smallest number

    lrMax: LREAL := 1.7976931348623157E+308; // Largest number
    lrPosMin : LREAL := 4.94065645841247E-324; // Smallest positve
number
    lNegMax: LREAL := -4.94065645841247E-324; // Largest negative
number
    lrMin: LREAL := -1.7976931348623157E+308; // Smallest number
END_VAR
```

NOTICE!
Support for the `LREAL` data type depends on the target device in use. Refer to the respective documentation as to whether or not the 64-bit type `LREAL` is converted to `REAL` or remains as `LREAL` when compiling the application. Conversion may result in the loss of information.

NOTICE!
If the value of the REAL/LREAL number is outside of the value range of the integer, then an undefined result is yielded from a data type conversion from REAL or LREAL to SINT, USINT, INT, UINT, DINT, UDINT, LINT, or ULINT. The result depends on the target system. An exception error is also possible. To get code that is independent of the target system, the application must catch value range violations.

If the REAL/LREAL number is within the value range of the integer data type, then the conversion operates the same way on all systems.

See also

- *Chapter 1.4.1.19.4.3 “REAL/LREAL constants” on page 634*

Data Type 'STRING'

A variable of data type STRING can have contain any character string. The amount of memory that is reserved during a declaration refers to characters and is shown in parentheses or brackets. If a size is not defined, then CODESYS allocates 80 characters by default.

As a rule, CODESYS does not limit the string length. However, the string function processes lengths of 1–255 only. If a variable is initialized with a string that is too long for the data type, then CODESYS truncates the string accordingly from the right.

NOTICE!
The memory required for a STRING variable is always one byte per character plus one additional byte (for example, 81 bytes for a "STRING(80)" declaration).

Example of a string declaration with 35 characters:

```
str : STRING(35):= 'This is a String';
```

See also

- *Chapter 1.4.1.19.4.4 “String Constants” on page 634*
- *Chapter 1.4.1.19.5.9 “Data type 'WSTRING'” on page 655*

Data Type 'TIME'

The data type is treated internally as DWORD. TIME is resolved in milliseconds.

Data type	Lower limit	Upper limit	Storage space	Resolution
TIME	T#0d0h0m0s0ms	T#49d17h2m47s295ms	32 bit	Milliseconds

See also

- *Chapter 1.4.1.19.5.6 “Data Type 'LTIME'” on page 650*
- *Chapter 1.4.1.19.4.5 “TIME/LTIME Constant” on page 635*
- *Chapter 1.4.1.19.4.6 “Date and Time Constants” on page 637*

Data Type 'LTIME'

Data Type 'LTIME'

You can use the data type LTIME as a time base for high-resolution timer. A high-resolution timer has a resolution in nanoseconds.

Data Type	Lower Limit	Upper Limit	Memory
LTIME	LTIME#0NS	LTIME#213503D23H34M33S709MS551US615NS	64 bits

Syntax:

```
LTIME#<long time declaration>
```

The time declaration can include units of time that apply for the TIME constant as well as:

- "US": microseconds
- "NS": nanoseconds

Example:

```
LTIME1 := LTIME#1000D15H23M12S34MS2US44NS
```

See also

- *Chapter 1.4.1.19.5.5 “Data Type 'TIME'” on page 649*
- *Chapter 1.4.1.19.3.40 “Time Conversion” on page 595*

Date and Time Data Types

The data types `DATE`, `DATE_AND_TIME` (`DT`), and `TIME_OF_DAY` (`TOD`) are handled internally like a `DWORD` (32-bit value).

The data types `LDATE`, `LDATE_AND_TIME` (`LDT`), and `LTIME_OF_DAY` (`LTOD`) are treated internally like an `LWORD` (64-bit value).

The values of these data types are measured in seconds, milliseconds, and nanoseconds since 01/01/1970.

Data Type	Lower Limit	Upper Limit	Memory	Resolution
`DATE`	`DATE#1970-01-01` `D#1970-01-01`	`DATE#2106-02-07` `D#2106-02-07`	32-bit	Seconds (although only the day is displayed)
`DATE_AND_TIME` `DT`	`DATE_AND_TIME#1970-1-1-0:0:0` `DT#1970-1-1-0:0:0`	`DATE_AND_TIME#2106-02-07-06:28:15` `DT#2106-02-07-06:28:15`	32-bit	Seconds
`TIME_OF_DAY` `TOD`	`TIME_OF_DAY#0:0:0` `TOD#0:0:0`	`TIME_OF_DAY#23:59:59.999` `TOD#23:59:59.999`	32-bit	Milliseconds

Data Type	Lower Limit	Upper Limit	Memory	Resolution
LDATE	LDATE#1970-1-1 LD#1970-1-1	LDATE#2554-7-21 LD#2554-7-21	64-bit	Nanosec-onds (although only the day is displayed)
LDATE_AND_TIME LDT	LDATE_AND_TIME#1970-1-1-0:0:0 LDT#1970-1-1-0:0:0	LDATE_AND_TIME#2554-7-21:23:59:59.999999999 LDT#2554-7-21-23:59:59.99999999	64-bit	Nanosec-onds
LTIME_OF_DAY LTOD	LTIME_OF_DAY#0:0:0 LTOD#0:0:0	LTIME_OF_DAY#23:59:59.999999999 LTOD#23:59:59.999999999	64-bit	Nanosec-onds

Example

```
VAR
    //Date
    dateBottom : DATE :=  DATE#1970-1-1;
    dateTop : DATE :=  DATE#2106-2-7;
    dateAppointment : DATE := D#2020-2-7; // D prohibited

    //Date and time
    dtBottom : DATE_AND_TIME :=  DATE_AND_TIME#1970-1-1-0:0:0;
    dtTop : DT :=  DATE_AND_TIME#2106-02-07-06:28:15;
    dtAppointment : DT := DT#2020-2-7-12:55:1.234;

    //Time of day
    todBottom : TIME_OF_DAY :=  TIME_OF_DAY#0:0:0;
    todTop : TOD :=  TIME_OF_DAY#23:59:59.999;
    todAppointment : TOD := TOD#12:3:4.567;

    // Long date
    ldateBottom : LDATE :=  LDATE#1970-1-1;
    ldateTop : LDATE :=  LDATE#2106-2-7;
    ldateAppointment : LDATE := LD#2020-2-7; // LD prohibited

    // Long date and time
    ldtBottom : LDATE_AND_TIME :=  LDATE_AND_TIME#1970-1-1-0:0:0;
    ldtTop : LDT :=  LDATE_AND_TIME#2262-4-10-23:59:59.99999999;
    ldtAppointment : LDT := LDT#2020-2-7-12:55:1.234567891;

    //Long time of day
    ltodBottom : LTIME_OF_DAY :=  LTIME_OF_DAY#0:0:0;
    ltodTop : LTOD :=  LTIME_OF_DAY#23:59:59.999999999 ;
    ltodAppointment : LTOD := LTOD#12:3:4.567890123;

END_VAR
```

See also

- *Chapter 1.4.1.19.4.6 “Date and Time Constants” on page 637*

Data Type 'ANY' and 'ANY_<type>'

The data types ANY or ANY_<type> are used in interfaces of functions, function blocks, or methods in order to type input parameters whose type is unknown or unspecified: The input variables (VAR_INPUT) have a generic data type.

The compiler replaces the type of input variable internally with the data structure described below, whereby the value is not passed directly. Instead, a pointer is passed to the actual value so only a variable can be passed. Therefore, the data type is only specified when it is called. As a result, calls of such POUs can be made using arguments which each have different data types.

*Literals, replaced constants, and results of function calls or expressions **cannot** be passed to input variables (`VAR_IN_OUT`).*

Internal data structure for 'ANY' and 'ANY_<type>'

When code is compiled, the input variables are typed **internally** with `ANY` data type by the following structure. When the POU is called (at runtime), the argument is passed to a reference parameter.

```
TYPE AnyType :
STRUCT
        // the type of the actual parameter
    typeclass : __SYSTEM.TYPE_CLASS ;
        // the pointer to the actual parameter
    pvalue : POINTER TO BYTE;
        // the size of the data, to which the pointer points
    diSize : DINT;
END_STRUCT
END_TYPE
```

You can access the input variable within the POU via this structure by means of this structure, and for example query the passed value.

Example This compares whether or not two input variables have the same type and the same value.

```
FUNCTION funGenericCompare : BOOL
VAR_INPUT
    any1 : ANY;
    any2 : ANY;
END_VAR
VAR
    pTest : POINTER TO ARRAY [0..100] OF POINTER TO DWORD;
    diCount: DINT;
END_VAR

pTest := ADR(any1);
Generic_Compare := FALSE;
IF any1.typeclass <> any2.typeclass THEN
    RETURN;
END_IF
IF any1.diSize <> any2.diSize THEN
    RETURN;
END_IF
// Byte comparison
FOR iCount := 0 TO any1.diSize-1 DO
    IF any1.pvalue[iCount] <> any2.pvalue[iCount] THEN
        RETURN;
    END_IF
END_FOR
Generic_Compare := TRUE;
RETURN;
// END_FUNCTION
```

Declaration Syntax The syntax descriptions refer to a POU with exactly one parameter (an input variable).

```
FUNCTION | FUNCTION_BLOCK | METHOD <POU name> ( : <return data
type> )?
VAR_INPUT
    <input variable name> : <generic data type>;
END_VAR

<generic data type> = ANY | ANY_BIT | ANY_DATE | ANY_NUM | ANY_REAL |
ANY_INT | ANY_STRING
```

Example

```
FUNCTION funComputeAny : BOOL
VAR_INPUT
        anyInput1 : ANY; // valid data type see table
END_VAR
// END_FUNCTION

FUNCTION_BLOCK FB_ComputeAny
VAR_INPUT
    anyInput1 : ANY;
END_VAR
// END_FUNCTION_BLOCK

FUNCTION_BLOCK FB_ComputeMethod
METHOD methComputeAnny : BOOL
VAR_INPUT
    anyInput1 : ANY_INT; // valid data types are SINT, INT, DINT,
LINT, USINT, UINT, UDINT, ULINT
END_VAR
//END_METHOD
```

With compiler versions > 3.5.1.0, the generic IEC data types in the table are supported.

The table represents the hierarchy of the generic data types and provides information as to which generic data type of the formal parameter (declaration) allows which elementary data types of the argument (call).

Generic data type in the case of a formal parameter			**Permitted elementary data type in the case of an actual parameter (argument)**
ANY	ANY_BIT		• BYTE • WORD • DWORD • LWORD
	ANY_DATE		• DATE • DATE_AND_TIME, DT • TIME_OF_DAY, TOD • LDATE • LDATE_AND_TIME, LDT • LTIME_OF_DAY, LTOD
	ANY_NUM	ANY_REAL	REAL, LREAL
		ANY_INT	USINT, UINT, UDINT, ULINT SINT, INT, DINT, LINT
	ANY_STRING		STRING, WSTRING

Call

The syntax descriptions refer to a POU with exactly one parameter, to which an argument is passed. As a result, the data type of the argument specifies the generic data type of the input variable. For example, arguments of the type `BYTE, WORD, DWORD, LWORD` can be passed to a type `ANY_BIT` input variable.

Syntax of function call

```
<variable name> := <function name> ( <argument name> );
<argument name> : variable with valid data type
```

Syntax of function block call

```
<function block name> ( <input variable name> := <argument name> );
```

Syntax of method call

```
<function block name> . <method name> ( <input variable name> :=
<argument name> );
```

Example

```
PROGRAM PLC_PRG
VAR
    byValue : BYTE := 16#AB;
    iValue : INT := -1234;
    xResultByte : BOOL;
    xResultInt : BOOL;

    fbComputeAnyByte : FB_ComputeAny;
    fbComputeAnyInt : FB_ComputeAny;

    fbComputeM1 : FB_ComputeMethod;
    fbComputeM2 : FB_ComputeMethod;

    byN : BYTE := 1;
    wBitField1 : WORD := 16#FFFF;
    wBitField2 : WORD := 16#0001;
    xInit : BOOL;
    xResult : BOOL;
END_VAR

xResultByte := funComputeAny(byValue);
xResultInt := funComputeAny(iValue);

xResult := funGenericCompare(wBitField1, wBitField2);

fbComputeAnyByte(anyInput1 := byValue);
fbComputeAnyInt(anyInput1 := iValue);

fbComputeM1.methComputeAnny(anyInput1 := byValue);
fbComputeM2.methComputeAnny(anyInput1 := iValue);
// END_PRG
```

Data type 'WSTRING'

The data type WSTRING is interpreted in Unicode format as opposed to the data type STRING (ASCII). As a result of this coding, the number of displayed characters for WSTRING depends on the characters. A length of 10 for WSTRING means that the length of the WSTRING can take a maximum of 10 WORDs. However, for some characters in Unicode, multiple WORDS are required for coding a character so that the number of characters do not have to correspond to the length of the WSTRING (10 in this case). The data type requires 1 WORD of memory per character plus 1 WORD of extra memory. Each STRING requires only 1 byte. The data type WSTRING is terminated with a `0`.

Example:

```
wstr : WSTRING := "This is a WString";
```

See also

- *Chapter 1.4.1.19.5.4 “Data Type 'STRING'” on page 649*
- *Chapter 1.4.1.19.4.4 “String Constants” on page 634*

Data Type 'BIT'

The data type `BIT` is valid only in structures for the declaration of structure members or in a function block for the declaration of variables. A `BIT` variable can have the values `TRUE` (1) and `FALSE` (0). In this case, the variable requires exactly one bit of memory.

As a result, you can symbolically address individual bits by a name. `BIT` variables that are declared in succession are bundled in bytes. In this way, you can optimize memory use as opposed to `BOOL` types, which reserve 8 bits each. On the other hand, bit access is significantly more time-consuming. Therefore, you should use the `BIT` data type only when you need to define data in a predefined format.

See also

- *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*
- *Chapter 1.4.1.19.5.16 “Structure” on page 674*

Special Data Types '__UXINT', __XINT, and '__XWORD'

Variables with these data types are converted to a platform-compliant data type, depending on the target system.

CODESYS supports systems with address registers of 32-bit and 64-bit widths. For making the IEC code as independent from the target system as possible, you use the pseudo data types `__UXINT`, `__XINT`, and `__XWORD`. The compiler checks which target system types are current and then converts these data types into the appropriate standard data types.

Moreover, type conversion operators are provided for variables of these data types.

	Type conversion on 64-bit platforms	Type conversion on 32-bit platforms
`__UXINT`	`ULINT`	`UDINT`
`__XINT`	`LINT`	`DINT`
`__XWORD`	`LWORD`	`DWORD`

See also

- *Chapter 1.4.1.19.3.37 “Integer Conversion” on page 572*
- *Chapter 1.4.1.19.3.35 “Overloading” on page 565*

Pointers

A pointer stores the memory address of objects, such as variables or function block instances, at runtime.

Syntax of the pointer declaration:

```
<pointer name>: POINTER TO <data type | data unit type | function
block>;
```

Example

```
FUNCTION_BLOCK FB_Point
VAR
    piNumber: POINTER TO INT;
    iNumber1: INT := 5;
    iNumber2: INT;
END_VAR

piNumber := ADR(iNumber1); // piNumber is assigned to address of
iNumber1
iNumber2 := piNumber^; // value 5 of iNumber1 is assigned to
variable iNumber2 by dereferencing of pointer piNumber
```

Dereferencing a pointer means obtaining the value to which the pointer points. A pointer is dereferenced by appending the content operator `^` to the pointer identifier (for example, `piNumber^` in the example above). To assign the address of an object to a pointer, the address operator `ADR` is applied to the object: `ADR(iNumber1)`.

In online mode, you can click *"Edit ➔ Browse ➔ Go to Reference"* to jump from a pointer to the declaration location of the referenced variable.

NOTICE!

When a pointer points to an I/O input, write access applies. This leads to the compiler warning *"'<pointer name >' is not a valid assignment target"* when the code is generated. Example: `pwInput := ADR(wInput);`

If you require a construct of this kind, you have to first copy the input value (`wInput`) to a variable with write access.

Index access to pointers

CODESYS permits the index access `[]` to variables of type `POINTER TO`, as well as to the data types `STRING` or `WSTRING`.

The data, which the pointer points to, can also be accessed by appending the bracket operator `[]` to the pointer identifier(for example, `piData[i]`). The base data type of the pointer determines the data type and the size of the indexed component. In this case, the index access to the pointer is done arithmetically by adding the index dependent offset `i * SIZEOF( <base type> )` to the address of the pointer. The pointer is dereferenced implicitly at the same time.

Calculation: `piData[i] := (piData + i * SIZEOF(INT))^;`

This is **not**: `piData[i] != (piData + i)^`.

Index access `STRING`

When you use the index access with a variable of the type `STRING`, you get the character at the offset of the index expression. The result is of type `BYTE`. For example, `sData[i]` returns the i-th character of the character string `sData` as `SINT` (ASCII).

Index access `WSTRING`

When you use the index access with a variable of the type `WSTRING`, you get the character at the offset of the index expression. The result is of type `WORD`. For example, `wsData[i]` returns the i-th character of the character string as `INT` (Unicode).

Subtracting pointers

The result of the difference between two pointers is a value of type `DWORD`, even on 64-bit platforms when the pointers are 64-bit pointers.

Using references provides the advantage of guaranteeing type safety. That is not the case with pointers.

The memory access of pointers can be checked at runtime by the implicit monitoring function `CheckPointer`.

See also

- *Chapter 1.4.1.20.3.2.38 “Command 'Go To Reference'” on page 979*
- *Chapter 1.4.1.20.4.10.4 “Dialog 'Properties' - 'Build'” on page 1159*
- *Chapter 1.4.1.19.3.32 “Operator 'Content Operator'” on page 564*
- *Chapter 1.4.1.19.3.31 “Operator 'ADR'” on page 563*
- *Chapter 1.4.1.20.2.19.10 “POU 'CheckPointer'” on page 917*

Reference

A reference implicitly refers to another object. When accessed, the reference is implicitly dereferenced, and therefore does not need a special content operator `^` such as a pointer.

Syntax

```
<identifier> : REFERENCE TO <data type> ;
<data type>: base type of the reference
```

Example

```
PROGRAM PLC_PRG
VAR
    rspeA : REFERENCE TO DUT_SPECIAL;
    pspeA : POINTER TO DUT_SPECIAL;
    speB : DUT_SPECIAL;
END_VAR

rspeA REF= speB; // Reference rspeA is alias for speB. The code
corresponds to pspeA := ADR(speB);
rspeA := speD; // The code corresponds to pspeA^ := speB;
```

The readability of a program is made difficult when the same memory cell is accessed simultaneously by means of an identifier and its alias (for example, `speB` and `rspeA`).

NOTICE!
With compiler version >= V3.3.0.0, references are initialized (at 0).

> **NOTICE!**
> If a reference refers to a device input, then the access (for example, `rInput REF= Input;`) is applies as write access. This leads to a compiler warning when the code is generated: "`...invalid assignment target`".
>
> If you require a construct of this kind, you have to first copy the input value (`rInput`) to a variable with write access.

Invalid declarations

```
ariTest : ARRAY[0..9] OF REFERENCE TO INT;
priTest : POINTER TO REFERENCE TO INT;
rriTest : REFERENCE TO REFERENCE TO INT;
rbitTest : REFERENCE TO BIT;
```

A reference type must not be used as the base type of an array, pointer, or reference. Furthermore, a reference must not refer to a bit variable. These kinds of constructs generate compiler errors.

Comparison of reference and pointer

A reference has the following advantages over a pointer:

- Easier to use:
  A reference can access the contents of the referenced object directly and without dereferencing.
- Finer and simpler syntax when passing values:
  Call of a function block which passes a reference without an address operator instead of a pointer
  Example: `fbDoIt(riInput:=iValue);`
  Instead of: `fbDoIt_1(piInput:=ADR(iValue));`
- Type safety:
  When assigning two references, the compiler checks whether their base types match. This is not checked in the case of pointers.

Testing the validity of a reference

You can use the operator `__ISVALIDREF` to check whether or not a reference points to a valid value (meaning a value not equal to 0).

Syntax

```
<Boolean variable name> := __ISVALIDREF( <reference name> );
```

`<reference name>`: Identifier declared with `REFERENCE TO`

The Boolean variable is `TRUE` when the reference points to a valid value. Otherwise it is `FALSE`.

Example

```
PROGRAM PLC_PRG
VAR
iAlfa : INT;
riBravo : REFERENCE TO INT;
riCharlie : REFERENCE TO INT;
bIsRef_Bravo : BOOL := FALSE;
bIsRef_Charlie : BOOL := FALSE;
END_VAR

iAlfa := iAlfa + 1;
riBravo REF= iAlfa;
riCharlie REF= 0;
bIsRef_Bravo := __ISVALIDREF(riBravo);  (* becomes TRUE,
because riBravo references to iAlfa, which is non-zero
*)
bIsRef_Charlie := __ISVALIDREF(riCharlie); (* becomes FALSE,
because riCharlie is set to 0 *)
```

In compiler version 3.5.7.40 and later, the implicit monitoring function "CheckPointer" acts on variables of type `REFERENCE TO` in the same way as for pointer variables.

See also

- *Chapter 1.4.1.19.1.3.4.6 "Assignment Operator 'REF='" on page 468*
- *Chapter 1.4.1.20.2.19.10 "POU 'CheckPointer'" on page 917*

Data Type 'ARRAY'

An array is a collection of data elements of the same data type. CODESYS supports one- and multi-dimensional arrays of fixed or variable length.

Array of fixed length

You can define arrays in the declaration part of a POU or in global variable lists.

Syntax of the declaration of a one-dimensional array

```
<variable name> :  ARRAY[ <dimension> ] OF <data type> ( :=
<initialization> )? ;

<dimension> :  <lower index bound>..<upper index bound>
<data type> : elementary data types | user defined data types |
function block types
// (...)? : Optional
```

Syntax of the declaration of a multi-dimensional array

```
<variable name> :  ARRAY[ <1st dimension> ( , <next dimension> )+ ]
OF <data type> ( := <initialization> )? ;

<1st dimension> :  <1st lower index bound>..<1st upper index bound>
<next dimension> : <next lower index bound>..<next upper index bound>
<data type> : elementary data types | user defined data types |
function block types
// (...)+ : One or more further dimensions
// (...)? : Optional
```

The index limits are integers; maximum of the data type `DINT`.

Syntax for data access

```
<variable name>[ <index of 1st dimension> ( , <index of next
dimension> )* ]
// (...)* : 0, one or more further dimensions
```

Note the capability of using the implicit monitoring function `CheckBounds()` to monitor the maintenance of the index limits at runtime.

Example

One-dimensional array of 10 integer elements

```
VAR
    aiCounter : ARRAY[0..9] OF INT;
END_VAR
```

Lower index limit: 0

Upper index limit: 9

Initialization

```
aiCounter : ARRAY[0..9] OF INT := [0, 10, 20, 30, 40, 50, 60, 70,
80, 90];
```

Data access

```
iLocalVariable := aiCounter[2];
```

The value 20 is assigned to the local variable.

Example

2-dimensional array

```
VAR
    aiCardGame : ARRAY[1..2, 3..4] OF INT;
END_VAR
```

1st dimension: 1 to 2

2nd dimension: 3 to 4

Initialization

```
aiCardGame : ARRAY[1..2, 3..4] OF INT := [2(10),2(20)]; // Short
notation for [10, 10, 20, 20]
```

Data access

```
iLocal_1 := aiCardGame[1, 3]; // Assignment of 10
iLocal_2 := aiCardGame[2, 4]; // Assignment of 20
```

Example

3-dimensional array

```
VAR
    aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT;
END_VAR
```

1st dimension: 1 to 2

2nd dimension: 3 to 4

3rd dimension: 5 to 6

2 * 2 * 2 = 8 array elements

Initialization

```
aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [10, 20, 30, 40, 50,
60, 70, 80];
```

Data access

```
iLocal_1 := aiCardGame[1, 3, 5]; // Assignment of 10
iLocal_2 := aiCardGame[2, 3, 5]; // Assignment of 20
iLocal_3 := aiCardGame[1, 4, 5]; // Assignment of 30
iLocal_4 := aiCardGame[2, 4, 5]; // Assignment of 40
iLocal_5 := aiCardGame[1, 3, 6]; // Assignment of 50
iLocal_6 := aiCardGame[2, 3, 6]; // Assignment of 60
iLocal_7 := aiCardGame[1, 4, 6]; // Assignment of 70
iLocal_8 := aiCardGame[2, 4, 6]; // Assignment of 80
```

Initialization

```
aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [2(10), 2(20),
2(30), 2(40)]; // Short notation for [10, 10, 20, 20, 30, 30, 40,
40]
```

Data access

```
iLocal_1 := aiCardGame[1, 3, 5]; // Assignment of 10
iLocal_2 := aiCardGame[2, 3, 5]; // Assignment of 10
iLocal_3 := aiCardGame[1, 4, 5]; // Assignment of 20
iLocal_4 := aiCardGame[2, 4, 5]; // Assignment of 20
iLocal_5 := aiCardGame[1, 3, 6]; // Assignment of 30
iLocal_6 := aiCardGame[2, 3, 6]; // Assignment of 30
iLocal_7 := aiCardGame[1, 4, 6]; // Assignment of 40
iLocal_8 := aiCardGame[2, 4, 6]; // Assignment of 40
```

Example

3-dimensional arrays of a user-defined structure

```
TYPE DATA_A
STRUCT
 iA_1 : INT;
 iA_2 : INT;
 dwA_3 : DWORD;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    aData_A : ARRAY[1..3, 1..3, 1..10] OF DATA_A;
END_VAR
```

The array `aData_A` consists of a total of 3 * 3 * 10 = 90 array elements of data type `DATA_A`.

Initialize partially

```
aData_A : ARRAY[1..3, 1..3, 1..10] OF DATA_A := [(iA_1 := 1,
iA_2 := 10, dwA_3 := 16#00FF),(iA_1 := 2, iA_2 := 20, dwA_3 :=
16#FF00),(iA_1 := 3, iA_2 := 30, dwA_3 := 16#FFFF)];
```

In the example, only the first 3 elements are initialized explicitly. Elements to which no initialization value is assigned explicitly are initialized internally with the default value of the basic data type. This initializes the structure components at 0 starting with the element `aData_A[2, 1, 1]`.

Data access

```
iLocal_1 := aData_A[1,1,1].iA_1; // Assignment of 1
dwLocal_2 := aData_A[3,1,1].dwA_3; // Assignment of 16#FFFF
```

Example

Array of a function block

```
FUNCTION BLOCK FBObject_A
VAR
    iCounter : INT;
END_VAR
...

PROGRAM PLC_PRG
VAR
    aObject_A : ARRAY[1..4] OF FBObject_A;
END_VAR
```

The array `aObject_A` consists of 4 elements. Each element instantiates a `FBObject_A` function block.

Function call

```
aObject_A[2]();
```

Example

Implementation of `FB_Something` with method `FB_Init`

```
FUNCTION_BLOCK FB_Something
VAR
    _nId : INT;
    _lrIn : LREAL;
END_VAR
...

METHOD FB_Init : BOOL
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode : BOOL;
    nId : INT;
    lrIn : LREAL;
END_VAR

_nId := nId;
_lrIn := lrIN;
```

The function block `FB_Something` has a method `FB_Init` that requires 2 parameters.

Instantiation of the array with initialization

```
PROGRAM PLC_PRG
VAR
    fb_Something_1 : FB_Something(nId := 11, lrIn := 33.44);
    a_Something : ARRAY[0..1, 0..1] OF FB_Something[(nId := 12,
lrIn := 11.22), (nId := 13, lrIn := 22.33), (nId := 14, lrIn :=
33.55),(nId := 15, lrIn := 11.22)];
END_VAR
```

Array of arrays

The declaration of an "array of arrays" is an alternative syntax for multidimensional arrays. A collection of elements is nested instead of dimensioning the elements. The nesting depth is unlimited.

Syntax for declaration

```
<variable name> :  ARRAY[<first>] ( OF ARRAY[<next>] )+ OF <data
type> ( := <initialization> )? ;

<first> :  <first lower index bound>..<first upper index bound>
<next> : <lower index bound>..<upper index bound> // one or more
arrays
<data type> : elementary data types | user defined data types |
function block types
// (...)+ : One or more further arrays
// (...)? : Optional
```

Syntax for data access

```
<variable name>[<index of first array>] ( [<index of next array>] )+ ;
// (...)* : 0, one or more further arrays
```

Example

```
PROGRAM PLC_PRG
VAR
    aiPoints : ARRAY[1..2,1..3] OF INT := [1,2,3,4,5,6];
    ai2Boxes : ARRAY[1..2] OF ARRAY[1..3] OF INT := [ [1, 2, 3],
[ 4, 5, 6]];
    ai3Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF INT :=
[ [ [1, 2, 3, 4], [5, 6, 7, 8 ], [9, 10, 11, 12] ], [ [13, 14, 15,
16], [ 17, 18, 19, 20], [21, 22, 23, 24] ] ];
    ai4Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF
ARRAY[1..5] OF INT;
END_VAR

aiPoints[1, 2] := 1200;
ai2Boxes[1][2] := 1200;
```

The variables `aiPoints` and `ai2Boxes` collect the same data elements, however the syntax for the declaration differs from that of the data access.

aiPoints	ARRAY [1..2, 1..3] OF INT	
aiPoints[1, 1]	INT	1
aiPoints[1, 2]	INT	1200
aiPoints[1, 3]	INT	3
aiPoints[2, 1]	INT	4
aiPoints[2, 2]	INT	5
aiPoints[2, 3]	INT	6

ai2Boxes	ARRAY [1..2] OF ARRAY [1..3] OF INT	
ai2Boxes[1]	ARRAY [1..3] OF INT	
ai2Boxes[1][1]	INT	1
ai2Boxes[1][2]	INT	1200
ai2Boxes[1][3]	INT	3
ai2Boxes[2]	ARRAY [1..3] OF INT	
ai2Boxes[2][1]	INT	4
ai2Boxes[2][2]	INT	5
ai2Boxes[2][3]	INT	6

Array of variable length

In function blocks, functions, or methods, you can declare arrays of variable length in the `VAR_IN_OUT` declaration section.

The `LOWER_BOUND` and `UPPER_BOUND` operators are provided for determining the index limits of the actual used array at runtime.

Only statically declared arrays (not arrays generated by means of the operator `__NEW`) may be passed to an array with variable length.

Syntax of the declaration of a one-dimensional array of variable length

```
<variable name> :  ARRAY[*] OF <data type> ( := <initialization> )? ;

<data type> : elementary data types | user defined data types |
function block types
// (...)? : Optional
```

Syntax of the declaration of a multi-dimensional array of variable length

```
<variable name> :  ARRAY[* ( , * )+ ] OF <data type> ( :=
<initialization> )? ;

<data type> : elementary data types | user defined data types |
function block types
// (...)+ : One or more further dimensions
// (...)? : Optional
```

Syntax of the operators for calculating the limit index

```
LOWER_BOUND( <variable name> , <dimension number> )
UPPER_BOUND( <variable name> , <dimension number> )
```

Example

The `SUM` function adds the integer values of the array elements and returns the calculated sum as a result. The sum is calculated across all array elements available at runtime. As the actual number of array elements will only be known at runtime, the local variable is declared as a one-dimensional array of variable length.

```
FUNCTION SUM: INT;
VAR_IN_OUT
    aiData : ARRAY[*] OF INT;
END_VAR
VAR
 diCounter, diResult : DINT;
END_VAR

diResult := 0;
FOR diCounter := LOWER_BOUND(aiData, 1) TO UPPER_BOUND(aiData, 1)
DO // Calculates the length of the current array
 diResult := diResult + A[i];
END_FOR;
SUM := diResult;
```

See also

- *Chapter 1.4.1.8.2.3 “Declaring arrays” on page 228*
- *Chapter 1.4.1.20.2.19.1 “POU 'CheckBounds'” on page 906*

Data Type '__VECTOR'

Vector operations are supported natively only on 64-bit processors and offer a performance advantage only on these processors. The data sheet of the controller provides information about the processor used on the controller.

Currently, vector operations on the x86/64-bit platforms with SSE2 and ARM64 with NEON are supported natively. On all other platforms, vector operations are translated into individual statements. For example, vector addition is then executed with multiple single addition operations.

The command set extensions of the processors are SIMD extensions. SIMD (Single Instruction, Multiple Data) describes a computer architecture in which multiple data sets of the same type are processed simultaneously in parallel and therefore faster with one command call. In vector operations, for example, 4 pairs of numbers can then be added at the same time.

Syntax

```
<variable name> :  __VECTOR[ <vector size> ] OF <element type> ( :=
<initialization> )? ;

<vector size> : 1 |2 | 3 | 4 | 5| 6 | 7| 8
<element type> : REAL | LREAL
// (...)? : Optional
```

A vector data type is an array of floating-point numbers with a maximum of 8 elements. The operators `__vc<operator name>` are available for this data type. You can use these to implement vector operations without additional function calls.

Syntax for index access

```
<variable name>[ <index> ]
<index> : 0 | 1 | 2| 3 | 4 | 5| 6 | 7
```

When indexing a vector variable, you can access a single element of the vector. The index starts at 0 and goes until <vector size> - 1.

Example

```
PROGRAM PLC_PRG
VAR
    vcA : __VECTOR[3] OF REAL;
END_VAR

vcA[0] := 1.1;
vcA[1] := 2.2;
vcA[2] := 3.3;
```

Determining the optimal vector size

Use the optimal vector size depending on your target system as the vector size in order to program the most efficient code possible.

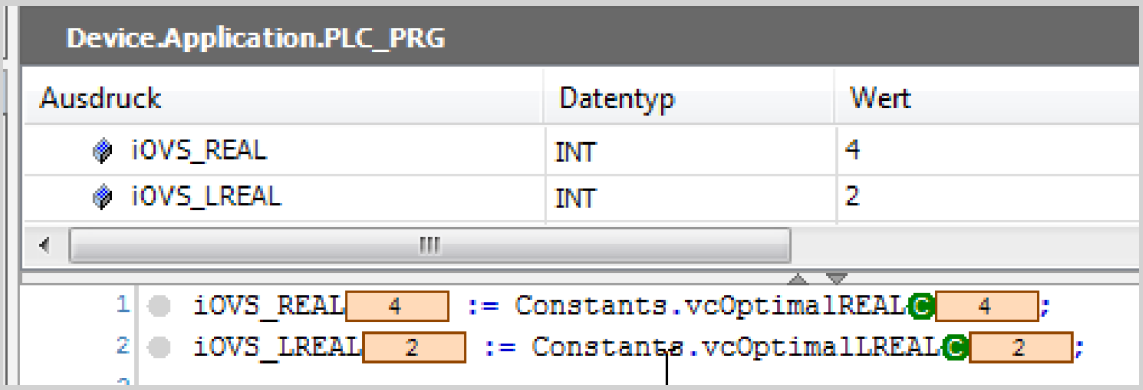
For target systems whose computer architecture is generally suitable for vector processing, we do not recommend using vectors of arbitrary size. There is an optimal vector size depending on the type of data processing of the processor. Vectors that are declared with this array size are processed as quickly as possible. Vectors that are declared as a larger array do not have an advantage in speed. Vectors that are declared as smaller arrays do not take full advantage of the processor's capabilities.

You can query the optimal size at runtime. You can find the information in the constants `Constants.vcOptimalREAL` (for vectors with `REAL` elements) and `Constants.vcOptimalLREAL` (for vectors with `LREAL` elements). The constants have the `LREAL` data type. If a constant returns the value `1` as the optimal value, then this means that accelerated vector processing is not available for the target system.

Example

```
PROGRAM PLC_PRG
VAR
    iOVS_REAL : INT; // Optimal vector size for REAL elements
    iOVS_LREAL : INT; // Optimal vector size for LREAL elements
END_VAR

iOVS_REAL := Constants.vcOptimalREAL;
iOVS_LREAL := Constants.vcOptimalLREAL;
```

An application that is loaded on the CODESYS Control Win V3 x64 target system returns the following values at runtime:



Operator __VCADD

The operator calculates the sum of two vectors.

Syntax

```
<vector variable> := <1st vector operand> __VCADD <2nd vector
operand>;
```

Example of addition

```
FUNCTION_BLOCK FB_ADD
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
    vcResult : __VECTOR[3] OF REAL;
END_VAR


vcResult := vcA __VCADD vcB;
```

Operator __VCSUB

The operator calculates the difference between two vectors.

Syntax

```
<vector variable> := <vector minuend> __VCSUB <vector subtrahend>;
```

Example of subtraction

```
FUNCTION_BLOCK FB_SUB
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
    vcResult0 : __VECTOR[3] OF REAL;
    vcResult1 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := vcA __VCSUB vcB;
vcResult1 := vcB __VCSUB vcA;
```

Operator __VCMUL

The operator calculates the product of two vectors or a scalar (floating-point number) and a vector.

Syntax

```
<vector variable> := <1st vector operand> __VCMUL <2nd vector
operand> | <scalar operand> __VCMUL <vector operand> | <vector
operand> __VCMUL <scalar operand> ;
```

Example of multiplication

```
FUNCTION_BLOCK FB_MUL
VAR
    rScalar : REAL := 1.1;
    vcA : __VECTOR[3] OF REAL;
    vcB : __VECTOR[3] OF REAL;
    vcResult0 : __VECTOR[3] OF REAL;
    vcResult1 : __VECTOR[3] OF REAL;
    vcResult2 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := vcA __VCMUL vcB;
vcResult1 := rScalar __VCMUL vcB;
vcResult2 := vcA __VCMUL 3.3;
```

Operator __VCDIV

The operator calculates the quotient of two vectors or a vector and a scalar.

Syntax

```
<vector variable> := <vector dividend> __VCDIV <vector divisor> | <
vector dividend> __VCMUL <scalar divisor> ;
```

Example of division

```
FUNCTION_BLOCK FB_DIV
VAR
    iScalar : INT := 3;
    rScalar : REAL := 1.5;
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
    vcResult0 : __VECTOR[3] OF REAL;
    vcResult1 : __VECTOR[3] OF REAL;
    vcResult2 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := vcA __VCDIV vcB;
// ERROR CODE vcResult1 := rScalar __VCDIV vcB;
// ERROR CODE vcResult1 := iScalar __VCDIV vcB;
// ERROR CODE vcResult1 := 3.3 __VCDIV vcB;
vcResult2 := vcA __VCDIV 1.5;
vcResult2 := vcA __VCDIV iScalar;
vcResult2 := vcA __VCDIV rScalar;
```

Operator __VCDOT

The operator calculates the dot product (scalar product) of two vectors.

Syntax

```
<scalar variable> := <1st vector operand> __VCDOT <2nd vector
operand> ;
```

Example of a dot product

```
FUNCTION_BLOCK FB_DOT
VAR
    rResult : REAL;
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
END_VAR

rResult := vcA __VCDOT vcB; // = 18
```

Operator __VCSQRT

The operator calculates the square root of each element in the vector.

Syntax

```
<vector variable> := __VCSQRT( <vector operand> );
```

Example of a square root

```
FUNCTION_BLOCK FB_SQRT
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(4, 9, 16);
    vcResult0 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := __VCSQRT(vcA);
```

Operator __VCMAX

The operator calculates the maximum vector of two vectors. The maximum is determined element by element.

Syntax

```
<vector variable> := __VCMAX( <1st vector operand>, <2nd vector
operand>);
```

Example of a maximum vector

```
FUNCTION_BLOCK FB_MAX
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 6);
    vcResult0 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := __VCMAX(vcA, vcB);
```

Operator __VCMIN

The operator calculates the minimum vector of two vectors. The minimum is determined element by element.

Syntax

```
<vector variable> := __VCMIN( <1st vector operand>, <2nd vector
operand>);
```

Example of a minimum vector

```
FUNCTION_BLOCK FB_MIN
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 6);
    vcResult0 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := __VCMIN(vcA, vcB);
```

Operator __VCSET_REAL

The operator sets all elements of a vector in a statement. The elements have the `REAL` data type.

Syntax

```
<vector variable> := __VCSET_REAL( <first literal>, ( < next
literal> )+ ) ;
( ... )+ // number of elements have to match
```

Example

```
FUNCTION_BLOCK FB_SET
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
END_VAR

vcA := __VCSET_REAL(4, 4, 4);
vcB := __VCSET_REAL(1.1, 2.2, 3.3);
```

Operator __VCSET_LREAL

The operator sets all elements of a vector at once in a statement. The elements have the `LREAL` data type.

They can be used wherever variables are valid, such as in assignments in implementations or as parameters in function calls.

Syntax

```
<vector variable> := __VCSET_LREAL( <first literal>, ( < next
literal> )+ ) ;
( ... )+ // number of elements have to match
```

Example

```
FUNCTION_BLOCK FB_SET
VAR
    vclA : __VECTOR[3] OF LREAL := __VCSET_LREAL(3, 3, 3);
    vclB : __VECTOR[3] OF LREAL := __VCSET_LREAL(1, 2, 3);
END_VAR

vclA := __VCSET_LREAL(-1.7976931348623158E+308, 0.0,
1.7976931348623158E+308);
vclB := __VCSET_LREAL(-1.7976931348623158E+308, 0.0,
1.7976931348623158E+308);
```

Operator __VCLOAD_REAL

The operator interprets any arbitrary memory area as a vector. This is helpful for connecting vector variables to existing code. The operator requires 2 parameters. The first parameter indicates the number of vector elements. The second parameter is a pointer to the `REAL` data.

Syntax

```
<vector variable> := __VCLOAD_REAL( <vector size>, <pointer to data
of type REAL> ) ;
<vector size> : 2 | 3 | 4 | 5| 6 | 7| 8
```

Example of vectorization

```
FUNCTION_BLOCK FB_LOAD
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    rData0 : REAL := 1.234;
    rData1: REAL := 5.678;
    rData2 : REAL := 9.123;
    pData: POINTER TO REAL := ADR(rData0);

    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
END_VAR

vcA := __VCLOAD_REAL(3, pData);
```

Operator __VCLOAD_LREAL

The operator interprets any arbitrary memory area as a vector. This is helpful for connecting vector variables to existing code. The operator requires 2 parameters. The first parameter indicates the number of vector elements. The second parameter is a pointer to the `LREAL` data.

Syntax

```
<vector variable> := __VCLOAD_REAL( <vector size>, <pointer to data
of type LREAL> );
<number of vector elements> : 1 | 2 | 3 | 4 | 5| 6 | 7| 8
```

Example of vectorization

```
FUNCTION_BLOCK FB_LOAD
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    lrData0 : LREAL := -1.7976931348623158E+308;
    lrData1: LREAL := 1.6E+308;
    lrData2 : LREAL := 1.7E+308;
    lrData3 : LREAL := -1.6E+308;
    plData: POINTER TO LREAL := ADR(lrData0);

    vclA : __VECTOR[4] OF LREAL := __VCSET_LREAL(4, 4, 4, 4);
END_VAR
vclA := __VCLOAD_LREAL(4, plData);
```

Operator __VCSTORE

The operator saves/copies the contents of the vector to the specified memory address. The number and the types of elements are automatically applied from the vector variables.

Syntax

```
__VCSTORE( <pointer to data>, <vector variable> );
```

Example of storage

```
FUNCTION_BLOCK FB_STORE
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    rData0 : REAL := 3;
    rData1: REAL := 3;
    rData2 : REAL := 3;
    pData: POINTER TO REAL := ADR(rData0);

    lrData0 : LREAL := 4;
    lrData1: LREAL := 4;
    lrData2 : LREAL := 4;
    lrData3 : LREAL := 4;
    plData: POINTER TO LREAL := ADR(lrData0);


    vcA : __VECTOR[3] OF REAL := __VCSET_REAL( 1.234, 5.678, 9.123);
    vclA : __VECTOR[4] OF LREAL :=
__VCSET_LREAL(-1.7976931348623158E+308, 1.6E+308, 1.7E+308,
-1.6E+308);
END_VAR

__VCSTORE(pData, vcA);
__VCSTORE(plData, vclA);
```

See also

- *Chapter 1.4.1.8.2.3 “Declaring arrays” on page 228*
- *Chapter 1.4.1.20.2.19.1 “POU 'CheckBounds'” on page 906*

Structure

A structure is a user-defined data type, which combines multiple variables of any data type into a logical unit. The variables declared within a structure are called members.

You make the type declaration of a structure in a *"DUT"* object which you create in the *"Project ➔ Add Object ➔ DUT"* menu or in the context menu of an application.

Syntax

```
TYPE <structure name> :
STRUCT
    ( <variable declaration optional with initialization> )+
END_STRUCT
END_TYPE
```

`<structure name>` is an identifier which is valid in the entire project so that you can use it like a standard data type. Moreover, you can declare any number of variables (at least one) which are supplemented optionally by an initialization.

Structures can also be nested. This means that you declare a structure member with an existing structure type. Then the only restriction is that you must not assign any address to the variable (structure member). (The AT declaration is not permitted here.)

Example

Type declaration

```
TYPE S_POLYGONLINE :
STRUCT
    aiStart : ARRAY[1..2] OF INT := [-99, -99];
    aiPoint1 : ARRAY[1..2] OF INT;
    aiPoint2 : ARRAY[1..2] OF INT;
    aiPoint3 : ARRAY[1..2] OF INT;
    aiPoint4 : ARRAY[1..2] OF INT;
    aiEnd : ARRAY[1..2] OF INT := [99, 99];
END_STRUCT
END_TYPE
```

Extension of a type declaration

An additional structure is declared from an existing structure. In addition to its own members, the extended structure also has the same structure members as the base structure.

Syntax

```
TYPE <structure name> EXTENDS <basis structure> :
STRUCT
    ( <variable declaration optional with initialization> )+
END_STRUCT
END_TYPE
```

Example

Type declaration

```
TYPE S_PENTAGON EXTENDS S_POLYGONLINE :
STRUCT
    aiPoint5 : ARRAY[1..2] OF INT;
END_STRUCT
END_TYPE
```

Declaration and initialization of structure variables

Example

```
PROGRAM progLine
VAR
    sPolygon : S_POLYGONLINE := (aiStart:=[1,1], aiPoint1:=[5,2],
aiPoint2:=[7,3], aiPoint3:=[8,5], aiPoint4:=[5,7], aiEnd:=[1,1]);
    sPentagon : S_PENTAGON := (aiStart:=[0,0], aiPoint1:=[1,1],
aiPoint2:=[2,2], aiPoint3:=[3,3], aiPoint4:=[4,4], aiPoint5:=[5,5],
aiEnd:=[0,0]);
END_VAR
```

You must not permitted to use initializations with variables. For an example of initializing an array of a structure, see the help page for the data type `ARRAY`.

Access to a structure member

You access structure members with the following syntax:

```
<variable name> . <component name>
```

Example

```
PROGRAM prog_Polygon
VAR
    sPolygon : S_POLYGONLINE := (aiStart:=[1,1], aiPoint1:=[5,2],
aiPoint2:=[7,3], aiPoint3:=[8,5], aiPoint4:=[5,7], aiEnd:=[1,1]);
    iPoint: INT;
END_VAR

// Assigns 5 to aiPoint
iPoint := sPolygon.aiPoint1[1];
```

Result: `iPoint = 5`

Symbolic bit access in structure variables

You can declare a structure with variables of data type `BIT` to combine individual bits into a logical unit. Then you can symbolically address individual bits by a name (instead of by a bit index).

Syntax declaration

```
TYPE <structure name> :
STRUCT
    ( <bit name> : BIT; )+
END_STRUCT
END_TYPE
```

Syntax of bit access

```
<structure name> . <bit name>
```

Example

Type declaration

```
TYPE S_CONTROL :
STRUCT
    bitOperationEnabled : BIT;
    bitSwitchOnActive : BIT;
    bitEnableOperation : BIT;
    bitError : BIT;
    bitVoltageEnabled : BIT;
    bitQuickStop : BIT;
    bitSwitchOnLocked : BIT;
    bitWarning : BIT;
END_STRUCT
END_TYPE
```

Bit access

```
FUNCTION_BLOCK FB_Controller
VAR_INPUT
        xStart : BOOL;
END_VAR
VAR_OUTPUT
END_VAR
VAR
    ControlDriveA : S_CONTROL;
END_VAR

IF xStart = TRUE THEN
    // Symbolic bit access
    ControlDriveA.bitEnableOperation := TRUE;
END_IF

PROGRAM PLC_PRG
VAR
    fbController : FB_Controller;
END_VAR

fbController();
fbController.xStart := TRUE;
```

*References and pointers to `BIT` variables are **invalid** declarations, as well as array elements with base type `BIT`.*

See also

- *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*

See also

- *Chapter 1.4.1.19.5.14 “Data Type 'ARRAY'” on page 660*
- *Chapter 1.4.1.19.5.10 “Data Type 'BIT'” on page 656*
- *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835*

Enumerations

An enumeration is a user-defined data type composed of a series of comma-separated components (enumeration values) for declaring user-defined variables. Moreover, you can use the enumeration components like constants whose identifier `<enumeration name>.<component name>` is recognized globally in the project.

You declare an enumeration in a DUT object, which you have already created in the project by clicking *“Add Object”*.

Declaration

Syntax

```
( {attribute 'strict'} )? // Pragma optional but recommended
TYPE <enumeration name> :
(
    <first component declaration>,
    ( <component declaration> ,)+
    <last component declaration>
)( <basic data type> )? ( := <default variable initialization> )? ;
END_TYPE

( ... )? : Optional
<component declaration> : <component name> ( := <component
initialization> )?
<basic data type> : INT | UINT | SINT | USINT | DINT | UDINT | LINT |
ULINT | BYTE | WORD | DWORD | LWORD
<variable initialization> : <one of the component names>
```

In an enumeration declaration, at least 2 components are usually declared. However, you can declare as many as you want. Every single component can be assigned its own initialization. Enumerations automatically have the basic data type `INT`, but you can specify another basic data type. Moreover, you can specify a component in the declaration with which an enumeration variable is then initialized.

The pragma `{attribute 'strict'}` causes a strict type test to be performed as described below.

Example

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE COLOR_BASIC :
(
    yellow,
    green,
    blue,
    black
)
; // Basic data type is INT, default initialization for all
COLOR_BASIC variables is yellow
END_TYPE
```

Enumeration with explicit basic data type

Extensions to the IEC 61131-3 standard

The basic data type for an enumeration declaration is `INT` by default. However, you can also declare enumerations that are based explicitly on another integer data type.

```
<basic data type> : INT | UINT | SINT | USINT | DINT | UDINT | LINT |
ULINT | BYTE | WORD | DWORD | LWORD
```

Example

Enumeration with basic data type DWORD

```
TYPE COLOR :
(
    white := 16#FFFFFF00,
    yellow := 16#FFFFFF00,
    green := 16#FF00FF00,
    blue := 16#FF0000FF,
    black := 16#88000000
) DWORD := black; // Basic data type is DWORD, default
initialization for all COLOR variables is black
END_TYPE
```

Strict programming rules

NOTICE!
In CODESYS V3.5 SP7 and later, the pragma `{attribute 'strict'}` is added automatically in the first line when declaring an enumeration.

The strict programming rules are activated when adding the pragma `{attribute 'strict'}`.

The following code is considered a compiler error:

- Arithmetic operations with enumeration components
  For example, an enumeration variable cannot be used as a counter variable in a `FOR` loop.
- Assignment of a constant value, which does not correspond to an enumeration value, to an enumeration component
- Assignment of a non-constant variable, which has another data type as the enumeration, to an enumeration component

Arithmetic operations can lead to undeclared values being assigned to enumeration components. A better programming style is to use `SWITCH`/`CASE` statements for processing component values.

Declaration and initialization of enumeration variables

Syntax

```
<variable name> : <enumeration name> ( := <initialization> )? ;
```

For a declaration of an enumeration variable with user-defined data type `<enumeration name>`, this can be initialized with an enumeration component.

Example

```
PROGRAM PLC_PRG
VAR
    colorCar: COLOR;
    colorTaxi : COLOR := COLOR.yellow;
END_VAR
```

The variable `colorCar` is initialized with `COLOR.black`. That is the default initialization for all enumeration variables of type `COLOR` and defined this way in the type declaration. The variable `colorTaxi` has its own initialization.

If no initializations are specified, then the initialization value is 0.

Example

```
PROGRAM PLC_PRG
VAR
    cbFlower : COLOR_BASIC;
    cbTree: COLOR_BASIC := COLOR_BASIC.green;
END_VAR
```

The variable `cbFlower` is initialized with `COLOR_BASIC.yellow`. That is the default initialization for all enumeration variables of type `COLOR_BASIC`. Because the enumeration declaration does not specify a component for initialization, the system automatically initializes with the component that has the value 0. This is usually the first of the enumeration components. However, it can also be another component that is not in the first position but explicitly initialized with 0.

The variable `cbTree` has an explicit initialization.

If no value is specified for both the type and the variable, then the following rule applies: If an enumeration contains a value for 0, then this value is the default initialization, and if not, then the first component in the list.

Example

Initialization with the 0 component

```
TYPE ENUM :
(
    e1 := 2,
    e2 := 0,
    e3
)
;
END_TYPE

PROGRAM PLC_PRG
VAR
    e : ENUM;
END_VAR
```

The variable `e` is initialized with `ENUM.e2`.

Initialization with the first component

```
TYPE ENUM2 :
(
    e1 := 3,
    e2 := 1,
    e3
)
;
END_TYPE

PROGRAM PLC_PRG
VAR
    e2 : ENUM2;
END_VAR
```

The variable `e2` is initialized with `ENUM.e1`.

Unique access to enumeration components

Extensions to the IEC 61131-3 standard

The enumeration components can also be used as constant variables with the identifier `<enumeration name>.<component name>`. Enumeration components are recognized globally in the project and access to them is unique. Therefore, a component name can be used in different enumerations.

Example

Component blue

```
PROGRAM PLC_PRG
VAR
    cbFlower : COLOR_BASIC;
    colorCar : COLOR;
END_VAR

(* unambiguous identifiers although the component names are
identical *)
cbFlower := COLOR_BASIC.blue;
colorCar := COLOR.blue;

(* invalid code *)
cbFlower := blue;
colorCar := blue;
```

See also

- *Chapter 1.4.1.19.3.72 “Operator - Enumeration namespace” on page 630*

Alias

An alias is a user-defined data type with which an alternative name for a basic type, data type, or function block is generated.

You make the type declaration of an alias in a *“DUT”* object which you create in the *“Project ➔ Add Object ➔ DUT”* menu or in the context menu of an application.

Syntax

```
TYPE <DUT name> : <basic type> | <data type> | <function block
name> ;
END_TYPE
```

Example

```
FUNCTION_BLOCK FB_Machine
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iCounter : INT;
END_VAR
iCounter := iCounter + 1;

// Alias for FB_Machine
TYPE A_ROBOT : FB_Machine;
END_TYPE

PROGRAM prog_Robot
VAR
    fbRobot : A_ROBOT;
END_VAR
fbRobot();
```

Data type 'UNION'

A UNION is a data structure that usually contains different data types.

In a union, all components have the same offset and therefore the same amount of memory. In the following declaration example of a union, an assignment to `name.a` will also affect `name.b`.

Example

```
TYPE name:
UNION
    a : LREAL;
    b : LINT;
END_UNION
END_TYPE
```

Subrange types

A subrange type is a data type whose value range is a subset of a base type.

Syntax for the declaration:

```
<name> : <int type> (<lower limit>..<upper limit>);
```

`<name>`	valid IEC identifier
`<int type>`	data type of the subrange (SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LINT, ULINT, LWORD).
`<lower limit>`	Lower limit of the range: constants that have to be compatible with the basic data type. The lower limit is also included in this range.
`<upper limit>`	Upper limit of the range: constants that have to be compatible with the base data type. The upper limit is also included in this range.

Examples:

```
VAR
 i : INT (-4095..4095);
 ui : UINT (0..10000);
END_VAR
```

If you assign a value to a subrange type in the declaration or implementation section that is not within this range (example: i:=5000), then CODESYS issues an error message.

Please note: In runtime mode, it is possible to monitor the range limits of a subrange type by using the implicit monitoring functions `CheckRangeSigned` and `CheckRangeUnsigned`.

See also

- *Chapter 1.4.1.20.2.19.7 “POU 'CheckLRangeSigned'” on page 914*
- *Chapter 1.4.1.20.2.19.9 “POU 'CheckLRangeUnsigned'” on page 916*

Redundancy State

The library `Redundancy` provides the structure `RedundancyState` with the components describes below.

- By calling `GetRedundancyState(ADR(<RedundancyState name>));`, the state of the redundancy system is read and stored. As a result, you get programmatic access to the redundancy state and can display it, for example in a visualization.
- The redundancy state is also automatically displayed in the *"Redundancy Configuration"* editor, on the *"Redundancy State"* tab. There in the system graphic in the lower right corner of the respective PLC, the redundancy state is displayed symbolically by circle symbols. Moreover, the state is output as text below in the *"Redundancy State"* field. In this way, you can monitor the state of the redundancy system.

The following table compares these two options.

Circle Symbol	Output text under *"Redundancy State"*	`STRUCT RedundancyState`	Description
		`RS_START`	Initial state Tries (when configured correctly) to synchronize with the other PLC If not correctly configured, then the state is set to `RS_SIMULATION_START`.
		`RS_SYNCHRO`	Boot application downloaded, data synchronized, and task started Fieldbus deactivated
A	*"Active"*	`RS_CYCLE_ACTIVE`	Works in redundant synchronized mode as active PLC Fieldbus activated
P	*"Passive"*	`RS_CYCLE_STANDBY`	Works in redundant synchronized mode as passive PLC Fieldbus deactivated
S	*"Standalone"*	`RS_CYCLE_STANDALONE`	Works in standalone mode, not synchronized with the other PLC Fieldbus activated
E	*"Error"*	`RS_CYCLE_ERROR`	Fieldbus error, occurred in redundant synchronized mode as active PLC Fieldbus deactivated
Sim	*"Simulation"*	`RS_SIMULATION`	If not configured Works in standalone mode, not synchronized with the other PLC Fieldbus deactivated
		`RS_BOOTUP_ERROR`	If in state `RS_CYCLE_ACTIVE` The other PLC will become active because the PROFIBUS displays a problem with the active PLC (us). Fieldbus deactivated

Circle Symbol	Output text under *“Redundancy State”*	`STRUCT Redundancy State`	Description
		`RS_SHUTDOWN_ACTIVE`	Ends the runtime systems in state `RS_CYCLE_STANDALONE` Note: Leave the fieldbus activated on download.
		`RS_SHUTDOWN_STANDBY`	Ends the runtime systems in state `RS_CYCLE_STANDBY` Note: Deactivate the fieldbus on download.
		`RS_SYNCHRO_ERROR`	Error occurred during the state `RS_SYNCHRO` Fieldbus deactivated
		`RS_SIMULATION_START`	With setting `Simulation=1` Works in standalone mode after the start, not synchronized with the other PLC Fieldbus deactivated Note: A synchronization can be triggered later with library functions.
		`RS_NO_LICENCE`	If no license is installed and the demo time has expired. No operation
	*“Unknown”*	No assignment	Indicates that the connection to the PLC is not online

See also

-
-

1.4.1.19.6 Pragmas

Pragma instructions affect the properties of one or more variables with regard to the compilation or pre-compilation process. Various categories of pragmas are available to you for this.

Message Pragmas

Message pragmas serve to force the display of messages in the Message window during the compilation process.

Insertion position: separate or already existing line in the text editor of a POU.

Table 44: 4 types of message pragmas

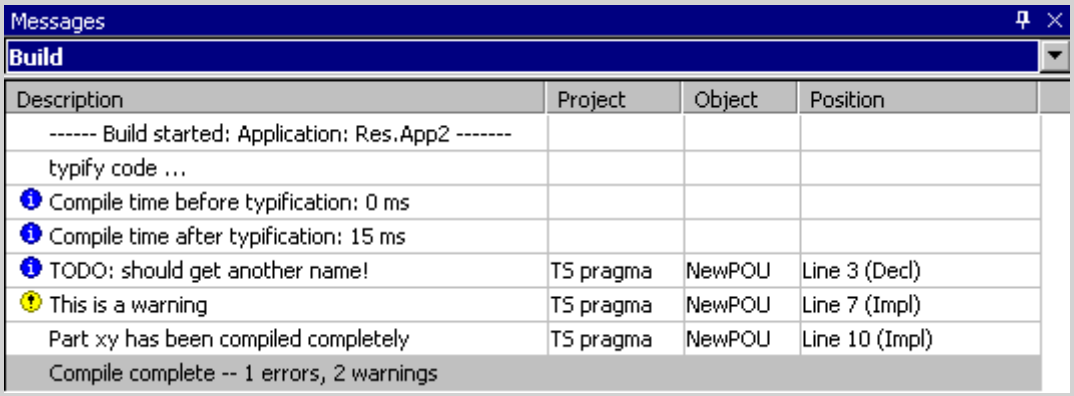
Pragma	Message type
`{text <'textstring'>}`	*"Text"*: display of <textstring>.
`{info <'textstring'>}`	*"Information "*: display of <textstring>.
`{warning <'textstring'>}`	*"Warning"*: display of <textstring>. Unlike the attribute pragma `'obsolete'`, you define the warning locally for the current position.
`{error <'textstring'>}`	*"Error "*: display of <textstring>.

In the CODESYS Message window you can jump with the help of the commands "Next Message" and "Previous Message" from a message of the category "Information", "Warning" and "Error" to the source position of the message. This means you jump to the position where the pragma is added in the source code.

Example

```
VAR
    var : INT; {info 'TODO: should get another name'}
 bvar : BOOL;
 arrTest : ARRAY [0..10] OF INT;
 i:INT;

END_VAR
    arrTest[i] := arrTest[i]+1;
 ivar:=ivar+1;

 {warning 'This is a warning'}
 {text 'Part xy has been compiled completely'}
```

Display in the Message window:



See also

- *Chapter 1.4.1.19.6.3 "Conditional Pragmas" on page 732*

Attribute Pragmas

Attribute pragmas affect the compilation and the pre-compilation.

CODESYS supports a series of pre-defined attribute pragmas. In addition you can use user-defined pragmas, which you can query with the help of conditional pragmas before the compilation of the project.

Attributes are defined within the declaration part. Exception: For the objects Action and Transition, which have no own declaration part, you can define the attributes at the beginning of the implementation part

When you define own attributes, please make them unambiguous. Uniqueness can be reached for example by adding a prefix to the attribute name. OEMs can use the vendor prefix for this purpose.

User-defined attributes

User-defined attributes are any application-defined or user-defined attributes that you can apply to POUs, actions, data type definitions and variables. You can query a user-defined attribute with the help of conditional pragmas before the compilation of the application.

You can query user-defined attributes with conditional pragmas with the operator `hasattribute`*.*

More detailed information and examples can be found in the chapter 'Conditional pragmas'.

Syntax:

```
{attribute 'attribute'}
```

Example for POUs and actions

Attribute `'vision'` for function *"fun1"*

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
 i : INT;
END_VAR
```

Example for variables

Attribute `'DoCount'` for variable `ivar`:

```
PROGRAM PLC_PRG
VAR
 {attribute 'DoCount'};
 ivar:INT;
 bvar:BOOL;
END_VAR
```

Example for data types

Attribute `'aType'` for data type `DUT_1`:

```
{attribute 'aType'}
TYPE DUT_1 :
STRUCT
 a:INT;
 b:BOOL;
END_STRUCT
END_TYPE
```

See also

- *Chapter 1.4.1.19.6.3 “Conditional Pragmas” on page 732*

Attribute 'call_after_global_init_slot'

> **NOTICE!**
> `VAR_INPUT` declarations in functions or methods that use the attribute lead to compile errors. Reason: Input variables are unknown in this case at the time of the call, which occurs implicitly during the online change.

The effect of this pragma is that all functions and programs containing this attribute are called after the global initialization. You define the order of calling by means of the attribute value.

Syntax:

```
{attribute 'call_after_global_init_slot' := '<slot>'}
```

`<slot>`: Integer value that defines the ranking in the order of the calls; the lower the value, the earlier the call takes place. If several function blocks have the same ranking for the attribute, then the order of their calls remains indefinite.

Insert location: First line above the declaration part of functions and programs

If a method possesses the attribute, CODESYS determines all instances of the corresponding function block and calls all instances in the specified slot. In this case you have no influence on the order of the instances among themselves.

See also

- *Chapter 1.4.1.19.10 “Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'” on page 748*

Attribute 'call_after_init'

> **NOTICE!**
> `VAR_INPUT` declarations in functions or methods that use the attribute lead to compile errors. Reason: Input variables are unknown in this case at the time of the call, which occurs implicitly during the online change.

The effect of this pragma is that a method is called implicitly after the initialization of a function block instance. For reasons of performance you must add the attribute both to the function block and to the method in its own first line above the declaration part.

Syntax:

```
{attribute 'call_after_init'}
```

Call: First line above the declaration part of the method and the function block.

CODESYS calls the method after the method FB_init and after the variable values of an initialization expression in the instance declaration have become valid.

This functionality is supported from compiler version 3.4.1.0.

Example

Definition:

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB
... <function block definition>

{attribute 'call_after_init'}
METHOD FB_AfterInit
... <method definition>
```

The definition implements, for example, the following declaration in the subsequent code processing:

```
inst : FB := (in1 := 99);
```

Code processing:

```
inst.FB_Init();
inst.in1 := 99;
inst.FB_AfterInit();
```

This allows a reaction to the user-defined initialization in `FB_AfterInit`.

See also

- *Chapter 1.4.1.19.10 “Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'” on page 748*

Attribute 'call_after_online_change_slot'

> **NOTICE!**
> `VAR_INPUT` declarations in functions or methods that use the attribute lead to compile errors. Reason: Input variables are unknown in this case at the time of the call, which occurs implicitly during the online change.

The effect of this pragma is that all functions and programs containing this attribute are called after an online change. You define the order of calling by means of the attribute `<slot>`.

Syntax:

```
{attribute 'call_after_online_change_slot' := '<slot>'}
```

`<slot>`: Integer value that defines the ranking in the order of the calls; the lower the value, the earlier the call takes place. If several function blocks have the same ranking for the attribute, then the order of their calls remains indefinite.

Call: First line above the declaration part of functions and programs.

If a method possesses the attribute, then CODESYS determines all instances of the function block concerned. CODESYS calls all instances in the specified slot. In this case you have no influence on the order of the instances among themselves.

NOTICE!
Since the application cannot run during the online change, each code executed in this situation can lead to a jitter. Therefore, keep the extent of the executive code as small as possible.

See also

- *Chapter 1.4.1.20.3.6.6 “Command 'Online Change'” on page 1033*

Attribute 'call_before_global_exit_slot'

NOTICE!
`VAR_INPUT` declarations in functions or methods that use the attribute lead to compile errors. Reason: Input variables are unknown in this case at the time of the call, which occurs implicitly during the online change.

The effect of this pragma is that all functions and programs containing this attribute in a dedicated first line of their declaration are called before the GlobalExit. The GlobalExit takes place before a new download or a reset. Function blocks provided with an `FB_Exit` method are affected. The order of calling is defined by means of the attribute value.

Syntax:

```
{attribute 'call_before_global_exit_slot' := '<slot>'}
```

Insert location: First line above the declaration part of functions and programs.

`<slot>`: Integer value that defines the ranking in the order of the calls; the lower the value, the earlier the call takes place. If several function blocks have the same ranking for the attribute, then the order of their calls remains indefinite.

If a method possesses the attribute, then the method is called for all instances of the function block concerned. CODESYS calls all instances in the specified slot. In this case you have no influence on the order of the instances among themselves.

See also

- *Chapter 1.4.1.19.10 “Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'” on page 748*

Attribute 'call_on_type_change'

With this pragma, you can mark a method of a function block A that should be called when the data type changes for one or more function blocks B, C, etc. that are referenced by A. The referencing can be defined by a pointer variable or a `REFERENCE` variable.

Syntax:

```
{attribute 'call_on_type_change':= '<name of the first referenced
function block>|<name of the second referenced function block>|<name
of the ... referenced function block>'}
```

Insert location: Line above the first line in the method declaration.

Example

Function blocks with references

```
FUNCTION_BLOCK FB_A
...
VAR
    var_pt: POINTER TO FB_B;
    var_ref: REFERENCE TO FB_C;
END_VAR
...
```

Method for reaction to a type change in the references FB_B and FB_C

```
{attribute 'call_on_type_change' := 'FB_B, FB_C'}
METHOD METH_react_on_type_change : INT
VAR_INPUT
...
```

Attribute 'conditionalshow'

The pragma has the effect that the identifiers of an integrated compiled library `<library name> .compiled-library`, which are decorated with the pragma, are hidden before programming an application. The POUs can be called but the variables are invisible in the CODESYS user interface.

Affected features

- Library management
- Debugging
- Input Assistant
- Function "List components"
- Monitoring
- Symbol configuration

This is useful when you develop libraries. As the library developer, you decorate function blocks or variables with the pragma. As a result, you determine which identifiers are hidden in an application after integration. If you want to show the hidden identifiers later, for example for debugging or further development of the library, you can reactivate its visibility.

Syntax

```
{attribute 'conditionalshow' ( := ' <some text> ' )? }
```

`<some text>`: Optional string literal to control the visibility of the identifiers decorated with this kind of pragma by means of a command-line command and this literal. When the pragma is specified without a literal, the variables in the CODESYS development environment are always hidden, regardless of how CODESYS was started. For more help about this, see the document "Library Development Summary".

Insert location: Top line in the declaration part of a function block, above a variable

Example For more examples, see the document "Library Development Summary".

Hiding a variable

```
FUNCTION_BLOCK FB_DataManager
VAR
    {attribute 'conditionalshow' := 'Library_Developer'}
    iLocal : INT;
    iCounter : INT;
END_VAR
```

The variable `iLocal` is invisible.

Hiding a function block

```
{attribute 'conditionalshow' := 'Library_Developer'}
FUNCTION_BLOCK FB_DataManager
VAR
    iLocal : INT;
    iCounter : INT;
END_VAR
```

The identifiers `FB_DataManager`, `iLocal`, and `iCounter` are invisible.

Visibility in case of existing source code file

When the source code file `<library name> .library` from an integrated library also exists at the same memory location (repository), the identifiers are visible despite the pragmas. That is regardless of whether or not an attribute value has been specified in the declaration.

Command-line call to activate visibility

You can also enable the visibility of the hidden variable without a source code file by starting CODESYS with the command-line option `conditionalshowsymbols`. To enable the visibility, specify the attribute values of the pragma which are separated by commas.

Syntax

```
codesys.exe --conditionalshowsymbols=" <some text> ( ,<next text> )*
"
```

Example

```
codesys.exe --conditionalshowsymbols="Library_Developer"
codesys.exe --conditionalshowsymbols="Group_A,Group_B"
```

See also

- *Chapter 1.4.1.15 “Using the Command-Line Interface” on page 442*
- *Chapter 1.4.1.19.6.2.16 “Attribute 'hide'” on page 700*
- *Chapter 1.4.1.19.6.2.8 “Attribute 'conditionalshow_all_locals'” on page 691*
- "Library Development Summary", "Visibility Control" Chapter

Attribute 'conditionalshow_all_locals'

The pragma has the effect that **all local** variables of a library POU decorated with the pragma are hidden from application programmers. The POUs of an integrated compiled library `<library name> .compiled-library` can be called, but the variables are invisible in the CODESYS user interface.

Affects features:

- Library management
- Debugging
- Input Assistant
- Function "List components"
- Monitoring
- Symbol configuration

This is useful when you develop libraries. As the library developer, you decorate function blocks with the pragma. As a result, you determine that their identifiers are hidden in an application after integration. If you want to show these identifiers later, for example for debugging or further development of the library, you can reactivate its visibility.

Syntax

```
{attribute 'conditionalshow_all_locals' ( := ' <some text> ' )? }
```

`<some text>`: Optional string literal to control the visibility of the identifiers decorated with this kind of pragma by means of a command-line command and this literal. When the pragma is specified without a literal, the variables in the CODESYS development environment are always hidden, regardless of how CODESYS was started. For more help about this, see the document "Library Development Summary".

Insert location: Top line in the declaration part of the function block.

Example

Hiding all local variables

```
{attribute 'conditionalshow_all_locals' := 'Library_Developer'}
FUNCTION_BLOCK FB_DataManager
VAR
    iLocal : INT;
    iCounter : INT;
END_VAR
```

For more examples, see the document "Library Development Summary".

Visibility in case of existing source code file

When the source code file `<library name> .library` from an integrated library also exists at the same memory location (repository), the library POU variables are visible despite the pragmas. That is regardless of whether or not an attribute value has been specified in the declaration.

Command-line call to activate visibility

You can also enable the visibility of the hidden variable without a source code file by starting CODESYS with the command-line option `conditionalshowsymbols`. To enable the visibility, specify the attribute values of the pragma which are separated by commas.

Syntax

```
codesys.exe --conditionalshowsymbols=" <some text> ( ,<next text> )*
"
```

Example

```
codesys.exe --conditionalshowsymbols="Library_Developer"
codesys.exe --conditionalshowsymbols="Group_A,Group_B"
```

See also

- *Chapter 1.4.1.19.6.2.17 “Attribute 'hide_all_locals'” on page 703*
- *Chapter 1.4.1.19.6.2.7 “Attribute 'conditionalshow'” on page 690*
- "Library Development Summary", chapter "Visibility Control"

Attribute 'const_replaced', Attribute 'const_non_replaced'

The attribute `'const_replaced'` has the effect that the constant is replaced in the code, independently of the setting of the *“Replace constants”* compiler option. The attribute has an effect for variables of scalar types only, but not for compound types like arrays and structures.

You insert the pragma `{attribute 'const_non_replaced'}` accordingly in order to explicitly deactivate the *“Replace constants”* compiler option. This has the effect, for example in the symbol configuration, that the constant is available and can be exported despite the compiler option.

The *“Replace constants”* option in the *“Compile Options”* category of the *“Project Settings”* dialog is preset for the entire project, because replacing constants generally leads to faster code and less memory usage.

Syntax:

```
{attribute 'const_replaced'}
```

```
{attribute 'const_non_replaced'}
```

Insert location: Line above the declaration line of the global variables.

Example

The constants `iTestCon` and `xTestCon` are available in the symbol configuration because the *“Replace constants”* option deactivated.

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
    {attribute 'const_non_replaced'}
    iTestCon    :    INT  := 12;
    {attribute 'const_non_replaced'}
    xTestCon    :    BOOL := TRUE;
    rTestCon    :    REAL := 1.5;
END_VAR

VAR_GLOBAL
    iTestVar    :    INT  := 12;
    xTestVar    :    BOOL := TRUE;
END_VAR
```

See also

- *Chapter 1.4.1.20.4.11.3 “Dialog Box 'Project Settings' - 'Compileoptions'” on page 1173*
- *Chapter 1.4.1.9.2 “Symbol Configuration” on page 357*

Attribute 'dataflow'

With this pragma you control the data flow in the processing of function blocks in the FBD/LD/IL editor. The attribute defines the input or output of a function block to which the continuing connection to the next or previous function block is connected.

You may provide only 1 input and 1 output with the attribute in the declaration of a function block.

Syntax:

```
{attribute 'dataflow'}
```

Insertion position: line above the line with the declaration of the corresponding variables.

In the case of function blocks without the attribute `'dataflow'`, CODESYS determines the data flow as follows: first of all the connection is placed between an output and an input of same data type. The highest input or output variable of the function blocks is always taken. If there are no variables of a corresponding data type, CODESYS connects the highest output with the highest input of the neighboring function blocks.

Example

The connection between `FB` and the preceding function block is established via the input variable `i1`. The connection between `FB` and the following function block is established via the output variable `outRes1`.

```
FUNCTION_BLOCK FB
VAR_INPUT
 r1 : REAL;
 {attribute 'dataflow'}
 i1 : INT;
 i2 : INT;
 r2 : REAL;
END_VAR

VAR_OUTPUT
{attribute 'dataflow'}
 outRes1 : REAL;
 out1 : INT;
 g1 : INT;
 g2 : REAL;
END_VAR
```

See also

- *Chapter 1.4.1.8.3.1.1 “Programming function block diagrams (FBD)” on page 237*

Attribute 'displaymode'

With this pragma you define the display mode of an individual variable. This definition overwrites the global setting for the display of the monitoring variable, which takes place via the commands in the menu *“Debug ➔ Display Mode”*.

Syntax:

```
{attribute 'displaymode':=<displaymode>}
```

The following definitions are possible

- Binary format
  - `{attribute 'displaymode':='bin'}`
  - `{attribute 'displaymode':='binary'}`
- Decimal format
  - `attribute 'displaymode':='dec'}`
  - `{attribute 'displaymode':='decimal'}`
- Hexadecimal format
  - `{attribute 'displaymode':='hex'}`
  - `attribute 'displaymode':='hexadecimal'}`

Insertion position: line above the line with the declaration of the corresponding variables.

Example

```
VAR
 {attribute 'displaymode':='hex'}
 dwVar1: DWORD;
END_VAR
```

See also

- *Chapter 1.4.1.20.3.7.24 “Command 'Display Mode' - 'Binary', 'Decimal', 'Hexadecimal'” on page 1058*

Attribute 'enable_dynamic_creation'

The pragma `enable_dynamic_creation` is needed for using the `__NEW` operator for function blocks.

Syntax:

```
{attribute 'enable_dynamic_creation'}
```

Insert location: First line in the declaration of the function block.

See also

- *Chapter 1.4.1.19.3.58 “Operator '__NEW'” on page 614*

Attribute 'estimated-stack-usage'

The pragma provides an estimated value for the stack size requirement.

Methods with recursive calls cannot pass a stack check because stack usage cannot be determined. As a result, a warning is issued. To prevent this warning, you can give the method an estimated value (in bytes) for the stack size requirement. Then the method passes the stack check successfully.

Syntax

```
{attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'}
```

Example

```
{attribute 'estimated-stack-usage' := '127'} // 127 bytes
METHOD PUBLIC DoIt : BOOL
VAR_INPUT
END_VAR
```

Insert location: First line above the declaration part of the method.

The section "Method call" includes an example that uses this pragma.

Recursive method call

Within its implementation, a method can call itself, either directly by means of the `THIS` pointer, or by means of a local variable for the assigned function block.

Use recursions mainly for processing recursive data types such as linked lists. In general, we recommend to be careful when using recursion, as unexpectedly deep recursions can cause stack overflow and machine downtime.

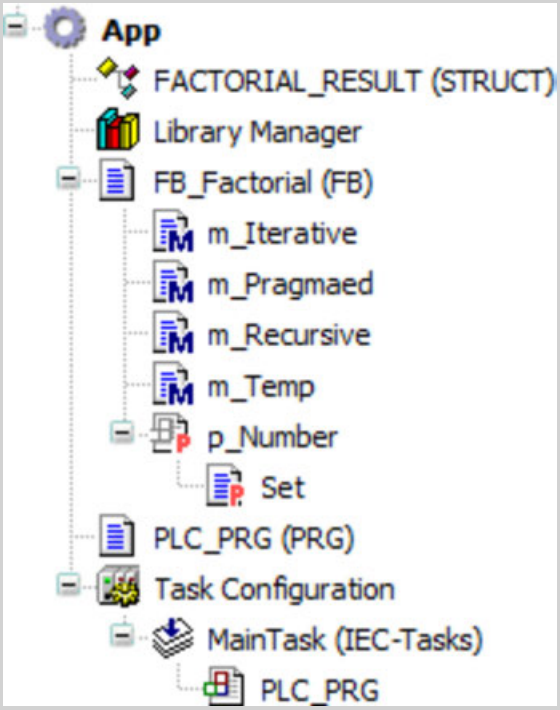
Example

Calculation of the factorial

The following program `PLC_PRG` calculates the factorial of a number in the `FB_Factorial` function block in a different way, each in its own method.

- Method `m_Iterative`: Iterative
- Method `m_Pragmaed`: Recursive with warning suppression
- Method `m_Recursive`: Recursive
- Method `m_Temp`: Temporary with warning suppression

A warning is issued for the `m_Recursive` method only.

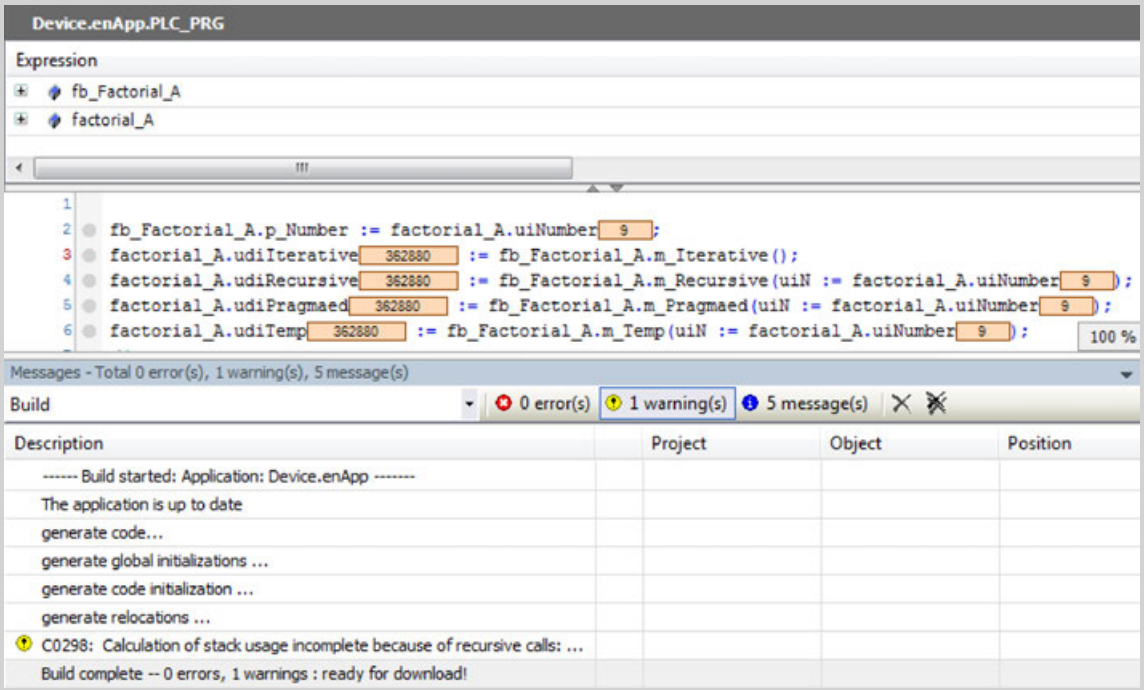



```
// Contains the data  of the factorial calculation of uiNumber
TYPE FACTORIAL_RESULT :
STRUCT
    uiNumber : UINT;
    udiIterative : UDINT;
    udiRecursive : UDINT;
    udiPragmaed : UDINT;
    udiTemp : UDINT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    fb_Factorial_A : FB_Factorial;
    factorial_A : FACTORIAL_RESULT := (uiNumber := 9,
udiIterative := 0, udiRecursive := 0, udiPragmaed := 0 );
END_VAR
fb_Factorial_A.p_Number := factorial_A.uiNumber;
factorial_A.udiIterative := fb_Factorial_A.m_Iterative();
factorial_A.udiRecursive := fb_Factorial_A.m_Recursive(uiN :=
factorial_A.uiNumber);
factorial_A.udiPragmaed := fb_Factorial_A.m_Pragmaed(uiN :=
factorial_A.uiNumber);
factorial_A.udiTemp := fb_Factorial_A.m_Temp(uiN :=
factorial_A.uiNumber);
```

```
//Factorial calculation in different ways
FUNCTION_BLOCK FB_Factorial
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    uiN : UINT;
    udiIterative : UDINT;
    udiPragmaed : UDINT;
    udiRecursive : UDINT;
END_VAR

// Iterative calculation
METHOD PUBLIC m_Iterative : UDINT
VAR
    uiCnt : UINT;
END_VAR
m_Iterative := 1;
IF    uiN > 1 THEN
    FOR uiCnt := 1 TO uiN DO
        m_Iterative := m_Iterative * uiCnt;
    END_FOR;
    RETURN;
ELSE
    RETURN;
END_IF;

//Recursive calculation with suppressed warning
{attribute 'estimated-stack-usage' := '99'}
METHOD PUBLIC m_Pragmaed : UDINT
VAR_INPUT
    uiN : UINT;
END_VAR
VAR
END_VAR
m_Pragmaed := 1;
IF    uiN > 1 THEN
    m_Pragmaed := uiN * THIS^.m_Pragmaed(uiN := (uiN - 1));
    RETURN;
ELSE
    RETURN;
END_IF;

//Recursive calculation
METHOD PUBLIC m_Recursive : UDINT
VAR_INPUT
    uiN : UINT;
END_VAR
VAR
END_VAR
m_Recursive := 1;
IF    uiN > 1 THEN
    m_Recursive := uiN * THIS^.m_Recursive(uiN := (uiN - 1) );
    RETURN;
ELSE
    RETURN;
END_IF;

// Called by temporary FB instance
{attribute 'estimated-stack-usage' := '99'}
METHOD PUBLIC m_Temp : UDINT
VAR_INPUT
    uiN : UINT;
END_VAR
```

```
VAR
    fb_Temp : FB_Factorial;
END_VAR
m_Temp := 1;
IF    uiN > 1 THEN
    m_Temp := uiN * fb_Temp.m_Temp(uiN := (uiN - 1));
    RETURN;
ELSE
    RETURN;
END_IF;

PROPERTY p_Number : UINT
uiN := p_Number; //Setter method
```

Only the `m_Recursive` issues a warning when the program is executed.



See also

- *Chapter 1.4.1.8.22.4 “Calling methods” on page 314*
- *Chapter 1.4.1.8.22 “Object-Oriented Programming” on page 310*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*

Attribute 'ExpandFully'

The effect of this pragma is that the components of an array used as an input variable for referenced visualizations are made visible in the Properties dialog box of the visualization.

Syntax:

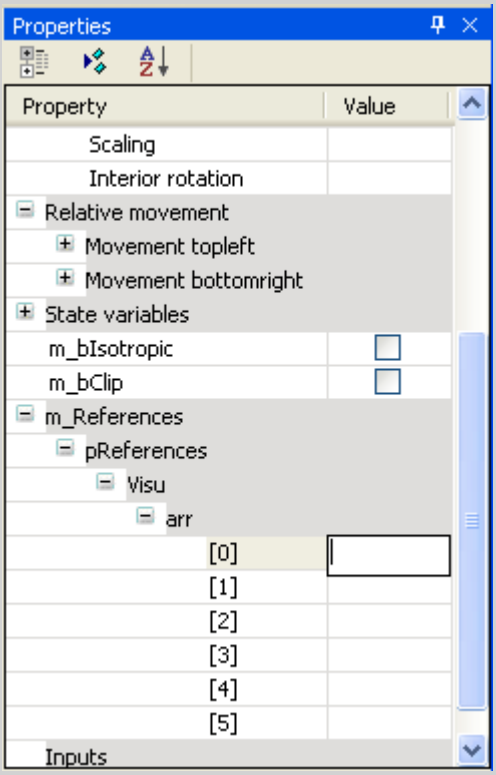
```
{attribute 'ExpandFully'}
```

Insertion position: the line above the line with the declaration of the array.

Example

The visualization *“visu”* is to be inserted into a frame inside the visualization *“visu_main”*. `arr` is defined as an input variable in the interface editor of *“visu”* and will thus be available later for assignments in the Properties dialog box of the frames in *“visu_main”*. In order to also make the individual components of `arr` available in this Properties dialog box, you must insert the attribute `'ExpandFully'` directly before `arr` in the interface editor of visu. Declaration in the interface editor of *“visu”*:

```
VAR_INPUT
 {attribute 'ExpandFully'}
 arr : ARRAY[0..5] OF INT;
END_VAR
```



Attribute 'global_init_slot'

The pragma defines the initialization order of programming blocks and global variable lists.

Variables in a list (GVL or POU) are initialized from top to bottom.

If there are several global variable lists, then the initialization order is not defined.

The initialization does not apply for the initialization of literal values, for example `1`, `'hello'`, `3.6`, or constants of base data types. However, you must define the initialization order yourself if there are dependencies between the lists. You can assign a defined initialization slot to a GVL or POU with the `'global_init_slot'` attribute.

Constants are initialized before the variables and in the same order as the variables. During initialization, the POUs are sorted according to the value for <slot>. Then the code for initializing the constants is generated and afterwards the code for initializing the variables.

Syntax:

```
{attribute 'global_init_slot' := '<slot>'}
```

`<slot>`: Integer value that defines the position in the call order. The default value for a POU (program, function block) is 50000. The default value for a GVL is 49990. A lower value means an earlier initialization. Caution: If several blocks or GVLs receive the same value for the `'global_init_slot'` attribute, then the initialization order remains undefined.

Insert location: The pragma always affects the entire GVL or POU and therefore it must be located above the `VAR_GLOBAL` or POU declaration.

If several programming blocks have got assigned the same value for the attribute `'global_init_slot'`, the order of their initialization remains undefined.

Example

The program includes two global variable lists `GVL_1` and `GVL_2`, as well as a `PLC_PRG` program that uses variables from both lists. `GVL_1` uses the variable `B` for initializing a variable `A`, which is initialized in `GVL_2` with a value of `1000`.

GVL_1

```
VAR_GLOBAL   //49990
 A : INT := GVL_2.B*100;
END_VAR
```

GVL_2

```
VAR_GLOBAL   //49990
 B : INT := 1000;
 C : INT := 10;
END_VAR
```

PLC_PRG

```
PROGRAM PLC_PRG  //50000
VAR
 ivar: INT := GVL_1.A;
 ivar2: INT;
END_VAR

ivar:=ivar+1;
ivar2:=GVL_2.C;
```

In this case, the compiler prints an error because `GVL_2.B` is used for initializing `GVL_1.A` before `GVL_2` has been initialized. You can prevent this by using the `global_init_slot` attribute to position `GVL_2` before `GVL_1` in the initialization sequence.

In this example, `GVL_1` must have at least one slot value of 49989 in order to achieve the earliest initialization within the program. Every lower value has the same effect:

GVL_2

```
{attribute 'global_init_slot' := '100'}
VAR_GLOBAL
  B : INT := 1000;
END_VAR
```

Using `GVL_2.C` in the implementation part of `PLC_PRG` is also not critical even without using a pragma because both GVLs are initialized before the program in either case.

Attribute 'hide'

Using the pragma `{attribute 'hide'}` to hide variables and POUs does not have the desired effect in most cases. Instead, you should use the pragma `{attribute 'conditionalshow'}`.

The pragma prevents the variables and POUs defined with it from being shown in the CODESYS user interface. As a result, you can intentionally hide these identifiers without restricting the access. This can be useful when you develop libraries.

Affected features:

- Library management
- Debugging
- Input Assistant
- Function "List components"

- Monitoring
- Symbol configuration

The variables or POUs defined with the pragma are neither visible in the Library Manager nor are they suggested in the Input Assistant or in the "List components" function. The pragma prevents those marked variables from being displayed in the symbol configuration. As a result, you cannot export these kinds of variables as symbols. The variables are also invisible in online mode, and therefore their values cannot be monitored. Moreover, you cannot use any debugging functionalities and you do not have any support when checking for bugs.

Syntax:

```
{attribute 'hide'}
```

Insert location: For variables, above the line with the declaration of the variables. For POUs, in the first line.

If you, the application developer, know the exact instance path of the hidden POUs and variables, then you can access them in the code.

Example of hidden variable

The function block `FB_MyA` contains the attribute pragma `{attribute 'hide'}` to hide the local variable `xInvisibleIn`.

```
FUNCTION_BLOCK FB_MyA
VAR_INPUT
    iInA : INT;
    {attribute 'hide'}
    xInvisibleIn : BOOL;
    xInit: BOOL;
END_VAR
VAR_OUTPUT
     iOutA : INT;
END_VAR
VAR
    iCounter : INT;
END_VAR
```

Two instances of the function block `FB_MyA` are defined in the main program.

```
PROGRAM PLC_PRG
VAR
    fbMyA1, fbMyA2 : FB_MyA;
    xVar2 : BOOL;
    iVar1 : INT;
    iVar2 : INT;
END_VAR
fbMyA1(iInA := 1, xInit := TRUE, xInvisibleIn := TRUE, iOutA =>
iVar1);
fbMyA2(iInA := 1, xInit := TRUE, iOutA => iVar2);
```

When the input value for `fbMyA1` is implemented, the "List components" function, which opens when you type `fbMyA1.` (in the implementation part of `PLC_PRG`), displays the variables `iInA`, `xInit`, and `iOutA`, but not the hidden variable `xInvisibleIn`.

Example of hidden library POU

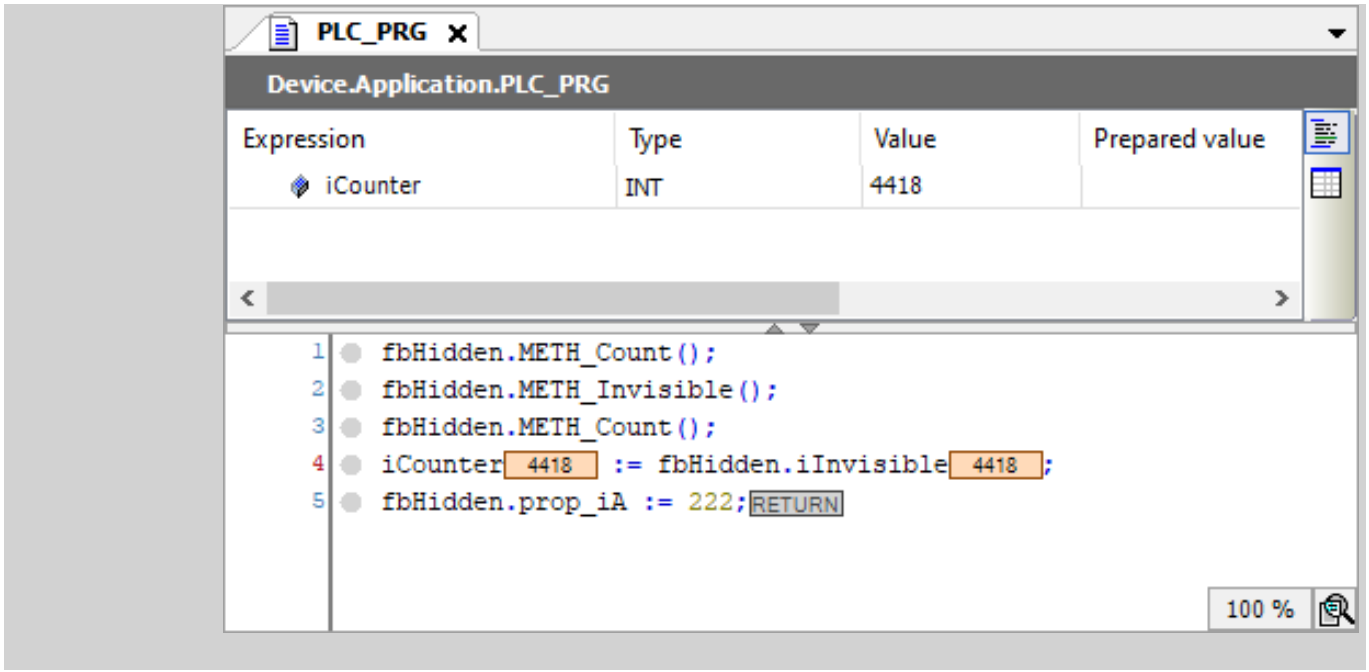
`FB_A` is a function block of the library `HiddenFunctionality` with the default namespace `HIDDEN`. To hide the identifier and the POU code from application developers, begin the declaration of the POU with the attribute pragma `{attribute 'hide'}`. To hide the subordinate POUs (actions, methods, properties, and transitions) in the same way, also begin their declarations with `{attribute 'hide'}`.

```
{attribute 'hide'}
FUNCTION_BLOCK FB_A
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iA : INT;
    iCount : INT;
    iInvisible : INT;
END_VAR

{attribute 'hide'}
METHOD METH_Count : INT
VAR_INPUT
END_VAR
iCount := iCount + 1;

{attribute 'hide'}
METHOD METH_Invisible : BOOL
VAR_INPUT
END_VAR
iInvisible := iInvisible + 1;

{attribute 'hide'}
PROPERTY PUBLIC prop_iA : INT
```

For you as the application developer, all POUs are invisible. You can use them only if you know the instance path.

```
PROGRAM PLC_PRG
VAR
    fbHidden : HIDDEN.FB_A; // Hidden function block from library
HiddenFunctionality
    iCounter : INT;
END_VAR
fbHidden.METH_Invisible();
iCounter := fbHidden.iInvisible;
```

In online mode, no monitoring is performed.



With the pragma `hide_all_locals` you can hide all local variables of a POU.

See also

- *Chapter 1.4.1.19.6.2.17 “Attribute 'hide_all_locals'” on page 703*
- *Chapter 1.4.1.19.6.2.7 “Attribute 'conditionalshow'” on page 690*
- *Chapter 1.4.1.19.6.2.8 “Attribute 'conditionalshow_all_locals'” on page 691*

Attribute 'hide_all_locals'

The pragma prevents all local variables of a signature from being visible in the display of the 'List components' function, in the Input Assistant or in the declaration part in online mode. Moreover, these variables are hidden in the symbol configuration and therefore cannot be exported as symbols. The pragma is especially useful in library POUs to hide POU variables from users.

Affected features

- Library management
- Debugging
- Input Assistant
- Function "List components"
- Monitoring
- Symbol configuration

Syntax:

```
{attribute 'hide_all_locals'}
```

Insert location: First line above the declaration part of the POU

Example

The function block `FB_MyB` uses the attribute:

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK FB_MyB
VAR_INPUT
    iInB : INT;
    {attribute 'hide'}
    xInvisibleIn : BOOL;
    xInit: BOOL;
END_VAR
VAR_OUTPUT
     iOutB : INT;
END_VAR
VAR
    iCounter : INT;
    xVar : BOOL;
END_VAR
```

Two instances of the function block `FB_MyB` are defined in the main program.

```
PROGRAM PLC_PRG
VAR
    fbMyB1, fbMyB2: FB_MyB;
    iVar3: INT;
    iVar4: INT;
END_VAR

fbMyB1(iInB := 2, xInvisibleIn := TRUE, iOutB => iVar3);
fbMyB2(iInB := 2, iOutB => iVar4);
IF fbMyB2.iCounter > 100 THEN
    fbMyB2.xInit := TRUE;
END_IF
```

Now when you download the program to the controller, start it, and switch to online mode, the variables `iInB`, `xInit`, `iOutB`, and `xReset` are displayed in the declaration editor. However, the hidden local variables `iCounter` and `xVar` are not displayed.

See also

- *Chapter 1.4.1.19.6.2.16 “Attribute 'hide'” on page 700*

Attribute 'initialize_on_call'

The pragma causes input variables of a function block to be initialized on each call of the function block. If an input variable is affected which expects a pointer and this pointer has been removed during an online change, then the variable is initialized to `zero`.

Syntax:

```
{attribute 'initialize_on_call'}
```

Insert location: Always in the first line of the declaration part for the entire function block, and also in a line above the declaration of the individual input variable.

Example

```
{attribute 'initialize_on_call'}
FUNCTION_BLOCK fb
VAR_INPUT
                {attribute 'initialize_on_call'}
                pInt : POINTER TO INT := 0;
                {attribute 'initialize_on_call'}
                iVal : INT := 0;
END_VAR
```

Attribute 'init_namespace'

The effect of this pragma is that a variable of the type `STRING` or `WSTRING`, which is declared in a library function block with this pragma, is initialized when used in the project with the current namespace of the library.

Syntax

```
{attribute 'init_namespace'}
```

Insertion position: the line above the line with the declaration of the variables in a library function block.

Example

The function block *"POU"* is provided with the necessary attributes:

```
FUNCTION_BLOCK POU
VAR_OUTPUT
 {attribute 'init_namespace'}
 myStr: STRING;
END_VAR
```

An instance `fb` of the function block POU is defined within the main program PLC_PRG:

```
PROGRAM PLC_PRG
VAR
 fb:POU;
 newString: STRING;
END_VAR
 newString := fb.myStr;
```

The variable `myStr` is initialized with the current namespace, for example `MyLib`. This value is assigned to `newString` in the main program.

See also

- *Chapter 1.4.1.20.2.14 "Object 'Library Manager'" on page 874*

Attribute 'init_on_onlchange'

The effect of this pragma is that the variable to which the pragma is applied is initialized with each online change.

NOTICE!

For compiler version 3.5.0.0 and later, a fast online change is performed for minor changes. In this case, only the modified blocks are compiled and downloaded. In particular, no initialization code is generated. This means that also no code is generated when variables with the `init_on_onlchange` attribute are initialized. As a rule, this has no effect because the attribute is used primarily for initializing variables with addresses. However, it cannot happen that a variable changes its address during an online change.

To secure the effect of the `init_on_onlchange` attribute in the entire application code, you must deactivate the fast online change in general for the application by using the compiler definition `no_fast_online_change`. To do this, insert the definition in the application *"Properties"* (*"Build"* tab).

Syntax:

```
{attribute 'init_on_onlchange' }
```

Insert location: The line above the line with the declaration of the variables.

See also

- *Chapter 1.4.1.20.4.10.4 "Dialog 'Properties' - 'Build'" on page 1159*

Attribute 'instance-path'

This pragma can be applied to a local STRING variable and causes this local STRING variable to be initialized in sequence with the device tree path of the POU to which it belongs. This can be useful for error messages. The application of the pragma requires the application of the attribute `'reflection'` to the associated POU, as well as the application of the additional attribute `'noinit'` to the STRING variable.

Syntax:

```
{attribute 'instance-path'}
```

Insertion position: the line above the line with the declaration of the `STRING` variable.

Example

The following function block contains the attributes `'reflection'`, `'instance-path'` and `'noinit'`.

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
 {attribute 'instance-path'}
 {attribute 'noinit'}
 str: STRING;
END_VAR
```

An instance *"myPOU"* of the function block *"POU"* is defined within the main program *"PLC_PRG"*:

```
PROGRAM PLC_PRG
VAR
 myPOU:POU;
 myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

Following the initialization of the instance `myPOU`, the path of the instance myPOU is assigned to the string variable `str`, in the example `PLCWinNT.Application.PLC_PRG.myPOU`. This path is assigned in the main program to the variable `myString`.

NOTICE!
You can define the length of a string to be whatever you like (even >255), but you must consider that the string will be truncated at the end if it is assigned to a variable whose data type is too small for it.

See also

- *Chapter 1.4.1.19.6.2.39 “Attribute 'reflection'” on page 727*
- *Chapter 1.4.1.19.6.2.30 “Attribute 'noinit'” on page 713*

Attribute 'io_function_block', 'io_function_block_mapping'

With the `'io_function_block'` attribute, you mark a function block in order to prepare it for the assignment to a channel in the I/O mapping of the device configuration. Then it is shown in the *“Select function block”* dialog.

With the `'io_function_block_mapping'` attribute, you mark a parameter that should be used when mapping the FB to a device channel in this kind of function block. You can provide the attribute to multiple parameters of the function block. For I/O mapping, the first one is used automatically whose type matches the channel (input, output, data type).

Syntax:

```
{attribute 'io_function_block'}
{attribute 'io_function_block_mapping'}
```

Insert location: The line above the first line in the declaration of the function block, or the line above the parameter declaration.

Example

```
{attribute 'io_function_block'}
FUNCTION_BLOCK Scale_Output_Int
VAR_INPUT
    iInput : INT;
    iNumerator : INT;
    iDenominator : INT :=1;
    iOffset : INT := 0;
END_VAR
VAR_OUTPUT
 {attribute 'io_function_block_mapping'}
    iOutput : INT;
END_VAR
VAR
```

See also

- *Chapter 1.4.1.20.4.3 “Dialog 'Select Function Block'” on page 1150*
- *“Linking a device with a function block instance” on page 218*

Attribute 'is_connected'

You use the pragma `'is_connected'` to mark a Boolean function block variable which, when a function module instance is called, provides information about whether the associated input of the POU has an assignment.

The use of the pragma requires the use of the attribute `'reflection'` on the affected function block.

Syntax:

```
{attribute 'is_connected' := '<input variable>'}
```

Example

In the function block `FB`, a local variable is declared for each input variable (`in1` and `in2`) and the attribute `'is connected'` is prepended to it each time with the name of the input variable. The func itself gets the pragma attribute `'reflection'`.

When an instance of the function block is called, the local variable is `TRUE` in the case that the input assigned to it has received an assignment.

```
{attribute 'reflection'}
FUNCTION_BLOCK FB
VAR_INPUT
  in1: INT;
  in2: INT;
END_VAR
VAR
  {attribute 'is_connected' := 'in1'}
  in1_connection_info: BOOL;
  {attribute 'is_connected' := 'in2'}
  in2_connection_info: BOOL;
END_VAR
```

Assumption: When the function block instance is called, `in1` receives an external assignment and `in 2` does not receive an assignment. This results in the following code:

```
in1_connection_info := TRUE;
in2_connection_info := FALSE;
```

See also

- *Chapter 1.4.1.19.6.2.39 “Attribute 'reflection'” on page 727*
- *Chapter 1.4.1.20.2.18.2 “Object 'Function Block'” on page 883*

Attribute 'linkalways'

The pragma `{attribute 'linkalways'}` instructs the compiler to always include a POU or a library POU in the compile information. During the build, the POU is compiled and is part of the application code. During the download, the POU is downloaded to the PLC.

Syntax:

```
{attribute 'linkalways'}
```

Insertion location: The first line in the declaration part of the POU or library POU

The POU may be valid throughout the project (saved in the *“POUs”* view) or throughout the application (saved in the *“Devices”* view).

You can also select the “Link always” option in the “Build” tab of a POU's object properties.

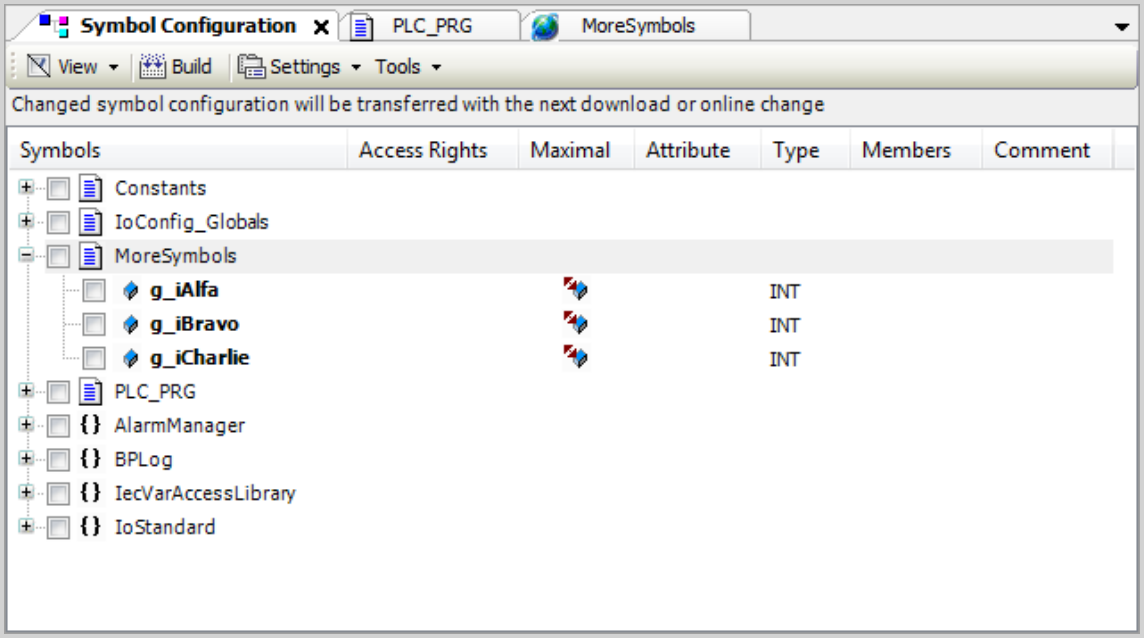
Example

The *"MoreSymbols"* GVL contains the pragma `{attribute 'linkalways'}`. The variables declared there are also part of the application code, regardless of any access.

GVL `MoreSymbols`

```
{attribute 'linkalways'}
VAR_GLOBAL
    g_iAlpha: INT;
    g_iBravo: INT;
    g_iCharlie: INT;
END_VAR
```

The symbol configuration also accesses the compile information. As a result, the variables of the `MoreSymbols` GVL are always provided for selection in the *"Symbol Configuration"* editor.



See also

- *Chapter 1.4.1.20.4.10.4 "Dialog 'Properties' - 'Build'" on page 1159*
- *Chapter 1.4.1.9.2 "Symbol Configuration" on page 357*

Attribute 'monitoring'

The effect of this pragma is that you can monitor values of properties or function calls in the online view of the IEC editor or in a watch list. There are two possible attribute values for this: `'variable'` and `'call'`

Syntax

```
{attribute 'monitoring' := 'variable'}
{attribute 'monitoring' := 'call'}
```

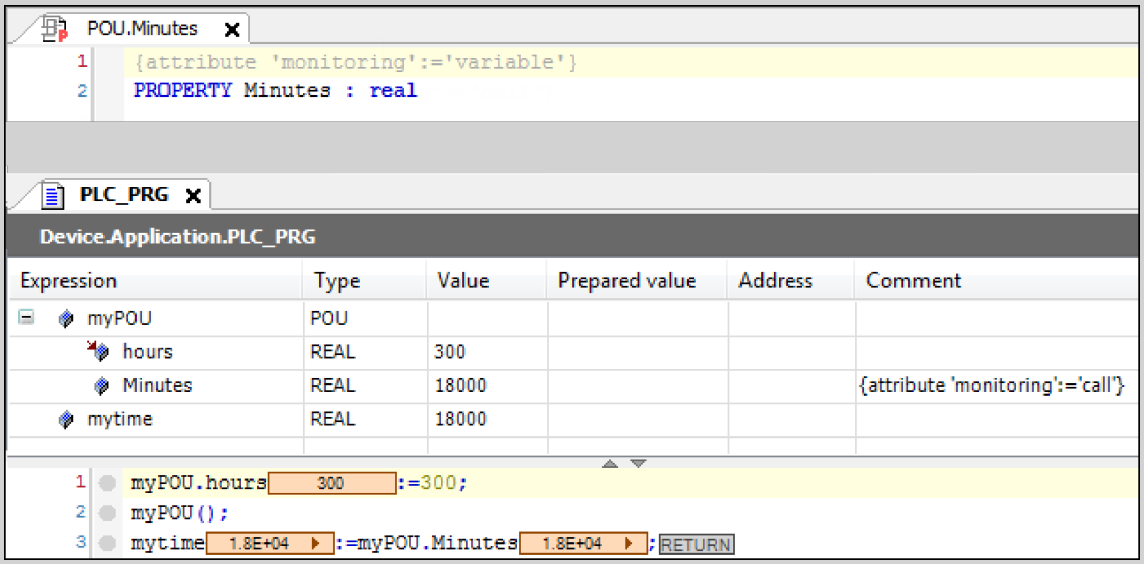
Monitoring of programming objects and their properties

In the online view of a function block or program, you can monitor the subordinate properties in addition to the local variables. This allows you to monitor the values of the `Get` and `Set` methods.

Insert either the pragma `{attribute 'monitoring' := 'variable'}` or `{attribute 'monitoring' := 'call'}` in the declaration of the property block. The current values of the property are then displayed automatically in the IEC editor or in a watch list.

Example

In online mode, the `PLC_PRG` object shows the value of the `Minutes` property at the call location inline in the ST editor. This is because the pragma `{attribute 'monitoring' := 'variable'}` is located in the declaration of the `Minutes` property.



Check carefully for each application which attribute pragma is suitable for displaying the desired value. This depends on whether further operations with the variables are implemented within the property.

1. Pragma `{attribute 'monitoring':='variable'}`:

An implicit variable is created for the property, which is then always given the current property value when the application calls the `Set` or `Get` method. The value stored last in this variable is displayed in the monitoring.

2. Pragma `{attribute 'monitoring':='call'}`:

You can use this attribute only for properties that return simple data types or pointers, but not for structured types. The value to be monitored is read or written by calling the property directly. This means that the monitoring service of the runtime executes the `Get` or `Set` method of the property.

NOTICE!
When you insert the pragma `{attribute 'monitoring':='call'}` for monitoring, you have to pay attention to possible side effects. These kinds of side effects can occur if additional operations are implemented in the property.

NOTICE!
The pragma `{attribute 'monitoring'}` is also evaluated for the symbol configuration. Only read access is possible for the value `'variable'`.

With the context menu command “Add Watch”, a variable on which the cursor is currently positioned is applied directly into the monitoring list in online mode.

The forcing or writing of functions is not supported. However, you can implicitly implement forcing by adding an additional input parameter for the respective function, which serves as an internal force flag.

Function monitoring is not possible in the compact runtime.

See also

- ↳ *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*

Attribute 'no_assign', Attribute 'no_assign_warning'

The pragma `'no_assign'` results in compiler errors being displayed if an instance of the function block is assigned to another instance of the same function block. Such assignments are often to be avoided if the function block contains pointers and pointers lead to problems, because they are copied as well during the value assignment.

The pragma `'no_assign_warning'` results in the same as for the pragma `'no_assign'` with compiler warnings instead of compiler errors.

Syntax:

```
{attribute 'no_assign'}
```

Insert location: First line in the declaration part of a function block.

Example

Assignment of function block instances containing pointers.

In this example the value assignment of the function block instances will lead to problems during the execution of `fb_exit`:

```
VAR_GLOBAL
inst1 : TestFB;
  awsBufferLogFile : ARRAY [0..9] OF WSTRING(66);(* Area: 0,
Offset: 0x1304 (4868)*)
   LogFile : SEDL.LogRecord := (sFileName := 'LogFile.log',
pBuffer := ADR(awsBufferLogFile), udiMaxEntriesFile := UDINT#10000,
udiMaxBuffered := UDINT#10, uiLineSize := UINT#64, wsSep := " ",
xCircular := TRUE, siDateFormat := SINT#0, siTimeFormat := SINT#0);
END_VAR

 PROGRAM PLC_PRG
VAR
  inst2 : TestFB := inst1;
  LogFileNew
END_VAR
```

In this case `LogRecord` manages a list of pointers, for which various actions are executed in the case of `fb_exit`. Problems result due to the assignment, because `fb_exit` will be executed twice. You should prevent this by adding the attribute `'no_assign'` in the declaration of the function block *"TestFB"*:

```
{attribute 'no_assign'}
FUNCTION_BLOCK TestFB
VAR_INPUT
...
```

The following compiler errors are then displayed:

```
C0328: Assignment not allowed for type TestFB
C0328: Assignment not allowed for type LogRecord
```

If the pragma `no_assign_warning` is used instead of the pragma `no_assign` for the function block *"TestFB "*, then the `C0328` message is issued as compiler warning, not as a compiler error.

Attribute 'no_check'

This pragma prevents the check function being called for the POU (POUs for implicit checks). Since the check functions can affect the processing speed of the program, it can be useful to apply the attribute to function blocks that have already been checked or are frequently called.

You add the pragma to the declaration of a POU.

Syntax:

```
{attribute 'no_check'}
```

Insertion position: first line in the declaration part of the POU.

NOTICE!

The attribute also automatically affects the child objects of a POU!

Example: If the attribute is entered in a program, check functions will also not be carried out for actions that are assigned to this program.

Attribute 'no_copy'

In general an online change requires a re-allocation of instances, for example of a POU. In the process, the value of the variable contained in the instance is copied.

The pragma prevents the value of the variable contained in the instance from being copied in the course of an online change; instead, the variable is re-initialized in the course of an online change. This can be useful for a local pointer variable that points to a variable that has just been shifted by the online change and thus has a changed address.

You insert the attribute in the declaration part above the line of the declaration of the variables concerned.

Syntax:

```
{attribute 'no_copy'}
```

Attribute 'no-exit'

This attribute suppresses the call of the FB_exit method of a function block for a certain one of its instances. To do this you insert the attribute in the line before the declaration of the function block instance.

Syntax:

```
{attribute 'no-exit'}
```

Example

The method *“FB_exit”* is added to the function block *“POU_ex”*. Two instances of the function block *“POU_ex”* are created in the main program *“PLC_PRG”*.

```
PROGRAM PLC_PRG
VAR
 POU1 : POU_ex;
 {attribute 'no-exit'}
 POU2 : POU_ex;
END_VAR
```

POU1 is called, POU2 is not called.

See also

- *Chapter 1.4.1.19.10 “Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'” on page 748*

Attribute 'noinit'

This pragma is applied to variables that should not be implicitly initialized.

Syntax:

```
{attribute 'no_init'}
```

```
{attribute 'no-init'}
```

```
{attribute 'noinit'}
```

Insertion position: line above the declaration line of the variables concerned in the declaration part.

Example

```
PROGRAM PLC_PRG
VAR
 A : INT;

 {attribute 'no_init'}
 B : INT;
END_VAR
```

When the associated application is reset, the integer variable `A` is implicitly re-initialized with `0`, whereas the variable `B` retains its current value.

Attribute 'no_instance_in_retain'

You can use this pragma to prevent the instance of a function block from being stored in the retain memory.

Syntax:

```
{attribute 'no_instance_in_retain'}
```

Insert location:

Lines above the `FUNCTION_BLOCK` declaration in the declaration part of the function block.

Now when you declare an instance declaration of the function block as a `RETAIN` variable, an error message is issued.

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*

Attribute 'no_virtual_actions'

The pragma is used for function blocks that are derived from a function block implemented in SFC and use the fundamental SFC sequence of this base class. The actions called from it exhibit the same virtual behavior as methods. This means that the implementations of the actions in the base class can be replaced by the derived class with its own specific implementations.

If you apply the pragma to the base class, then its actions are protected against overloading.

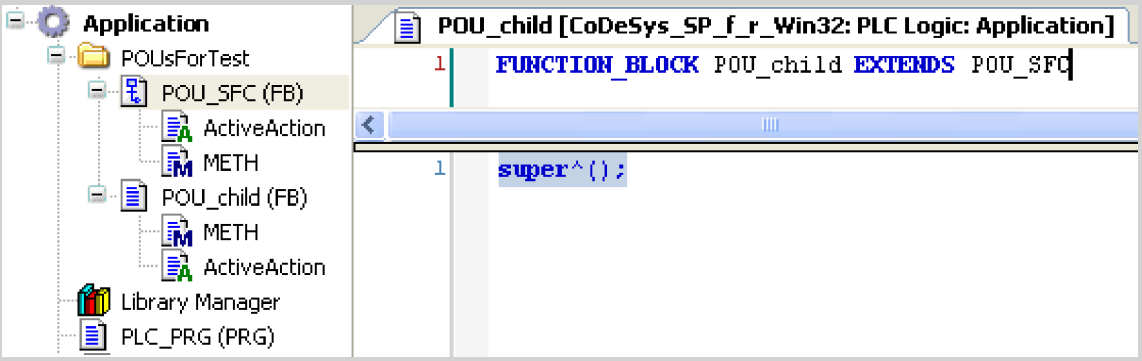
Syntax:

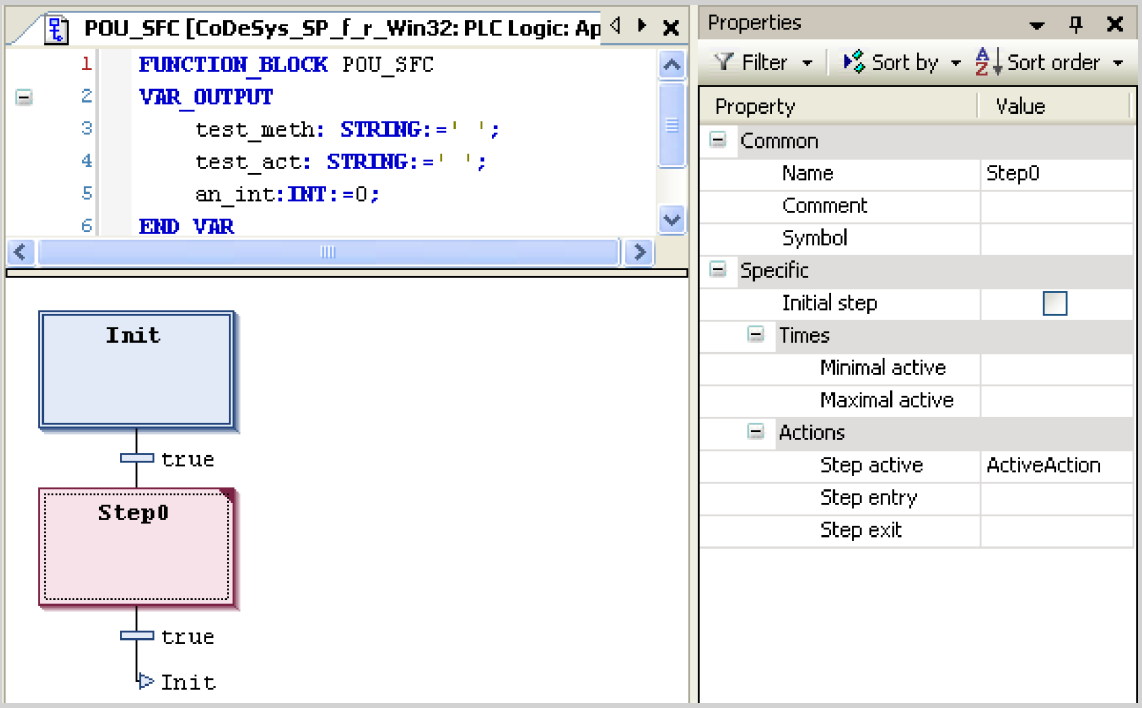
```
{attribute 'no_virtual_actions'}
```

Insert location: Top line in the declaration part of the function block

Example

The function block `POU_SFC` is the base class for the derived function block `POU_child`. The derived class `POU_child` calls the sequence of the base class written in SFC with the special variable `SUPER`.



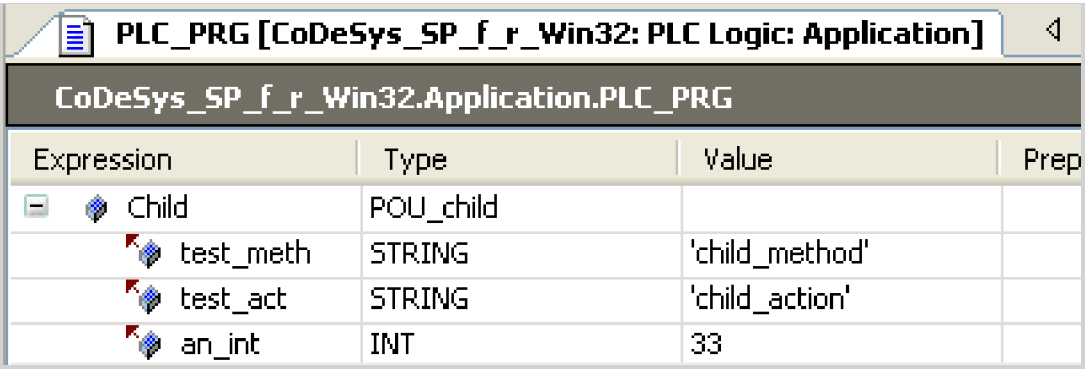


The exemplary implementation of this sequence is limited to the initial step, followed by a single step with a linked step action `ActiveAction`. This step with a linked step action takes care of the configuration of the output variables.

```
an_int:=an_int+1;     // Counting the action calls
test_act:='father_action';
METH();               // Call of the method METH in order to set the
string variable test_meth
```

In the case of the derived class `POU_child` the step action is replaced by a special implementation of `ActiveAction`. `Active Action` differs from the original only by the assignment of the string `'child_action'` in place of `'father_action'` at the variable `test_act`.

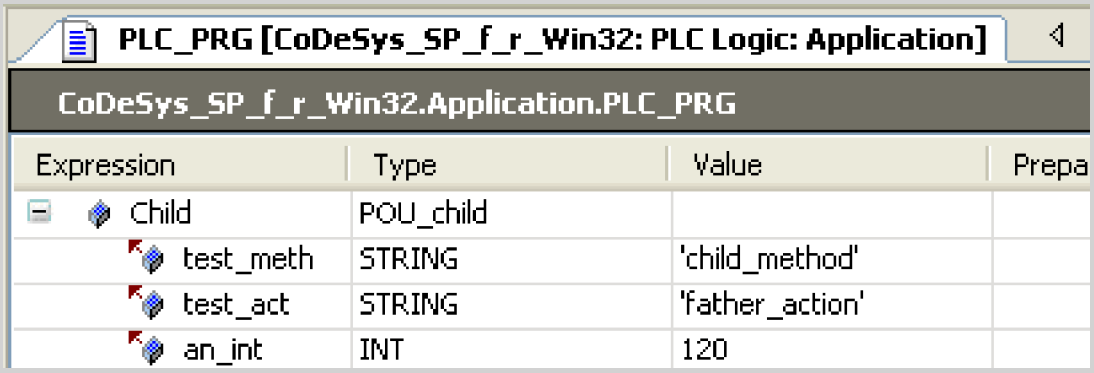
Likewise, the method `METH`, which assigns the string `'father_method'` to the variable `test_meth` in the base class, is overwritten so that `test_meth` now gets the value `'child_method'`. The main program `PLC_PRG` calls an instance of the function block `POU_child`, named `Child`. As expected, the value of the strings reflects the call of the action and method of the derived class:



Now, however, you place the pragma `{attribute 'no_virtual_actions'}` in front of the base class:

```
{attribute 'no_virtual_actions'}

FUNCTION_BLOCK POU_SFC...
```

This changes the behavior: While the implementation of the derived class is still used for the method `METH`, the call of the step action now results in a call of the action `ActiveAction` of the base class. Therefore `test_act` is now given the value `'father_action'`:



Attribute 'pingroup'

The effect of this pragma is that the input pins or output pins (parameters) are grouped in the declaration of a function block. In the FBD/LD editor a pin group defined in this way can be displayed as an enlarged or reduced unit on the inserted function block. Several groups are possible and are distinguished by their names. CODESYS saves the respective state (reduced) per function block box with the project options.

Syntax:

```
{attribute 'pingroup' := '<group name>'}
```

Insertion position: line above the declaration of the input or output variables concerned in the declaration part of a function block.

Example Two groups are defined: `general (i1, out1)` and `group1 (i2, g1). r1, r2, outRes1` and `g2` are always displayed

```
FUNCTION_BLOCK FB
VAR_INPUT
 r1 : REAL;
 {attribute 'pingroup' := 'general'}
 i1 : INT;
 {attribute 'pingroup' := 'group1'}
 i2 : INT;
 r2 : REAL;
END_VAR

VAR_OUTPUT
 outRes1 : REAL;
 {attribute 'pingroup' := 'general'}
 out1 : INT;
    {attribute 'pingroup' := 'group1'}
 g1 : INT;
 g2 : REAL;
END_VAR
```

Attribute 'pin_presentation_order_inputs/outputs'

The pragmas are evaluated in the CFC, FBD, and LD graphical editors, causing the order of inputs/outputs of the affected function block to be displayed as specified. You program the order by assigning the names of the inputs/outputs to the attribute in the desired order.

Syntax

```
{attribute 'pin_presentation_order_inputs' := '<First_Input_Name>,
(<Next_Input_Name>, )* ( *, )? (<Next_Input_Name>,)*
<Last_Input_Name>'}
{attribute 'pin_presentation_order_outputs' := '<First_Output_Name>,
(<Next_Output_Name>,)* ( *, )? (<Next_Output_Name>,)*
<Last_Output_Name>'}
```

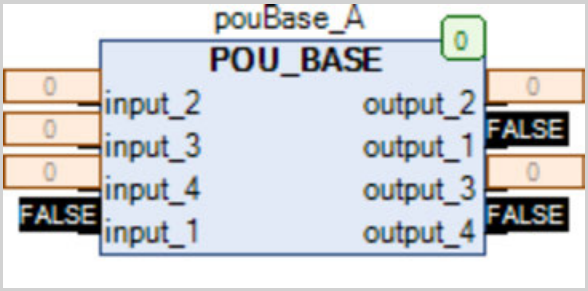
- `*`
  The terminal character serves as a wildcard for all inputs/outputs that are not specified in the display order. If the terminal character is missing, then the missing inputs/outputs are appended at the end.
- `( ... )?`
  The contents of the parentheses are optional.
- `( ... )*`
  The contents of the parentheses are optional again and can therefore occur not at all, one time, or several times.
- Insert location: First line in the declaration part of a function block.

> **NOTICE!**
> This pragma is not evaluated when pragma `{attribute 'pingroup' := '<Group_Name>'}` is used.

Example

```
{attribute 'pin_presentation_order_inputs' := 'input_2,*,input_1'}
{attribute 'pin_presentation_order_outputs' := 'output_2, output_1'}
FUNCTION_BLOCK POU_BASE
VAR_INPUT
    input_1 : BOOL;
    input_2 : INT;
    input_3 : INT;
    input_4 : INT;
END_VAR

VAR_OUTPUT
    output_1 : BOOL;
    output_2 : INT;
    output_3 : INT;
    output_4 : BOOL;
END_VAR

FUNCTION_BLOCK PLC_PRG
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    pouBase_A: POU_BASE;
END_VAR
```

In the representation of function module instance `pouBase_A`, the pragmas result in the following arrangement of input and output pins:



See also

- *Chapter 1.4.1.19.6.2.33 “Attribute 'pingroup'” on page 716*

Attribute 'obsolete'

The effect of this pragma is that a defined warning is displayed for a data type definition during compilation if the data type (structure, function block, etc.) is used in the project. This enables you, for example, to draw attention to the fact that a data type is no longer valid because, for example, an interface has changed and this should also be implemented in the project.

In contrast to a message pragma this warning is defined centrally for all instances of a data type.

Syntax:

```
{attribute 'obsolete' := 'user defined text'}
```

Insertion position: line of the data type definition or in a line above it.

Example The pragma is inserted in the definition function block `fb1`:

```
{attribute 'obsolete' := 'datatype fb1 not valid!'}
FUNCTION_BLOCK fb1
VAR_INPUT
 i:INT;
END_VAR
```

If you use fb1 as a data type, for example in `fbinst:fb1`, the following warning will be displayed when compiling the project: "datatype fb1 not valid".

See also

- *Chapter 1.4.1.19.6.1 “Message Pragmas” on page 683*

Attribute 'pack_mode'

The pragma defines how a data structure is packed during the allocation. The attribute has to be inserted above the data structure and affects the packing of the entire structure.

Syntax:

```
{attribute 'pack_mode' := ' <pack mode value>' }
```

Insert location: above the declaration of the data structure

Table 45: Possible values for <value>:

<pack mode value>	Associated packing method	Description
0	Aligned	All variables are allocated to byte addresses. There are no memory gaps.
1	1-byte-aligned	
2	2-byte-aligned	There are ● 1-byte variables at byte addresses ● 2-byte variables at addresses divisible by 2. A maximum gap of 1 byte results. ● 4-byte variables at addresses divisible by 2. A maximum gap of 1 byte results. ● 8-byte variables at addresses divisible by 2. A maximum gap of 1 byte results. ● Strings always at byte addresses. No gaps result.

<pack mode value>	Associated packing method	Description
4	4-byte-aligned	There are ● 1-byte variables at byte addresses ● 2-byte variables at addresses divisible by 2. A maximum gap of 1 byte results. ● 4 byte variables at addresses divisible by 4. A maximum gap of 3 byte results. ● 8-byte variables at addresses divisible by 4. A maximum gap of 3 byte results. ● Strings always at byte addresses. No gaps result.
8	8-byte-aligned	There are ● 1-byte variables at byte addresses ● 2-byte variables at addresses divisible by 2. A maximum gap of 1 byte results. ● 4 byte variables at addresses divisible by 4. A maximum gap of 3 byte results. ● 8 byte variables at addresses divisible by 8. A maximum gap of 7 byte results. ● Strings always at byte addresses. No gaps result.

Depending on the structure, there may be no difference in the memory mapping of the individual modes. Therefore, the memory allocation of a structure with `<pack mode value> = 4` *can correspond to that of* `<pack mode value> = 8`.

Arrays of structures: If the structures are combined in arrays, then bytes are added at the end of the structure so that the next structure is aligned.

NOTICE!
If the *"Compatibility layout"* option is selected in the symbol configuration and at the same time the attribute 'pack_mode' is used in the code, then problems can occur due to unintentional memory misalignment.

See also

- *Chapter 1.4.1.20.2.25 "Object 'Symbol Configuration'" on page 927*

Example 1

Example

```
{attribute 'pack_mode' := '1'}

TYPE myStruct:
STRUCT
  Enable: BOOL;
  Counter: INT;
  MaxSize: BOOL;
  MaxSizeReached: BOOL;
  END_STRUCT
END_TYPE
```

The memory range for a variable of the data type `myStruct` is allocated 'aligned'. If the storage address of its component `Enable` is `0x0100`, for example, then the component `Counter` follows at the address `0x0101`, `MaxSize` at address `0x0103` and `MaxSizeReached` at address `0x0104`. In the case of `'pack_mode':=2`, `Counter` would be at `0x0102`, `MaxSize` at `0x0104` and `MaxSizeReached` at `0x0106`.

Example 2

Example

```
STRUCT
  Var1 : BOOL  := 16#01;
  Var2 : BYTE  := 16#11;
  Var3 : WORD  := 16#22;
  Var4 : BYTE  := 16#44;
  Var5 : DWORD := 16#88776655;
  Var6 : BYTE  := 16#99;
  Var7 : BYTE  := 16#AA;
  Var8 : DWORD := 16#AA;
END_TYPE
```

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	Variable	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**
0	Var1	01	Var1	01	Var1	01	Var1	01	Var1	01
1	Var2	11	Var2	11	Var2	11	Var2	11	Var2	11
2	Var3	22	Var3	22	Var3	22	Var3	22	Var3	22
3	...	00	...	00	...	00	...	00	...	00
4	Var4	44	Var4	44	Var4	44	Var4	44	Var4	44
5	Var5	55	Var5	55						
6	...	66	...	66	Var5	55				
7	...	77	...	77	...	66				
8	...	88	...	88	...	77	Var5	55	Var5	55
9	Var6	99	Var6	99	...	88	...	66	...	66
10	Var7	AA	Var7	AA	Var6	99	...	77	...	77
11	Var8	AA	Var8	AA	Var7	AA	...	88	...	88
12	...	00	...	00	Var8	AA	Var6	99	Var6	99
13	...	00	...	00	...	00	Var7	AA	Var7	AA
14	...	00	...	00	...	00				
15					...	00				
16							Var8	AA	Var8	AA
17							...	00	...	00
18							...	00	...	00
19							...	00	...	00
20										
21										
22										
23										
24										
25										
26										
27										

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	Variable	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**
28										
29										
30										
31										

Example 3

Example

```
STRUCT
  Var1 : BYTE  := 16#01;
  Var2 : LWORD := 16#11;
  Var3 : BYTE  := 16#22;
  Var4 : BYTE  := 16#44;
  Var5 : DWORD := 16#88776655;
  Var6 : BYTE  := 16#99;
  Var7 : BYTE  := 16#AA;
  Var8 : WORD  := 16#AA;
END_TYPE
```

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	Variable	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**
0	Var1	01	Var1	01	Var1	01	Var1	01	Var1	01
1	Var2	11	Var2	11						
2	...	00	...	00	Var2	11				
3	...	00	...	00	...	00				
4	...	00	...	00	...	00	Var2	11		
5	...	00	...	00	...	00	...	00		
6	...	00	...	00	...	00	...	00		
7	...	00	...	00	...	00	...	00		
8	...	00	...	00	...	00	...	00	Var2	11
9	Var3	22	Var3	22	...	00	...	00	...	00
10	Var4	44	Var4	44	Var3	22	...	00	...	00
11	Var5	55	Var5	55	Var4	44	...	00	...	00
12	...	66	...	66	Var5	55	Var3	22	...	00
13	...	77	...	77	...	66	Var4	44	...	00
14	...	88	...	88	...	77			...	00
15	Var6	99	Var6	99	...	88			...	00
16	Var7	AA	Var7	AA	Var6	99	Var5	55	Var3	22
17	Var8	AA	Var8	AA	Var7	AA	...	66	Var4	44
18	...	00	...	00	Var8	AA	...	77		
19					...	00	...	88		
20							Var6	99	Var5	55
21							Var7	AA	...	66
22							Var8	AA	...	77
23							...	00	...	88
24									Var6	99
25									Var7	AA
26									Var8	AA
27									...	00

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	Variable	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**	**Variable**	**Value**
28										
29										
30										
31										

Behavior without pack mode

If pack mode is not used, then the compiler typically uses pack mode 4 or 8, depending on the device description. In each case, a pack mode which is particularly beneficial for the processor is used so that memory access can be performed. This is also called natural alignment or a natural alignment of data.

Negative effects when using pack mode

Unaligned memory access can be the result of using the attribute `'pack_mode'`. This means, for example, that a data type with a size of 4 bytes is then located at an address which is not divisible by 4. Normally, on a 32-bit system a 32-bit data type can be read and written with a single memory access. On some platforms, for example on ARM platforms, this is possible only when this value is aligned in the memory. On other platforms, it can be that the access is possible but it is performed much more slowly.

Example

```
{attribute 'pack_mode':=1}

TYPE DUT
STRUCT
by1 : BYTE;
dw1 : DWORD;
END_STRUCT
END_TYPE
```

On an ARM platform, the value `dw1` cannot be read with a single access. When an attempt is made to access this element directly, the ARM processor will throw an exception.

Assumption: The following read access is performed: `dwTest := dut1.dw1;`

For this access to the `DWORD dw1`, four memory accesses are required because each byte is read, shifted, and disjuncted individually. The flow is somewhat the same as in the following example in which a `DWORD` is generated from an array of four bytes:

```
dwHelp := bytes[0];
dwResult := dwHelp;
dwHelp := bytes[1];
dwHelp := SHL(dwHelp, 8);
dwResult := dwResult OR dwHelp;
dwHelp := bytes[2];
dwHelp := SHL(dwHelp, 16);
dwResult := dwResult OR dwHelp;
dwHelp := bytes[3];
dwHelp := SHL(dwHelp, 24);
dwResult := dwResult OR dwHelp;
```

Obviously, this kind of access is much slower than access to a `DWORD`, which is aligned appropriately in the memory.

```
pdw := ADR(dut1.dw1);
dwTest := pdw^;
```

However, the compiler will not generate the access of the example when this kind of member is accessed by means of a pointer. This means that the following code results in an exception on an ARM platform.

```
pdw := ADR(dut1.dw1);
dwTest := pdw^;
```

For performance reasons, you should therefore avoid working with structures which are not naturally aligned.

A packed structure must not contain an unpacked structure.

Attribute 'ProcessValue'

With the `'ProcessValue'` attribute, you mark a component of a structure. In the CFC editor, you can then use the command *"Use attributed member as input"* in order to connect this structure to an input of scalar type.

Syntax:

```
{attribute 'ProcessValue'}
```

Insert location: Line above the affected structure variable.

Example

```
TYPE QINT :
STRUCT
    Status : STRING;
    {attribute 'ProcessValue'}
    Value1 : INT;
    Value2 : INT;
END_STRUCT
END_TYPE
```

See also

- *Chapter 1.4.1.20.3.12.36 "Command 'Use Attributed Member as Input'" on page 1102*

Attribute 'qualified_only'

The effect of this pragma is that variables of a global variable list are only addressed by specifying the global variable name, for example `gvl.g_var`. This also applies to variables of the type Enumeration and can be helpful in avoiding being mistaken for local variables.

Syntax:

{attribute 'qualified_only'}

Insertion position: line above `VAR_GLOBAL` in a GVL

Example Global Variable List *"GVL"*:

```
{attribute 'qualified_only'}
VAR_GLOBAL
 iVar:INT;
END_VAR
```

Within a POU, for example *"PLC_PRG"*, the global variable `iVar` can only be addressed using the prefix `GVL`:

```
GVL.iVar:=5;
```

Conversely, the following incomplete call of the variable will create an error:

```
iVar:=5;
```

Attribute 'reflection'

The pragma is used to identify POUs in which some variables require special treatment and are tagged with a specific attribute for this purpose. Currently, this applies to the attributes `'instance-path'` and `'is-connected'` for function block variables. The compiler searches only blocks marked with `'reflection'` for variables with these attributes and therefore needs less time.

Syntax:

```
{attribute 'reflection'}
```

For examples, see the description of the attributes `'instance-path'` and **'is-connected'**.

See also

- *Chapter 1.4.1.19.6.2.21 "Attribute 'instance-path'" on page 706*
- *Chapter 1.4.1.19.6.2.23 "Attribute 'is_connected'" on page 707*

Attribute 'subsequent'

The pragma is used to allocate consecutive variables in memory. When the list changes, the entire variable list is allocated to a new memory area. This pragma is used in programs and global variable lists.

Syntax:

```
{attribute 'subsequent'}
```

NOTICE!
`VAR_TEMP` in a program with attribute `'subsequent'` leads to a compiler error.

When a variable in the list is qualified with `RETAIN`, all variables of the declaration part are stored in the memory area for `RETAIN`.

Attribute 'symbol'

The pragma `{attribute 'symbol'}` defines which variables of a program or a global variable list are to be adopted into the symbol configuration. This means that the variables are exported as symbols to a symbol list. This symbol list is then available for external access both as an XML file in the project directory and as a file that is invisible to the user on the target system. For example, the symbol list is then available for access by an OPC server. The variables thus equipped with a symbol are loaded by CODESYS to the controller, even if they are not explicitly configured or visible in the editor of the symbol configuration.

In any case, however, an object *"Symbol configuration"* must be created below the application concerned in the device tree.

Syntax:

```
{attribute 'symbol' := '<access possibilities>'}
```

`<access possibilities>`: `none`, `read`, `write`, `readwrite`. The default value `readwrite` applies if no parameter is specified.

Insertion position:

- in order to affect only an individual variable, you must place the pragma in the line before the variable declaration.
- In order to be effective for all variables in the declaration part of a program, you must place the pragma in the first line of the declaration editor. In this case, too, you can still set instructions for individual variables explicitly in the respective line.

Example

With the following configuration the variables `A` and `B` are exported with read and write permission. Variable `D` is exported with read permission.

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PLC_PRG
VAR
 A : INT;
 B : INT;
{attribute 'symbol' := 'none'}
 C : INT;
{attribute 'symbol' := 'read'}
 D : INT;
END_VAR
```

See also

- ↳ *Chapter 1.4.1.8.6 "Using Pragmas" on page 263*
- ↳ *Chapter 1.4.1.9.2 "Symbol Configuration" on page 357*

Attribute 'to_string'

The pragma affects how the result of converting an enumeration component with the `TO_STRING` operator is output. If the enumeration declaration has the pragma, then the name of the enumeration component appears as a string instead of the numeric value.

Syntax:

```
{attribute 'to_string'}
```

Insert location: First line above the declaration part of the enumeration.

Example

Declaration of the enumeration `color`:

```
{attribute 'to_string'}
TYPE color :
(
    red := 0,
    blue := 1,
    green := 2
);
END_TYPE
```

Conversion with `TO_STRING`:

```
PROGRAM PLC_PRG
VAR
    i_color: Color;
    s_show_color: STRING;
END_VAR
i_color := 1;
s_show_color := TO_STRING(i_color);
```

In this case, `str_show_color` gets the value `'blue'` instead of `'1'` as the conversion result.

See also

- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Attribute 'warning disable', attribute 'warning restore'

This pragma causes certain warnings to be suppressed. The `warning restore` pragma causes a suppressed message to be reactivated.

Syntax:

```
{warning disable <compiler ID>}
```

```
{warning restore <compiler ID>}
```

`<compiler ID>`: ID located at the beginning of an error or a warning message.

Example

Compiler message:

```
typify code ...
C0196: Implicit conversion from unsigned Type 'UINT' to signed Type
'INT' : possible change of sign
Compile complete -- 0 errors
```

Applying the pragma to a variable declaration:

```
VAR
 {warning disable C0195}
 test1 : UINT := -1;
 {warning restore C0195}
 test2 : UINT := -1;
END_VAR
```

`test1` does not generate an error message, `test2` generates an error message.

Effects of Pragmas on Symbols

POUs and variables can change their behavior with respect to the symbol configuration as a result of pragmas. A detailed description can be found on the help page of each pragma.

Pragma with attribute	Effect	See also
`{attribute 'call_after_global_init_slot' := ' <slot> '}`	None	
`{attribute 'call_after_init'}`	None	
`{attribute 'call_after_online_change_slot' := ' <slot> '}`	None	
`{attribute 'call_before_global_exit_slot' := ' <slot> '`	None	
`{attribute 'call_on_type_change':= ' comma separated list of referenced function blocks> '}`	None	

`{attribute 'conditionalshow' := ' <some text> '}`
`{attribute 'conditionalshow'}`
`{attribute 'conditionalshow_all_locals' := ' <some text> '}`
`{attribute 'conditionalshow_all_locals'}`
	The marked variables are hidden and therefore cannot be exported.
	However, if the source code file from the compiled library is available, or if CODESYS has been started with the command-line option `conditionalshowsymbols`, then the marked variables are visible despite the pragma.
		Chapter 1.4.1.19.6.2.7 “Attribute 'conditional-show'” on page 690
		Chapter 1.4.1.19.6.2.8 “Attribute 'conditional-show_all_locals'” on page 691

`{attribute 'const_replaced'}` `{attribute 'const_non_replaced'}`	Replaced constants are not available in the symbol configuration editor and therefore cannot be exported. A constant being replaced depends on whether or not the *“Replace constants”* compiler option has been selected for all constants and whether or not pragmas overwrite the compiler option for individual constants.	*Chapter 1.4.1.19.6.2.9 “Attribute 'const_replaced', Attribute 'const_non_replaced'” on page 692*
`{attribute 'dataflow'}`	None	
`{attribute 'displaymode':= <displaymode> }`	None	
`{attribute 'enable_dynamic_creation'}`	None	
`{attribute 'estimated-stack-usage' := ' <estimated stack size in bytes> '}`	None	
`{attribute 'ExpandFully'}`	None	
`{attribute 'global_init_slot' := '<slot>'}`	None	

Pragma with attribute	Effect	See also
`{attribute 'hide'}`	Variables are hidden and therefore cannot be exported.	*Chapter 1.4.1.19.6.2.16 "Attribute 'hide'" on page 700*
`{attribute 'hide_all_locals'}`	Variables are hidden and therefore cannot be exported.	*Chapter 1.4.1.19.6.2.17 "Attribute 'hide_all_locals'" on page 703*
`{attribute 'initialize_on_call'}`	None	
`{attribute 'init_namespace'}`	None	
`{attribute 'init_on_onlchange' }`	None	
`{attribute 'instance-path'}`	None	
`{attribute 'io_function_block'}` `{attribute 'io_function_block_mapping'}`	None	
`{attribute 'is_connected' := '<input variable> '}`	None	
`{attribute 'linkalways'}`	POUs and library POUs are integrated in the compile list and therefore cannot be exported.	*Chapter 1.4.1.19.6.2.24 "Attribute 'linkal-ways'" on page 708*
`{attribute 'monitoring' := 'variable'}` `{attribute 'monitoring' := 'call'}`	Properties `PROPERTY` or functions (`FUNCTION`) are provided as symbols.	*Chapter 1.4.1.19.6.2.25 "Attribute 'monitor-ing'" on page 709*
`{'no_assign' }` `{'no_assign_warning' }`	None	
`{attribute 'no_check'}`	None	
`{attribute 'no_copy'}`	None	
`{attribute 'no-exit'}`	None	

`{attribute 'no_init'}`
`{attribute 'no-init'}`
`{attribute 'noinit'}`
	None	

`{attribute 'no_instance_in_retain'}`	None	
`{attribute 'no_virtual_actions'}`	None	

Pragma with attribute	Effect	See also
`{attribute 'pingroup' := ' <group name> '}`	None	
`{attribute 'pin_presentation_order_inputs' := '< input name > '( , <next input name> )* }` `{attribute 'pin_presentation_order_outputs' := '< output name > '( , <next output name> )* }`	None	
`{attribute 'obsolete' := 'user defined text'}`	None	
`{attribute 'pack_mode' := ' <pack mode value> '}`	Can lead to intentional memory misalignment	*Chapter 1.4.1.20.2.25 “Object 'Symbol Configuration'” on page 927*
`{attribute 'ProcessValue'}`	None	
`{attribute 'qualified_only'}`	None	
`{attribute 'reflection'}`	None	
`{attribute 'subsequent'}`	None	
`{attribute 'symbol' := '<access possibilities>'}`	Variable is exported as symbol. The variable is displayed in the symbol list only when the *“View”*, *“Symbols Exported via Attribute”* option is selected in the symbol configuration editor. The access rights, which have been defined with the pragma, are displayed In the *“Attribute”* column.	*Chapter 1.4.1.19.6.2.41 “Attribute 'symbol'” on page 728*
`{attribute 'to_string'}`	None	
`{warning disable <compiler ID> }` `{warning restore <compiler ID> }`	None	

See also

- *Chapter 1.4.1.9.2 “Symbol Configuration” on page 357*
- *Chapter 1.4.1.20.2.25 “Object 'Symbol Configuration'” on page 927*

Conditional Pragmas

The purpose of conditional pragmas is to influence the generation of code in the pre-compilation process or the compilation process. The ST implementation language supports these pragmas.

NOTICE!
They use conditional pragmas in the implementations of POUs. CODESYS does not evaluate these conditional pragmas if you use them in the declaration part.

With conditional pragmas you affect whether implementation code is taken into account for the compilation. For example, you can make this dependent on whether a certain variable is declared, whether a certain function block exists, etc.

Pragma	Description
`{define <identifier>}`	The value can be queried and compared later with `hasvalue`.
`{undefine <identifier>}`	The `{define}` statement of the identifier `<identifier>` is canceled, and the identifier is 'undefined' again from now on. The pragma is ignored if the specified identifier is not defined at all.

`{IF <expr>}...`
`{ELSIF <expr>}...`
`{ELSE}...`
`END_IF}`
	These are pragmas for the conditional compilation.
	The specified expressions `<expr>` must be constant at the time of compilation; they are evaluated in the order in which they appear here until one of the expressions indicates a non-zero value. The text linked to the instruction is compiled; the other lines are ignored. The order of the sections is fixed. The `ELSIF` and `ELSE` sections are optional. The `ELSIF`-segments may occur any number of times. You can use several conditional compilation operators within the constants `<expr>`.

`<expr>`	You can use one or more **operators** within the constant expression `<expr>` within the conditional compilation pragma `{IF}` or `{ELSIF}` .

You can enter expressions and `define` definitions as “compiler definitions” in the “Compile” tab in the Properties dialog of POUs. If you enter `define` definitions in the properties dialog, you must omit the term `{define}`, contrary to the definition in the implementation code. In addition, you can specify several `define` definitions in the properties dialog, separated by commas.

See also

- *Chapter 1.4.1.8.3.3 “Structured Text (ST), Extended Structured Text (ExST)” on page 253*

Operator `defined (<identifier>)`

This operator causes the expression to be given the value `TRUE`. The requirement is that the identifier `<identifier>` was defined with the help of a `{define}` instruction and not undefined again afterwards with an `{undefine}` instruction; otherwise `FALSE` is returned.

Example

Requirement: The applications `App1` and `App2` exist. The variable `pdef1` is defined by a `{define}` statement in `App1`, but not in `App2`.

```
{IF defined (pdef1)}
(* This code is processed in App1 *)
{info 'pdef1 defined'}
    hugo := hugo + SINT#1;
{ELSE}
(* the following code is only processed in App2 *)
{info 'pdef1 not defined'}
    hugo := hugo - SINT#1;
{END_IF}
```

This also contains an example of a message pragma: Only the message `pdef1 defined` is displayed in the message view when the application is compiled, because `pdef1` is actually defined. The message `pdef1 not defined` is displayed if `pdef1` is not defined.

Operator `defined (variable: <variable>)`

This operator causes the expression to be given the value `TRUE` if the variable `<variable>` is declared within the current scope; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. The variable `g_bTest` is declared in `App1`, but not in `App2`.

```
{IF defined (variable: g_bTest)}
(* the following code is only processed in App2*)
    g_bTest := x > 300;
{END_IF}
```

Operator `defined (type: <identifier>)`

The operator causes the expression to be given the value `TRUE` if a data type is declared with the identifier `<identifier>`; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. The data type `DUT` is declared in `App1`, but not in `App2`.

```
{IF defined (type: DUT)}
(* the following code is only processed in App1*)
    bDutDefined := TRUE;
{END_IF}
```

Operator `defined (pou: <pou name>)`

The operator causes the expression to be given the value `TRUE` if a function block or an action with name `<pou-name>` exists; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. The function block `CheckBounds` exists in `App1`, but not in `App2`.

```
{IF defined (pou: CheckBounds)}
(* the following code is only processed in App1 *)
    arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] +
1;
{ELSE}
(* the following code is only processed in App2 *)
    arrTest[i] := arrTest[i]+1;
{END_IF}
```

Operator `defined (task: <identifier>)`

Not yet implemented!

The operator causes the expression to be given the value `TRUE` if a task is defined with the name `<identifier>`; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. The task `PLC_PRG_Task` is defined in `App1`, but not in `App2`.

```
IF defined (task: PLC_PRG_Task)}
(* the following code is only processed in App1 *)
    erg := plc_prg.x;
{ELSE}
(* the following code is only processed in App2 *)
    erg := prog.x;
{END_IF}
```

Operator `defined (resource: <identifier>)`

Not yet implemented!

The operator causes the expression to be given the value `TRUE` if a resource object with the name `<identifier>` exists for the application; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. A resource object `glob_var1` of the global variable list exists for `App1`, but not for `App2`.

```
{IF defined (resource:glob_var1)}
(* the following code is only processed in App1 *)
    gvar_x := gvar_x + ivar;
{ELSE}
(* the following code is only processed in App2 *)
    x := x + ivar;
{END_IF}
```

Operator `defined (IsSimulationMode)`

The operator causes the expression to be given the value `TRUE` if the application runs on a simulated device, i.e. in simulation mode.

See also

- *Chapter 1.4.1.11.1 “Testing in simulation mode” on page 394*

Operator defined (IsLittleEndian)

The operator causes the expression to be given the value `FALSE`, if the CPU memory is organized in Big Endian (Motorola byte order).

Operator defined (IsFPUSupported)

If the expression returns the value `TRUE`, then the code generator produces an FPU code (for the floating-point unit processor) when calculating with `REAL` values. Otherwise CODESYS emulates FPU operations, which is much slower.

Operator hasvalue (RegisterSize, '<register size>')

`<register size>`: Size of a CPU register in bits

This operator causes the expression to return the value `TRUE` if the size of a CPU register is equal to `<register size>`.

Possible values for `<register size>`

- `16` for C16x,
- `64` for X86-64 bit
- `32` for X86-32 Bit

Operator hasvalue (PackMode, '<pack mode value>')

The checked pack mode depends on the device description, not on the pragma that can be specified for individual DUTs.

Operator hasattribute (pou: <pou name>, '<attribute>')

This operator causes the expression to be given the value `TRUE` if the attribute `<attribute>` is specified in the first line of the declaration part of the function block `<pou name>`; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. The function `fun1` is declared in `App1` and `App2`. However, in `App1` it is also provided with the pragma `{attribute 'vision'}`.

In App1:

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
    i : INT;
END_VAR
VAR
END_VAR
```

In App2:

```
FUNCTION fun1 : INT
VAR_INPUT
    i : INT;
END_VAR
VAR
END_VAR
```

Pragma instruction:

```
{IF hasattribute (pou: fun1, 'vision')}
(* the following code is only processed in App1 *)
    ergvar := fun1(ivar);
{END_IF}
```

See also

- ↳ *Chapter 1.4.1.19.6.2.1 “User-defined attributes” on page 686*

Operator `hasattribute (variable: <variable>, '<attribute>')`

This operator causes the expression to be given the value `TRUE` if the pragma `{attribute '<attribute>'}` is assigned to the variable in the line before the variable declaration; otherwise `FALSE` is returned.

Example

Requirement: The two applications `App1` and `App2` exist. The variable `g_globalInt` is used in `App1` and `App2`, but in `App1` the attribute `'DoCount'` is assigned to it in addition.

Declaration of `g_GlobalInt` in `App1`

```
VAR_GLOBAL
    {attribute 'DoCount'}
    g_globalInt : INT;
    g_multiType : STRING;
END_VAR
```

Declaration `g_GlobalInt` in `App2`:

```
VAR_GLOBAL
    g_globalInt : INT;
    g_multiType : STRING;
END_VAR
```

Pragma instruction:

```
{IF hasattribute (variable: g_globalInt, 'DoCount')}
    (* the following code is only processed in App1 *)
    g_globalInt := g_globalInt + 1;
{END_IF}
```

See also

- ↳ *Chapter 1.4.1.19.6.2.1 “User-defined attributes” on page 686*

Operator `hastype (variable: <variable>, <type-spec>)`

This operator causes the expression to be given the value `TRUE` if the variable `<variable>` is of the data type `<type-spec>`; otherwise `FALSE` is returned.

Possible data types for `<type-spec>`:

- `BOOL`
- `BYTE`
- `DATE`
- `DATE_AND_TIME (DT)`
- `DINT`
- `DWORD`
- `INT`
- `LDATE`
- `LDATE_AND_TIME (LDT)`
- `LINT`
- `LREAL`
- `LTIME`
- `LTIME_OF_DAY (LTOD)`
- `LWORD`
- `REAL`
- `SINT`
- `STRING`
- `TIME`
- `TIME_OF_DAY (TOD)`
- `ULINT`

- UDINT
- UINT
- USINT
- WORD
- WSTRING

Example

Requirement: The two applications App1 and App2 exist. The variable g_multitype is declared in App1 with data type LREAL, in App2 with data type STRING.

```
{IF (hastype (variable: g_multitype, LREAL))}
(*  the following code is only processed in App1 *)
    g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
(* the following code is only processed in App2 *)
    g_multitype := 'this is a multitalent';
{END_IF}
```

Operator hasvalue (<define-ident>, '<char-string>')

This operator causes the expression to be given the value TRUE if a variable is defined with the identifier <define-ident> and has the value <char-string>; otherwise FALSE is returned.

Example

Requirement: The two applications App1 and App2 exist. The variable test is used in the applications App1 and App2; in App1 it is given the value 1, in App2 the value 2.

```
{IF hasvalue(test,'1')}
(*  the following code is only processed in App1 *)
x := x + 1;
{ELSIF hasvalue(test,'2')}
(*  the following code is only processed in App2 *)
    x := x + 2;
{END_IF}
```

Operator hasconstantvalue(<variable>, <literal expression>)

You can use this operator to query the declared value of a constant.

Example

Requirement:

```
{IF hasconstantvalue(test,'1')}
(*  the following code is only processed in App1 *)
    x := x + 1;
{ELSIF hasconstantvalue(test,'2')}
(*  the following code is only processed in App2 *)
    x := x + 2;
{END_IF}
```

Operator NOT <operator>

The expression is given the value TRUE if the reverse value of <operator> returns the value TRUE. <operator> can be one of the operators described in this chapter.

Example

Requirement: The two applications `App1` and `App2` exist.`PLC_PRG1` exists in `App1` and `App2`, and the POU `CheckBounds` exists only in `App1`.

```
{IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))}
(* the following code is only processed in App2 *)
    bANDNotTest := TRUE;
{END_IF}
```

Operator <operator> AND <operator>

The expression is given the value `TRUE` if the two specified operators return `TRUE`. `<operator>` can be one of the operators described in this chapter.

Example

Requirement: The applications `App1` and `App2` exist.`PLC_PRG1` exists in `App1` and `App2`, the POU `CheckBounds` only in `App1`.

```
{IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))}
    (* the following code is only processed in App1 *)
    bANDTest := TRUE;
{END_IF}
```

Operator <operator> OR <operator>

The expression returns `TRUE` if one of the two specified operators returns `TRUE`. `<operator>` can be one of the operators described in this chapter.

Example

Requirement: The two applications `App1` and `App2` exist. The POU `PLC_PRG1` exists in `App1` and `App2`, and the POU `CheckBounds` exists only in `App1`.

```
{IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))}
(* the following code is only processed in App1 and in App2 *)
    bORTest := TRUE;
{END_IF}
```

Operator (<operator>)

`()` parenthesizes the operators.

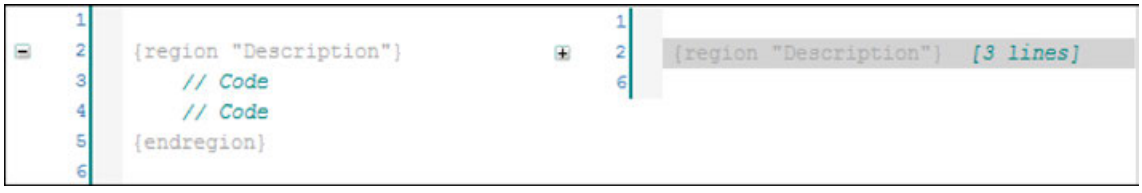
See also

- *Chapter 1.4.1.8.6 “Using Pragmas” on page 263*
- *Chapter 1.4.1.19.6.2.1 “User-defined attributes” on page 686*

Region Pragma

This pragma is used for grouping several lines into one block in a text editor. The block can be named. Region pragmas can also be nested.

Code with region pragma: Expanded and collapsed views



The pragma can be used in the ST editor and all declaration editors. Syntax highlighting can be customized in the options.

See also

- *Chapter 1.4.1.20.3.2.18 “Command 'Collapse All Folds'” on page 971*
- *Chapter 1.4.1.20.3.2.17 “Command 'Expand All Folds'” on page 971*

1.4.1.19.7 Identifiers

Rules for identifier designation

Rules for identifiers of variables

- An identifier must not contain spaces or special characters.
- Capitalization is ignored. For example, `VAR1` and `var1` refer to the same variable.
- The underscore is recognized. For example, `A_BCD` and `AB_CD` are treated as two different identifiers. Multiple consecutive underscores are not permitted.
- The length of an identifier is unrestricted.

Rules for multiple use of identifiers (namespaces)

- An identifier must not be declared two times locally.
- An identifier can be used more than one time globally. If a local variable has the same name as a global variable, then the local variable has priority within the POU.
- An identifier must not be identical to a keyword, such as the scope `VAR_Global`.
- A variable that is declared in a global variable list can have the same name as a variable defined in another GVL. CODESYS provides features that extend the standard for the namespace or scope of variables:
  - Global namespace operator:
    An instance path that begins with a dot always opens a global namespace. If there is a local variable (for example, `ivar`) that has the same name as a global variable, then you refer to the global variable as `.ivar`.
  - The name of a global variable list can define the namespace uniquely for the include variables. Therefore, you can declare variables with the same name in different global variables list and still uniquely reference by prepending the list name.
    For example, `globlist1.ivar := globlist2.ivar; (* ivar from GVL globlist2 is copied to ivar in GVL globlist1 *).`
  - Variables that are defined in the global variable list of a library included in the project can be addressed uniquely according to the following syntax:
    `<name scope library>.< GVL name>.<variable name>`
    For example, `globlist1.ivar := lib1.globlist1.ivar (* ivar from GVL globlist1 in library lib1 is copied to ivar in GVL globlist1 *).`
- When inserting a library, you also use the Library Manager to define a namespace. In this way, you can make unique references to a library block or library variable by `<namespace library>.<block name|variable name>`. Note that when libraries are nested, you have to reference the namespaces of all libraries are in succession
  Example: If `Lib1` is referenced by `Lib0`, then the POU `func` in `Lib1` is addressed by `Lib0.Lib1.fun`: `ivar := Lib0.Lib1.fun(4, 5); (* return value from func is copied to variable ivar in the project *)`

We recommend that you apply the following rules in addition to the items that you have to consider specifically for variables declaration. By doing this, you get the best possible harmonization when assigning names.

Recommendations for variable names

Whenever possible, you should name variables in Hungarian notation in applications and libraries. Find a meaningful, short, English name for each variable as a base name, which can consist of several words. Write the first letter of each word in uppercase, the remaining letters in lowercase. In front of the base name, append a prefix in lowercase to indicate the data type of the variable.

Example: `iFileSize : INT;`

Data Type	Prefix	Description
BOOL	x	We expressly recommend x as the prefix for Boolean variables in order to distinguish them from identifiers of the data type BYTE. The prefix indicates the view of an IEC programmer.
	b	Reserved
BYTE	by	Bit string; not for arithmetic operations
WORD	w	Bit string; not for arithmetic operations
DWORD	dw	Bit string; not for arithmetic operations
LWORD	lw	Bit string; not for arithmetic operations
SINT	si	Arithmetic integer data type, 8-bit
USINT	usi	Arithmetic integer data type, 8-bit
INT	i	Arithmetic integer data type, 16-bit
UINT	ui	Arithmetic integer data type, 16-bit
DINT	di	Arithmetic integer data type, 32-bit
UDINT	udi	Arithmetic integer data type, 32-bit
LINT	li	Arithmetic integer data type, 64-bit
ULINT	uli	Arithmetic integer data type, 64-bit
REAL	r	Arithmetic floating-point data type, 32-bit
LREAL	lr	Arithmetic floating-point data type, 64-bit
STRING	s	Single-byte character string of variable length (default setting: 80 characters)
WSTRING	ws	Double-byte character string of variable length (default setting: 80 characters)
TIME	tim	Time duration, 32-bit
LTIME	ltim	Time duration, 64-bit
• TIME_OF_DAY • TOD	tod	Time of day, 32-bit
• LTIME_OF_DAY • LTOD	ltod	Time of day, 64-bit
• DATE_AND_TIME • DT	dt	Date and time
• LDATE_AND_TIME • LDT	ldt	
DATE	• dat • d	Calender date
LDATE	• ldat • ld	Calender date

Data Type	Prefix	Description
POINTER	p	
ARRAY	a	
Enumeration	e	

Example

```
VAR
    bySubIndix: BYTE;
    xFlag: BOOL;
    udiCounter: UDINT;
END_VAR
```

Identifier	Description	Example
Nested declaration	Prefixes are attached successively in the order of declaration.	pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
Function block instance Variable of user-defined data type	Prefix: Abbreviation for the name of the function block or data type	cansdoReceivedTelegram: CAN_SDOTelegram; TYPE CAN_SDOTelegram : (* prefix: sdo *) STRUCT wIndex: WORD; bySubIndex:BYTE; byLen:BYTE; aby: ARRAY [0..3] OF BYTE; END_STRUCT END_TYPE
Local constant Local constant variable	Prefix: c_, followed by the type prefix and the variable name	VAR CONSTANT c_uiSyncID: UINT := 16#80; END_VAR
Global variable	An additional prefix is appended to the library prefix. g_	VAR_GLOBAL CAN_g_iTest: INT; END_VAR
Global constants Global constant variable	An additional prefix is appended to the library prefix. gc_	VAR_GLOBAL CONSTANT CAN_gc_dwExample: DWORD; END_VAR

Recommendations for variable names CODESYS V3.x libraries

Identifier	Description	Example

Variable	Corresponds to the description for variable names, with the exception that global variables and constants do not require library prefixes because the namespace replaces the function.	`g_iTest: INT; // Declaration`
		`CAN.g_iTest; // Implementation; call in the program`

Recommendations for identifiers for user-defined data types (DUT)

Identifier for	Description	Example

Structures	Library prefix followed by an underscore and a short, informative description of the structure. The associated prefix for created variables of this structure should follow the colon as a comment.	`TYPE CAN_SDOTelegram : (* prefix: sdo *)`
		`STRUCT`
		`wIndex : WORD;`
		`bySubIndex : BYTE;`
		`byLen : BYTE;`
		`abyData: ARRAY [0..3] OF BYTE;`
		`END_STRUCT`
		`END_TYPE`

Enumerations	Library prefix followed by an underscore and the identifier in uppercase.
	Note: In past CODESYS versions, enumeration values > 16#7FFF caused errors because they were not automatically converted to `INT`. For this reason, enumerations should always be declared with correct `INT` values.
		`TYPE CAL_Day :`
		`(`
		`CAL_MONDAY,`
		`CAL_TUESDAY,`
		`CAL_WEDNESDAY,`
		`CAL_THURSDAY,`
		`CAL_FRIDAY,`
		`CAL_SATURDAY,`
		`CAL_SUNDAY`
		`);`
		Declaration:
		`eToday: CAL_Day;`

Recommendations for identifiers for user-defined data types (DUT) in CODESYS V3 libraries

Identifier for	Description	Example

DUT names in CODESYS V3 libraries	The namespace replaces the need for the library prefix. Therefore, it is omitted. Enumeration values are also defined without a library prefix.	Library with namespace `CAL`
		`TYPE DAY :`
		`(`
		`□MONDAY`
		`□TUESDAY,`
		`□WEDNESDAY,`
		`□THURSDAY,`
		`□FRIDAY,`
		`□SATURDAY,`
		`□SUNDAY`
		`);`
		Declaration:
		`eToday: CAL.Day;`
		Usage in the application
		`IF eToday = CAL.Day.MONDAY THEN`

Recommendations for identifiers for POUs, functions, function blocks, programs

Identifier for	Description	Example
POUs: Functions, function blocks, programs	Library prefix followed by an underscore and a short, informative POU name. Like for variables, the first letter of each word is uppercase and all other letters are lowercase. We recommend that you compose the POU name from a verb and a noun. For function blocks, the associated prefix for created instances should follow the name as a comment.	`FUNCTION_BLOCK CAN_SendTelegram (* prefix: canst *)`
Actions	Only actions that the block itself calls, beginning with `prv_`. Otherwise, actions do not have a prefix.	

Recommendations for identifiers for POUs in CODESYS V3 libraries

Identifier for	Description	Example
POU	The library prefix is omitted because the name-space replaces the function of the library prefix.	`FUNCTION_BLOCK SendTelegram (* prefix: canst *)`
Method	Only methods that the block itself calls, beginning with `prv_`. Otherwise, methods do not have a prefix.	
Interface Interface	`I`	`ICANDevice`

Recommendations for identifiers for visualizations

NOTICE!
Note that a visualization is not named the same as another block in the project because this may cause problems when changing visualizations.

See also

- *Chapter 1.4.1.8.2 “Declaration of Variables ” on page 222*
- *Chapter 1.4.1.19.5 “Data Types” on page 646*
- *Chapter 1.4.1.19.2 “Variables” on page 526*

1.4.1.19.8 Shadowing Rules

In CODESYS, you are generally allowed to use the same identifier for different elements. For example, a POU and a variable can be named the same. However, you should avoid this practice in order to prevent confusion.

Negative example: In the following code snippet, a local function block instance has the same name as a function:

Example

```
FUNCTION YYY : INT
;
END_FUNCTION

FUNCTION_BLOCK XXX
;
END_FUNCTION_BLOCK

PROGRAM PLC_PRG
VAR
    YYY : XXX;
END_VAR
YYY();
END_PROGRAM
```

In such a case as this, it is unclear whether the instance or the function is called in the program.

To make sure that names are always unique, you should follow naming conventions, such as certain prefixes for variables. Rules for assigning identifiers can be found in the "Identifiers" chapter of the help.

Naming conventions can be checked automatically using the static code analysis of CODESYS. Static code analysis could also detect the duplicate use of the name `YYY` and report it as an error.

The consistent use of the attribute `qualified_only` for enumerations and global variable lists and the use of qualified libraries can also prevent ambiguous situations.

To make sure that a POU of the same name in the *"Devices"* view is not called when a POU in the *"POUs"* view is called, the operator `__POOL` should be prepended (for example, `svar_pou := __POOL.POU();`) when the name of the POU is called.

Shadowing: The compiler does not report any errors or warnings if the same identifier is used for different elements. Instead, the compiler searches the code in a specific order for the declaration of the identifier. If a declaration is found, then the compiler does not search for any other declarations elsewhere. If other declarations do exist, then they are "shadowed" for the compiler. The following section describes the shadowing rules (that is, the search order that the compiler uses when searching for the declaration for identifiers). The section "Ambiguous access and qualified access" provides ways to prevent ambiguous access and bypass shadowing rules.

Search order in the application

When the compiler encounters a single identifier in the code of an application, it searches for the corresponding declaration in the following order:

1. Local variables of a method
2. Local variables in the function block, program, or function, and in any base function blocks
3. Local methods of the POU
4. Global variables in the application, if the `qualified_only` attribute is not set in the variable list where the global variables are declared
5. Global variables in a parent application, if the `qualified_only` attribute is not set in the variable list where the global variables are declared
6. Global variables in referred libraries when neither the library nor the variable list requires qualified access
7. POU or type names from the application (that is, names of global variable lists, function blocks, and so on)
8. POU or type names from a parent application
9. POU or type names from a library
10. Namespaces of locally referred libraries and libraries that are published by libraries
11. Global variables in the *"POUs"* view, unless the `qualified_only` attribute is set in the variable list where they are declared
12. POU or type names from the *"POUs"* view (that is, names of global variable lists, function blocks, and so on)

Libraries that are inserted in the Library Manager of the "POUs" view are mirrored in the Library Manager in all applications in the project with the appropriate placeholder resolution. These libraries then form a common namespace with the libraries in the application. Therefore, there is no shadowing of libraries in the pool by libraries in the application.

Search order in the library

When the compiler encounters a single identifier in the code of a library, it searches for the corresponding declaration in the following order:

1. Local variables of a method
2. Local variables in the function block, program, or function, and in any base function blocks
3. Local methods of the POU
4. Global variables in the local library, if the `qualified_only` attribute is not set in the variable list where the global variables are declared
5. Global variables in referred libraries when neither the library nor the variable list requires qualified access

6. POU or type names from the local library (that is, names of global variable lists, function blocks, and so on)

7. POU or type names from a referred library

8. Namespaces of locally referred libraries and libraries that are published by locally refereed libraries

Ambiguous access and qualified access

Despite these search orders, ambiguous access can still occur. For example, this is the case when a variable with the same name exists in two global variable lists that do not require qualified access. Such a case is reported by the compiler as an error (for example: `ambiguous use of the name XXX`).

This kind of ambiguous usage can be made unique by means of qualified access, for example by accessing via the name of the global variable list (example: `GVL.XXX`).

Qualified access can also always be used to avoid shadowing rules.

- The name of the global variable list can be used to uniquely access a variable in the list.
- The name of a library can be used to uniquely access elements in the library.
- The `THIS` pointer be used to uniquely access variables in a function block, even if a local variable with the same name exists in a method of the function block.

To find the declaration location of an identifier at any time, use the command *"Edit ➔ Browse ➔ Go to Definition"*. This can be especially helpful if the compiler produces an apparently obscure error message.

Searching in instance paths

The search orders described above do not apply to identifiers that exist as components in an instance path or to identifiers that are used as inputs in calls.

For access of the following type `yy.component`, it depends on the entity described by `yy` where the declaration of `component` is searched for.

If `yy` denotes a variable with a structured data type (that is, type `STRUCT` or `UNION`), then `component` is searched for in the following order:

- Local variables of the function block
- Local variables of the base function block
- Methods of the function block
- Methods of the base function block

If `yy` denotes a global variable list or a program, then `component` is searched for in this list only.

If `yy` denotes a namespace of a library, then `component` is searched for in this library exactly as described in the section above "Search order in the library".

Only in the second instance does the compiler decide whether access to the found element is granted (that is, whether the variable is only locally accessible, or whether a method is private). If access is not allowed, an error is issued.

See also

- *Chapter 1.4.1.19.7 "Identifiers" on page 740*
- *Chapter 1.4.1.19.6.2.38 "Attribute 'qualified_only'" on page 726*
- *Chapter 1.4.1.19.2.15 "THIS" on page 539*
- *Chapter 1.4.1.19.3.73 "Operator '__POOL'" on page 630*

1.4.1.19.9 Keywords

In all editors, you must capitalize keywords that for example denote scopes, data types, or operators.

Keywords cannot be used as variable names.

Examples

```
VAR
END_VAR
BOOL_TO_INT
IF
THEN
ELSE
LTIME
MUL
XOR
PERSISTENT
PROGRAM
```

CODESYS checks the correct use of keywords automatically and highlights errors immediately during input with a wavy underline.

When CODESYS creates implicit code, variables and functions are generally given a name that is prepended with two underscores (__). The use of double underscores in the implementation code is prevented automatically. This eliminates conflicts between internal system identifiers and identifiers assigned by the programmer.

The following keywords are used in the CODESYS export format. Therefore, you may not use them as identifiers:

- ACTION
- END_ACTION
- END_FUNCTION
- END_FUNCTION_BLOCK
- END_PROGRAM

Other valid keywords:

- VAR_ACCESS
- READ_ONLY
- READ_WRITE
- PARAMS

1.4.1.19.10 Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'

You can declare the methods explicitly in order to influence the initialization of function block variables, as well as the behavior when exiting function blocks.

The type of the return value for the implicit methods is `BOOL`. The value is not evaluated by the system, but the type should not be changed.

`FB_Init` is always available implicitly and it is used primarily for initialization. For a specific influence, you can also declare the methods explicitly and provide additional code there with the standard initialization code.

`FB_Reinit` must be implemented explicitly. If this method exists, then it is called after the instance of the affected function block is **copied**. That happens during an online change after changes to the function block declaration (signature change) in order to reinitialize the new instance module. To reinitialize the basic implementation of the function block, you must call `FB_Reinit` explicitly.

`FB_Exit` must be implemented explicitly. If there is an implementation, then the method is called before the controller removes the code of the function block instance (implicit call).

The following shows some use cases of these methods for different operating conditions.

Operating condition "First download"

When downloading an application to a PLC with factory settings, the memory of all variables must be offset to the required initial state. In this way, the data areas of function block instances are assigned the required values. By the explicit implementation of `FB_Init` for function blocks, you can react specifically to this situation in the application code. By evaluating the method parameters `bInCopyCode` (`FALSE`) and `bInitRetains` (`TRUE`), you can detect this operating condition clearly. (See "Operating condition "Online Change"" and "Operating condition "Re-download"".)

Operating condition "Online Change"

Within the scope of the online change, you can influence the initialization of function block instances by means of the methods `FB_Exit`, `FB_Init`, and `FB_Reinit`. During the online change, the changes to the application that were made in offline mode are applied in the running PLC. This is the reason that the old instances of the function blocks are replaced by new instances as much as possible without incident. If no changes were made to the declaration part of a function block in the application before login, but in the implementation only, then the data areas are not replaced. Only code blocks are replaced. Then the methods `FB_Exit` , `FB_Init`, and `FB_Reinit` are not called.

If you have made changes to the declaration of a function block that lead to the copying operation described above, then you receive a message during the online change about possible unintended effects. In the “Details” of the message view, you see a list of all instances to be copied.

In the code of the `FB_Init` method, the parameter `bInCopyCode` (`TRUE`) can be evaluated to detect whether or not an online change is being executed.

The following calls occur in succession during an online change:

1. `FB_Exit`

   ```
   old_inst.FB_Exit(bInCopyCode := TRUE);
   ```

   You can call `FB_Exit` when exiting the old instance in order to trigger specific cleanup tasks **before the copy operation**. In this way, you can prepare the data for the following copy operation and influence the state of the new instance. You can notify other parts of the application about the pending change in location in the memory. Pay special attention to the variables of type `POINTER` and `REFERENCE`. These may no longer refer to the required memory locations after the online change. Interface variables (`INTERFACE`) are handled separately by the compiler and they are adapted accordingly during the online change. External resources such as sockets, files, or other handles can be applied by the new instance, in some case unchanged. Often they do not have to be treated specially during an online change. (See "Operating condition "Re-download"")

2. `FB_Init`

   ```
   new_inst.FB_Init(bInitRetains := FALSE, bInCopyCode := TRUE);
   ```

   `FB_Init` is called before the copy operation and can be used in order to execute specific operations for the online change. For example, you can initialize variables accordingly at the new location in the memory, or notify other parts of the application about the new location of specific variables in the memory.

3. Copy operation: `copy`

   ```
   copy(&old_inst, &new_inst);
   ```

   Existing values remain unchanged. For this purpose, they are copied from the old instance into the new instance.

4. `FB_Reinit`

   ```
   new_inst.FB_Reinit();
   ```

   This method is called after the copy operation and should set defined values for the variables of the instance. For example, you can initialize variables accordingly at the new location in the memory, or notify other parts of the application about the new location of specific variables in the memory. Design the implementation independent of the online change. The method can also be called from the application at any time in order to reset a function block instance to its original state.

With the `{attribute 'no_copy'}` attribute, you can prevent that this is copied during the online change for a single variable of the function block. It always retains the initial value.

See also

- *Chapter 1.4.1.20.3.3.19 “Command 'Settings of Memory Reserve for Online Change' ” on page 998*

Operating condition "New download"

When downloading an application, an existing application may be replaced on the PLC. Therefore, the provision of memory for the present function blocks must be regulated. You can use the `FB_Exit` method for implementing the required steps for this. For example, you can offset external resources (with socket and file handles) in a defined state.

You can detect this operating condition by checking whether or not the parameter `bInCopyCode = FALSE` for the `FB_Exit` method.

Operating condition "Start of application"

The initial assignments are processed before the first cycle of the application tasks.

Example

```
T1 : TON := (PT:=t#500ms);
```

These kinds of assignments are executed only after calling `FB_Init`. In order to control the effects of these assignments, you can provide a function block or a method of a function block with the `{attribute 'call_after_init'}` attribute. You must add the attribute above the declaration part of the function block body and above the declaration part of the corresponding method. A POU that extends another POU which uses the `{attribute 'call_after_init'}` attribute must also have the attribute. For the benefit of clarity, we recommend that the corresponding methods are overwritten with the same name, the same signature, and the same attribute. This requires calling `SUPER^.MyInit`. The name of the method can be chosen without restriction. (Exceptions: `FB_Init`, `FB_Reinit`, and `FB_Exit`). The method is called after processing the initial assignments and before starting the application tasks. Therefore, the method can react to user input.

When using `FB_Init` or `{attribute 'call_after_init'}`, remember that detecting errors in the `FB_Init` method or in methods decorated with the `{attribute 'call_after_init'}` attribute is tedious, because the setting of breakpoints may not have the expected effect.

NOTICE!
If the explicitly defined initialization code is reached during execution, then the function block instance is already completely initialized via the implicit initialization code. Therefore, there must not be a `SUPER^.FB_Init` call.

NOTICE!

`FB_Init` replaces the `INI` operator used in CoDeSys V2.3. The methods cannot be compared to the design of a constructor, such as in C#, C++, or Java. This has consequences for function blocks that extend other function blocks. (See below: "Derived function blocks")

Interface of method `FB_Init`

```
METHOD FB_Init : BOOL
VAR_INPUT
  bInitRetains : BOOL; // TRUE: the retain variables are initialized
(reset warm / reset cold)
  bInCopyCode : BOOL;  // TRUE: the instance will be copied to the
copy code afterward (online change)
END_VAR
```

You can declare additional function block inputs in an `FB_init` method. Then you have to set these inputs in the declaration of the function block instance.

Example

Method `FB_Init` for the `serialdevice` function block

```
METHOD PUBLIC FB_Init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // initializing of retain variable
    bInCopyCode : BOOL; // instance is copied to copy code
    iCOMnum : INT; // additional input: number of the COM
interface, that is to be observed
END_VAR
```

Instantiation of the `serialdevice` function block:

```
com1: serialdevice(iCOMnum:=1);
com0: serialdevice(iCOMnum:=0);
```

Interface of method `FB_Reinit`

```
METHOD FB_Reinit : BOOL
```

Interface of method `FB_Exit`

There is the mandatory parameter `bInCopyCode`.

```
METHOD FB_Exit : BOOL
VAR_INPUT
 bInCopyCode : BOOL; // TRUE: the exit method is called in order to
leave the instance which will be copied afterwards (online change).
END_VAR
```

Behavior for derived function blocks

If a function block is derived from another function block, then the `FB_Init` method of the derived function block must define the same parameters as the `FB_Init` method of the basic function block. However, you can add further parameters in order to set up a special initialization for the instance.

Example

The function blocks `MainFB`, `SubFB`, and `SubSubFB` are derived from each other. Therefore, `SubFB EXTENDS MainFB` and `SubSubFB EXTENDS SubFB` apply.

Calling order of methods `FB_Exit` and `FB_Init`:

1. `fbSubSubFb.FB_Exit(...);`
2. `fbSubFb.FB_Exit(...);`
3. `fbMainFb.FB_Exit(...);`
4. `fbMainFb.FB_Init(...);`
5. `fbSubFb.FB_Init(...);`
6. `fbSubSubFb.FB_Init(...);`

See also

- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.19.6.2.3 “Attribute 'call_after_init'” on page 687*
- *Chapter 1.4.1.19.6.2.28 “Attribute 'no_copy'” on page 713*
- *Chapter 1.4.1.19.2.14 “SUPER” on page 538*

1.4.1.19.11 Error Messages and Warnings

Compiler error C0001

Message: Constant '<constant value>' too large for type '<data type>'

Possible error cause: A typed constant is too large for the given data type or a constant is too large for each possible data type.

Error correction: Use smaller constants or an appropriate data type for a typed constant.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    test1: INT;
    test2: INT;
    test3: LREAL;
END_VAR

test1 := 12345678912345566991923939292939911;
test2 := INT#123456;
test3 := 10E500;

--> C0001: Constant '12345678912345566991923939292939911' too large
for type 'ANY_INT'
--> C0001: Constant 'INT#123456' too large for type 'INT'
--> C0001: Constant '10E500' too large for type 'ANY_REAL'
```

Compiler error C0002

Message: '<operator 1>' or '<operator 2>' expected instead of '<tag>'

Possible error cause: Syntax error

Error correction: Use the correct syntax.

Example of the error:

```
PROGRAM PLC_PRG
Fun(1;

--> C0002:  ',' or ')' expected instead of ';'
```

Compiler error C0003

Message: '<value>' is not a valid bit number for '<variable>'

Possible error cause: Attempted access to a bit that is outside of the range for a data type.

Error correction: Use a bit value for the bit access that is lower than the number of bits in the data type of the variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    test1: WORD;
    test2: BOOL;
END_VAR

test1 := test2.17;

--> C0003:  '17' is not a valid bit number for 'w'
```

Compiler Error C0004

Message: '<variable>' is not a component of '<structure>'

Possible error cause: Component access with "." to a variable that is not a structured value or does not exist as a component of the structure.

Error correction: Access a defined component, or change the definition of the component in the data type. The input assistance "List components" provides all valid access to this position.

Example of the error:

```
TYPE DUT:
STRUCT
    x, y : INT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    test1 : DUT;
    test2 : INT;
END_VAR

test2 := test1.z;

--> C0032:  Type 'Unknown type: 'test1.z'' cannot be converted to
type 'INT'
--> C0004:  'z' is to a component of 'DUT'
```

Compiler error C0005

Message: Constant overflow in address '<address>'

Possible error cause: At least one component in the address does not fit into a 32-bit integer value.

Error correction: Use a valid address expression.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    X: BYTE;
END_VAR

X := %QB5555555555;

--> C0005: Constant overflow in address '%??'
```

Compiler error C0006

Message: '<operator>' expected instead of '<token>'

Possible error cause: Syntax error

Error correction: Use the correct syntax.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    x: INT;
    bTest : BOOL;
END_VAR

IF bTest
    x := 9;
END_IF

--> C0006:  'THEN' expected instead of 'x'
```

Compiler error C0007

Message: Expression expected instead of '<token>'

Possible error cause: Syntax error

Error correction: Use the correct syntax.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    x: INT;
    bTest : BOOL;
END_VAR

IF THEN
    x := 9;
END_IF

--> C0007: Expression expected instead of 'THEN'
```

Compiler error C0008

Message: Unexpected end-of-file found: '<operator 1>', '<operator 2>', or '<operator 3>' expected

Possible error cause: Syntax error

Error correction: Use the correct syntax.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    x: INT;
    bTest : BOOL;
END_VAR

IF bTest THEN
    x := 9;

--> C0008: Unexpected end-of-file found: 'ELSIF', 'ELSE' or 'END_IF'
expected
```

Compiler error C0009

Message: Unexpected token '<token>' found

Possible error cause: Syntax error

Error correction: Use the correct syntax.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

END_FOR;

--> C0009: Unexpected token 'END_FOR' found
```

Compiler error C0010

Message: Unexpected end-of-file found: '<token>' expected

Possible error cause: Syntax error

Error correction: Use the correct syntax.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i: INT;
END_VAR

FOR i := 0 TO 2 DO
;

--> C0010: Unexpected end-of-file 'END_FOR' found
```

Compiler error C0011

Message: No 'CASE' label found

Possible error cause: Syntax error in a CASE statement. A statement in a CASE statement is not assigned to a CASE label.

Error correction: Add a CASE label.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i: INT;
    x: INT;
END_VAR

CASE i OF
    x := 9;
END_CASE

--> C0011: No 'CASE' label found
```

Error correction:

```
CASE i OF
0:
    x := 9;
END_CASE
```

Compiler error C0013

Message: At least one statement is expected.

Possible error cause: At least one statement is expected at some positions in the code. For example, in the THEN and ELSE part of an IF statement, or in the body of a FOR loop.

Error correction: Add at least one statement at the selected position. It is enough to write a blank statement ";".

Example of the error:

```
PROGRAM PLC_PRG
VAR
    bTest: BOOL;
END_VAR

IF bTest THEN
END_IF

--> C0013: At least one statement is expected
```

Compiler error C0016

Message: Counter initialization expected

Possible error cause: Syntax error in a FOR loop. The counter variable is not initialized correctly.

Error correction: Pay attention to the correct syntax of the FOR loop.

```
FOR i := 0 TO 10 DO
    ;
END_FOR
```

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i: INT;
END_VAR

FOR i TO 10 DO
    ;
END_FOR

--> C0015: Counter initialization expected
```

Error correction:

```
FOR i := 0 TO 10 DO
    ;
END_FOR
```

Compiler error C0018

Message: <expression> is not a valid assignment target

Possible error cause: An expression with no write permission is on the left side of an assignment. Example: a constant.

Error correction: Assign only to variables that have write access.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i: INT;
END_VAR
VAR CONSTANT
```

```
    j: INT := 0;
END_VAR

j := i;

--> C0018:  'j' is not a valid assignment target
```

Compiler error C0020

Message: '<statement>' is no valid statement

Possible error cause: Syntax error (for example, too few or too many characters)

Error correction: Make sure that the syntax is correct.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    x : INT;
END_VAR

x = 2;

--> C0020: '(x = 2); ' is no valid statement
```

Error correction:

```
Example:
x := 2;
```

Compiler error C0022

Message: '<operator>' needs exactly '<number of operands>' operands

Possible error cause: Too many or too few operands are assigned to an operator.

Error correction: Assign the required number of operands to the operator.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    pt: POINTER TO INT;
END_VAR

pt := ADR(i,1);

--> C0022: 'ADR' needs exactly '1' operands
```

Error correction:

```
Example:
pt := ADR(i);
```

Compiler error C0023

Message: '<operator>' needs at least '<number of operands>' operands

Possible error cause: Too few operands are assigned to an operator.

Error correction: Assign the required number of operands to the operator.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR

i := MUX(30,40);

--> C0023: 'MUX' needs at least '3' operands
```

Error correction:

```
Example:
i := MUX(30,40,50);
```

Compiler error C0026

Message: Identifier expected instead of '<invalid identifier>'

Possible error cause: An invalid identifier is passed to a method.

Error correction: Use valid identifiers.

Example of the error:

```
METHOD 123
VAR_INPUT
END_VAR


--> C0243: The name used in the signature is not identical to the
object name
--> C0026: Identifier expected instead of '123'
```

Error correction:

```
Example:
METHOD METH123
```

Compiler error C0027

Message: size of string expected after '('

Possible error cause: The length of the string is not specified.

Error correction: Specify a string length between the parentheses.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     str : STRING();
END_VAR


--> C0027: size of string expected after '('
--> C0006: ';, :=, REF=, ( or [' expected instead of ')'
```

Error correction:

```
Example:
str : STRING(100);
```

Compiler error C0030

Message: Direct Address expected after 'AT' instead of '<identifier>'

Possible error cause: Either an invalid address or no address is assigned after 'AT'.

Error correction: Specify a valid address.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i AT ABC : INT;
END_VAR


--> C0030: Direct Address expected after 'AT' instead of 'ABC'
```

Error correction:

```
Example:
i AT %IW0 : INT;
```

Compiler error C0031

Message: Type definition expected instead of '<no data type>'

Possible error cause: An invalid type definition is assigned to the identifier.

Error correction: Specify a valid type definition.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : 0;
END_VAR


--> C0031: Type definition expected instead of '0'
```

Error correction:

```
Example:
i : INT;
```

Compiler error C0032

Message: Type '<type 1>' can not be converted to type '<type 2>'

Possible error cause: A variable is assigned to another variable with an incompatible type.

Error correction: Use a type conversion.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    test1: INT;
    test2: STRING;
END_VAR

test1 := test2;

-->C0032: Type 'STRING' cannot be converted to type 'INT'
```

Error correction:

```
Example:
test1 := TO_INT(test2);
```

Compiler Error C0033

Message: Type '<pointer type>' possibly not convertible to type '<data type>' .

Possible error cause: This error occurs only when checking pool objects. An attempt was made to convert a pointer to an integer. Because the size of pointers in a library is unknown, errors may occur when using the library.

Error correction: Use the type `__UXINT` or `__XWORD` for platform-independent calculations with pointers.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    ptr : POINTER TO INT;
    dw : DWORD;
END_VAR
    dw := ptr;

--> C0033: Type 'POINTER TO INT‘ possibly not convertible to type
'DWORD‘.
```

Compiler error C0035

Message: Program name, function or function block instance expected instead of '<invalid function>'

Possible error cause: A function is called that does not exist.

Error correction: Make sure that only program names, functions, and function Block Instances that exist are called.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

PLC_PRG.METH1();

METHOD METH
VAR_INPUT
END_VAR


--> C0004: 'METH1' is no component of 'PLC_PRG'
--> C0035: Program name, function or function block instance expected
instead of 'PLC_PRG.METH1'
```

Error correction:

```
Example:
PLC_PRG.METH();
```

Compiler Error C0036

Message: Cannot call object of type <type>

Possible error cause: An attempt has been made to call an object that does not support any calls.

Error correction: Only functions, function blocks, programs, methods, and actions can be called.

Example of the error:

```
VAR_GLOBAL GVL
    value : INT;
END_VAR
PROGRAM PLC_PRG
GVL();

--> C0036: Cannot call object of type 'VAR_GLOBAL‘.
```

Compiler error C0037

Message: '<invalid input>' is no input of '<function name>'

Possible error cause: A local variable is defined in a function call.

Error correction: Declare the variable as an input parameter.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : INT;
END_VAR
i := TEST(iVar := 1);

FUNCTION TEST : INT
VAR
    iVar : INT;
END_VAR


--> C0037: 'iVar' is no input of 'TEST'
```

Error correction:

```
Example:
VAR_INPUT
    iVar : INT;
END_VAR
```

Compiler error C0038

Message: '<invalid output>' is no output of '<function name>'

Possible error cause: A local variable is handled as an output in a function call.

Error correction: Declare the variable as an output parameter.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : INT;
     x : INT;
END_VAR
i := TEST(iVar => x);

FUNCTION TEST : INT
VAR
    iVar : INT;
END_VAR


--> C0038: 'iVar' is no output of 'TEST'
```

Error correction:

```
Example:
VAR_OUTPUT
    iVar : INT;
END_VAR
```

Compiler error C0039

Message: VAR_IN_OUT '<invalid variable>' must be assigned in call of '<function block name>'

Possible error cause: An IN_OUT variable is not passed to a function block that requires an IN_OUT variable.

Error correction: Assign the IN_OUT variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
inst();

FUNCTION_BLOCK FB
VAR_IN_OUT
    inout : INT;
END_VAR


--> C0039: VAR_IN_OUT 'inout' must be assigned in call of 'FB'
```

Error correction:

```
Example:
inst(inout := i);
```

Compiler error C0040

Message: Function '<function name>' requires exactly '<number of inputs>' input

Possible error cause: Too many or too few parameters are passed to the called function.

Error correction: Pass exactly as many parameters to the function as are expected.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : INT;
END_VAR
i := TEST(1,2);


FUNCTION TEST : INT
VAR_INPUT
    IN: INT;
END_VAR


--> C0040: Function 'TEST' requires exactly '1' inputs
```

Error correction:

```
Example:
i := Test(1);
```

Compiler error C0041

Message: VAR_IN_OUT parameter '<parameter name>' of '<function name>' needs variable with write access as input

Possible error cause: The passed parameter is not a variable with write access (but a constant for example).

Error correction: Pass a VAR_IN_OUT parameter with write access to the function.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : INT;
     x : INT;
END_VAR
i := Test(31415);

FUNCTION TEST : INT
```

```
VAR_IN_OUT
    in_out: INT;
END_VAR


--> C0041: VAR_IN_OUT' parameter 'in_out' of 'TEST' needs variable
with write access as input
```

Error correction:

```
Example:
i := Test(x);
```

Compiler Error C0042 (Compiler Version <= 3.4.10)

Message: Either all or none formal parameter have to be denoted in function call

Possible error cause: The parameters are explicitly assigned to the function in the wrong order.

Error correction: Use uniform formal parameters or implicit parameters.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : INT;
END_VAR
 i := Test(iPar1:=2, 5);

FUNCTION Test : INT
VAR_INPUT
    iPar1 : INT;
    iPar2 : INT;
END_VAR

--> Either all or none formal parameter have to be denoted in
function call
```

Compiler error C0043

Message: Wrong formal parameter: '<parameter name>' expected in this place

Possible error cause: The parameters are assigned to the function explicitly in the wrong order.

Error correction: Specify the parameters in the correct order.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     i : INT;
END_VAR
i := Test(iPar2 := 2, 5);


FUNCTION Test : INT
VAR_INPUT
    iPar1 : INT;
    iPar2 : INT;
END_VAR


--> C0043: Wrong formal parameter: 'iPar1' expected in this place
--> C0412: Multiple input assignments for parameter ''
```

Error correction:

```
Example:
i := Test(5, iPar2 := 2);
```

Compiler error C0044

Message: Assignment to input missing for parameter '<input variable name>' in call of '<function block name>'

Possible error cause: A parameter is passed although an input variable is not declared.

Error correction: Declare an input variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
inst(1);


FUNCTION_BLOCK FB
VAR_INPUT

END_VAR


--> C0044: Assignment to input missing for parameter '1' in call of
'FB'
```

Error correction:

```
Example:
VAR_INPUT
    in : INT;
END_VAR
```

Compiler error C0045

Message: Use of 'THIS' is not allowed in this context

Possible error cause: In order to be assigned to the current instance, THIS can be used only in a method, action, transition, or in the body of a function block. This error message appears for all other positions.

Error correction: Use THIS in an allowed context only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    test1: INT;
END_VAR

THIS^.test1 := 19;

--> C0018: 'THIS^.test1' is not a valid assignment target
--> C0062: 'THIS^' is not a structure variable
--> C0045: Use of 'THIS' is not allowed in this context
```

Compiler error C0046

Message: Identifier '<identifier name>' not defined

Possible error cause: An identifier is used that is not declared.

Error correction: Declare the variables that you want to use.

Example of the error:

```
PROGRAM PLC_PRG
VAR

END_VAR

i := 1;


--> C0018: 'i' is no valid assignment target
--> C0046: Identifier 'i' not defined
```

Error correction:

```
Example:
VAR
    i : INT;
END_VAR
```

Compiler error C0047

Message: Cannot apply indexing with '[]' to an expression of type '<data type>'

Possible error cause: A data type that is not an array is indexed with '[]'.

Error correction: Index data types with '[]' only if they are declared as arrays.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
i[1];

--> C0047: Cannot apply indexing with '[]' to an expression of type
'INT'
```

Compiler error C0048

Message: Array requires exactly '<number>' indexes

Possible error cause: Too many or too few indexes are specified when using an array.

Error correction: Specify as many indexes as there are dimensions assigned to the array.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr1 : ARRAY[1..2,1..3] OF INT;
END_VAR
arr1[1] := 5;


--> C0048: Array requires exactly 2 indexes
```

Error correction:

```
Example:
arr1[1,2] := 5;
```

Compiler error C0049

Message: The constant index '<index value>' is not within the range from '<start index>' to '<end index>'

Possible error cause: An index is specified that is outside the size of the array.

Error correction: Use only indexes that are within the size of he array.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr1 : ARRAY[1..2] OF INT;
END_VAR
arr1[3] := 1;


--> C0049: The constant index '3' is not within the range from '1' to
'2'
```

Error correction:

```
Example:
arr1[2] := 1;
```

Compiler error C0050

Message: Bitaccess requires literal or symbolic integer constant

Possible error cause: No literal or an integer constant is specified in a bit access.

Error correction: Use a literal or an integer constant.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    x : INT;
END_VAR

i.x := FALSE;


--> C0018: 'i.x' is no valid assignment target
--> C0050: Bitaccess requires literal or symbolic integer constant
```

Error correction:

```
Example:
i := Test(x);
```

Compiler Error C0051

Message: Single byte string expected for an attribute value instead of '<value>‘.

Possible error cause: A character string does not appear at the displayed location as expected.

Error correction: Replace the current value with a string.

Example of the error:

```
PROGRAM PLC_PRG
{IF hasattribute(pou: MyPOU, MyAttribute)}
{END_IF}

--> C0051: Single byte string expected for an attribute value instead
of 'MyAttribute'.
```

Error correction:

```
PROGRAM PLC_PRG
{IF hasattribute(pou: MyPOU, 'MyAttribute')}
{END_IF}
```

Compiler Error C0053

Message: Compiler version <version> has been withdrawn. Please use a higher compiler version instead.

Possible error cause: The current compiler version cannot ne used.

Error correction: Adapt the current compiler version in the project (Project Environment, Project Settings).

Compiler error C0061

Message: Bitaccess on function call is not allowed

Possible error cause: Bit access is performed on a function.

Error correction: Use bit access only for supported data types.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

Test().2;

FUNCTION Test : INT
VAR_INPUT

END_VAR


--> C0061: Bitaccess on function call is not allowed
```

Compiler error C0062

Message: '<variable name>' is no structured variable

Possible error cause: A variable that is not a structure variable is treated like a structure variable.

Error correction: Make sure that the variable is a structure variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pt : PUNKT;
    i : INT;
END_VAR

i.x := 1024;


TYPE Punkt :
STRUCT
    x : REAL;
    y : REAL;
END_STRUCT
```

```
END_TYPE


--> C0018: 'i.x' is no valid assignment target
--> C0062: 'Variable' is no structured variable
```

Error correction:

```
Example:
pt.x := 1024;
```

Compiler error C0064

Message: Dereferencing requires a pointer

Possible error cause: A variable that is not a pointer variable is dereferenced.

Error correction: Dereference only variables that are pointer variables.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    pi : POINTER TO INT;
END_VAR
i^:=1;


--> C0018: 'i^' not a valid assignment target
--> C0064: Dereferencing requires a pointer
```

Error correction:

```
Example:
pi := ADR(i);
pi^ := 1;
```

Compiler Error C0065

Message: There is no global definition for '<name>'.

Possible error cause: The value searched for is not a global variable, global POU, or other value that can be accessed globally.

Error correction: Declare '<name>' as a global variable.

Example of the error:

```
PROGRAM PLC_PRG
.someValue := 5;


--> C0065: There is no global definition for 'someValue‘
```

Error correction:

```
Example:
VAR_GLOBAL
    someValue : INT;
END_VAR
```

Compiler error C0066

Message: Cannot compare type '<data type>' with type '<data type>'

Possible error cause: Two data types are compared which cannot be compared with each other.

Error correction: Compare only data types that can be compared with each other.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    re : REAL;
    str: STRING;
    b : BOOL;
END_VAR

b := i > str;


--> C0066: Cannot compare type 'INT' with type 'STRING'
```

Error correction:

```
Example:
b := i > re;
```

Compiler error C0068

Message: Compare not possible on objects of type '<data type>'

Possible error cause: Objects are being compared in which a comparison is not possible.

Error correction: Compare only data types in which a comparison is possible (INT, REAL, etc.).

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BOOL;
    arr1 : ARRAY [1..2] OF INT;
    arr2 : ARRAY [1..2] OF INT;
END_VAR

b := arr1 > arr2;


--> C0068: Compare not possible on objects of type 'ARRAY [1..2]'
```

Compiler error C0069

Message: Compare not possible on objects of type '<data type>' or '<data type>'

Possible error cause: Two different objects are being compared in which a comparison is not possible.

Error correction: Compare only data types in which a comparison is possible (INT, REAL, etc.).

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BOOL;
    arr1 : ARRAY [1..2] OF INT;
    arr2 : ARRAY [1..3] OF INT;
END_VAR

b := arr1 > arr2;


--> C0069: Compare not possible on objects of type 'ARRAY [1..2]' or
'ARRAY [1..3]'
```

Compiler error C0070

Message: 'INI' operator needs function block instance or data unit type instance

Possible error cause: Neither a function block instance nor a DUT instance is applied to the INI operator.

Error correction: Pass only function block instances or DUT instances to the INI operator.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BOOL;
    inst : FB;
END_VAR

b := INI(b, TRUE);

FUNCTION_BLOCK FB
VAR
END_VAR


--> C0070: 'INI' operator needs function block instance or data unit
type instance
```

Error correction:

```
Example:
b := INI(inst, TRUE);
```

Compiler error C0072

Message: Operator <operator name>' is not possible on type '<data type>'

Possible error cause: An operator is applied to an incompatible type.

Error correction: Apply operators only on compatible types.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    str : STRING;
END_VAR

str := ABS(str);


--> C0072: Operator 'Abs' is not possible on type 'STRING'
```

Error correction:

```
Example:
i := ABS(i);
```

Compiler error C0074

Message: Unexpected array initialisation

Possible error cause: Syntax error in the array initialization

Error correction: Correct the syntax

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr1 : INT := [1,2,3,4,5,6];
END_VAR
```

```
--> C0074: Unexpected array initialisation
--> C0032: Cannot convert type 'Unknown type: [1,2,3,4,5,6]' to type
'INT'
```

Error correction:

```
Example:
arr1 : ARRAY [1..6] OF INT := [1,2,3,4,5,6];
```

Compiler error C0075

Message: Too many initializers for array

Possible error cause: Too many values are specified for the size of the array.

Error correction: The number of assigned values must correspond to the size of the array.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5,6];
END_VAR
```

```
--> C0075: Unexpected array initialisation
```

Error correction:

```
Example:
arr1 : ARRAY [1..6] OF INT := [1,2,3,4,5,6];
```

Compiler error C0076

Message:Unexpected structure initialisation

Possible error cause: Syntax error in the structure initialization

Error correction: Make sure that the syntax is correct.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    st1 : INT := (p1 := 1);
END_VAR
```

```
--> C0076: Unexpected structure initialisation
--> C0032: Cannot convert type 'STRUCT(p1:=1)' to type 'INT'
--> C0046: Identifier 'p1' not defined
--> C0018: 'p1' is no valid assignment target
```

Error correction:

```
Example:
st1 : STRUCT1 := (p1:=1,p2:=10);
```

Compiler error C0077

Message: Unknown type: '<data type>'

Possible error cause: Invalid data type in the declaration (maybe a syntax error)

Error correction: Specify valid data types only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INTEGER;
END_VAR


--> C0077: Unknown type: 'INTEGER'
```

Error correction:

```
Example:
arr1 : ARRAY[1..2] OF STRUCT1 := (p1:=1,p2:=10);
```

Compiler Error C0078

Message: Unsupported type: '<data type>'

Possible error cause: The used type is not supported by the current device and therefore cannot be used.

Error correction: If possible, use a different type. For example, `REAL` instead of `LREAL`.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    value : LREAL;
END_VAR


--> C0078: Unsupported type: 'LREAL‘
```

Error correction:

```
Example:
PROGRAM PLC_PRG
VAR
    value : REAL;
END_VAR
```

Compiler error C0080

Message: Functionblock '<function block name>' must be instantiated to be accessed

Possible error cause: Missing function Block Instantiation

Error correction: Instantiate the function block.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

FB();


FUNCTION_BLOCK FB
VAR
END_VAR


--> C0080: Functionblock 'FB' must be instantiated to be accessed
```

Error correction:

```
Example:
VAR
    inst : FB;
END_VAR
inst();
```

Compiler error C0081

Message: Unexpected Pragma: '<pragma name>' found without matching 'if'

Possible error cause: The IF condition is missing when using the pragma.

Error correction: Complete the IF condition of the pragma.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR

i := 5;
{END_IF}


--> C0081: Unexpected Pragma: 'END_IF' found without matching 'if'
```

Error correction:

```
Example:
{IF <expression>}
i := 5;
{END_IF}
```

Compiler error C0082

Message: '<invalid pragma>' is no valid condition for pragma

Possible error cause: When using a pragma, an invalid expression is used in the IF condition.

Error correction: Use valid pragma conditions.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR

{IF abc}
i := 5;
{END_IF}

--> C0082: '!!!ERROR!!!' is no valid condition for pragma
```

Error correction:

```
Example:
{IF defined (abc)}
```

Compiler error C0084

Message: '<pragma operand>' is no valid operand for pragma

Possible error cause: Syntax error

Error correction: Use valid pragma operands.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

{IF defined(0)}
{END_IF}


--> C0084: 'defined(null)' is no valid operand for pragma
```

Error correction:

```
Example:
{IF defined (abc)}
```

Compiler Error C0085

Message: Define value expected instead of '<value>‘.

Possible error cause: A string is expected instead of the current value at the displayed location of the pragma.

Error correction: Replace the current value with a string.

Example of the error:

```
PROGRAM PLC_PRG

{IF hasvalue(define, defineValue)}
{END_IF}


--> C0086: C:0085: Define value expected instead of 'defineValue‘
```

Error correction:

```
Example:
PROGRAM PLC_PRG

{IF hasvalue(define, '120')}
{END_IF}
```

Compiler error C0086

Message: No definition found for interface '<interface name>'

Possible error cause: An undefined interface is used.

Error correction: Define the interface.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : FB;
END_VAR


FUNCTION_BLOCK FB IMPLEMENTS XY
VAR
END_VAR


--> C0086: No definition found for interface 'XY'
```

Error correction:

```
Example:
INTERFACE XY
```

Compiler error C0087

Message: There is no implementation for method '<method name>' defined in interface '<interface name>'.

Possible error cause: One of the methods specified by the interface has not be provided by the implemented function block.

Error correction: Implement all methods that are specified by the interface.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : FB;
END_VAR

INTERFACE XY
METHOD METH1
VAR_INPUT
END_VAR

FUNCTION_BLOCK FB IMPLEMENTS XY
VAR
END_VAR
METHOD METH2
VAR_INPUT
END_VAR


--> C0087: There is no implementation for method 'METH1' defined in
interface 'XY'
```

Compiler error C0089

Message: Interface of overridden method '{0}' of interface '{1}' does not match declaration

Possible error cause: The signature of the implemented method does not match the signature of the method in the interface.

Error correction: Make sure that the same return types and parameters are declared.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : FB;
END_VAR

INTERFACE XY
METHOD METH1
VAR_INPUT
    iPar : INT;
END_VAR

FUNCTION_BLOCK FB IMPLEMENTS XY
VAR
END_VAR
METHOD METH1
VAR_INPUT
```

```
END_VAR


--> C0089: Interface of overridden method 'METH1' of interface 'XY'
does not match declaration
```

Compiler error C0090

Message: No definition found for base class '<function name>'

Possible error cause: The function block specified as the base does not exist or is not a function block.

Error correction: Use a function block as the base.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : FB;
END_VAR

FUNCTION_BLOCK FB EXTENDS POU
VAR
END_VAR

FUNCTION POU
VAR
END_VAR


--> C00090: No definition found for base class 'POU'
```

Compiler error C0091

Message: Recursion in base function block list: <function name>

Possible error cause: A base function block is extended by itself.

Error correction: Recursion in base function block lists is not possible.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : FB;
END_VAR


FUNCTION_BLOCK FB EXTENDS FB
VAR
END_VAR


--> C00091: Recursion in base function block list: FB -> FB
```

Compiler error C0094

Message: Interface of overridden method '<method name>' of interface '<function block name>' doesn't match declaration

Possible error cause: The signature of the method of the first interface does not match the signature of the method in the second interface, which is extended by the first.

Error correction: Align the signatures.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     xyz : XY2;
END_VAR

FUNCTION_BLOCK XY
METHOD METH1
VAR_INPUT
END_VAR

FUNCTION_BLOCK XY2 EXTENDS XY
METHOD METH1
VAR_INPUT
    iPar : BOOL;
END_VAR

--> C00094: Interface of the overridden method METH1 of interface XY
doesn't match declaration
```

Compiler error C0096

Message: Only one base function block may be defined in EXTENDS-list.

Possible error cause: Two or more base function blocks are defined in the EXTENDS list.

Error correction: Define only one base function block in the EXTENDS list.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    fb : FB;
END_VAR

FUNCTION_BLOCK FB EXTENDS FB2, FB3
VAR
END_VAR

FUNCTION_BLOCK FB2
VAR
END_VAR

FUNCTION_BLOCK FB3
VAR
END_VAR


--> C00096: Only one base function block may be defined in EXTENDS-
list
```

Compiler error C0097

Message: Duplicate definition of variable '<variable name>' in function block '<function block name>' and in base '<base function block name>'

Possible error cause: A variable is declared with the same name in a function block and its base.

Error correction: Use different variable names.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    fb : FB;
END_VAR


FUNCTION_BLOCK FB EXTENDS FB2
VAR
    i : INT;
END_VAR


FUNCTION_BLOCK FB2
VAR
    i : INT;
END_VAR


--> C00097: Duplicate definition of variable 'i' in function block
'FB' and in base 'FB2'
```

Compiler error C0098

Message: The keyword "FUNCTIONBLOCK" is no longer supported. Use "FUNCTION_BLOCK" instead.

Possible cause of error: Syntax error

Error correction: Use the keyword "FUNCTION_BLOCK".

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : FB;
END_VAR

FUNCTIONBLOCK FB
VAR
END_VAR


--> C00098: The keyword "FUNCTIONBLOCK" is no longer supported. Use
"FUNCTION_BLOCK" instead. Use "FUNCTION_BLOCK" instead.
```

Compiler Error C0099 (Compiler Version < 3.5.7.0)

Message: Local defined enumeration are no longer supported. Use datatype definition instead.

Possible error cause: A local enumeration declaration was used together with a compiler version that does not support this.

Error correction: Use a later compiler version, or define the enumeration in a DUT.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    localEnumVar : (RED, GREEN, BLUE) := RED;
END_VAR


--> C0099: Local defined enumeration are no longer supported. Use
datatype definition instead.
```

Compiler error C0101

Message: Data Recursion: '<recursion>'

Possible error cause: Recursive data initialization over two function blocks

Error correction: Avoid recursions for data initialization.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst0 : FB1;
END_VAR

FUNCTION_BLOCK FB1
VAR
    inst1 : FB2;
END_VAR

FUNCTION_BLOCK FB2
VAR
    inst2 : FB1;
END_VAR


--> C0101: Data Recursion: FB1->FB2->FB1
```

Compiler error C0102

Message: Out of retain memory: Variable '<variable name>', <byte size> bytes.

Possible error cause: More retain memory is used than is available on the PLC. It is also possible that the retain memory is too fragmented due to incremental builds.

Error correction: Use the *“Clean”* for fragmenting the memory. This will force the reallocation of all data at the next build.

Compiler error C0104

Message: 'Out of global data memory: Variable '<variable name>', <byte size> bytes.

Possible error cause: More memory for data is used than is available on the PLC. It is also possible that the memory is too fragmented due to incremental builds.

Error correction: Use the *“Clean”* for fragmenting the memory. This will force the reallocation of all data at the next build.

Compiler error C0114

Message: Invalid destination <jump label> for 'JMP'

Possible error cause: Syntax error or typographical error in the JMP destination

Error correction: Correct the typographical or syntax error.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
JMP 0;


--> C0114: Invalid destination 0 for 'JMP'
```

Compiler Error C0115

Message: The second parameter of a conditional call (????ALWAYS CALC????) has to be a valid call statement.

Error correction: Specify the call of a function, method, or function block in the second parameter of the conditional ????CALC??? call.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    condition : BOOL;
END_VAR
CALC(condition, 1+2);


--> C0115: The second parameter of a conditional call has to be a
valid call statement.
```

Error correction:

```
Example:
CALC(condition, MyFunction(1,2))
```

Compiler error C0116

Message: The label '<jump label>' is a duplicate

Possible error cause: A label is defined multiple times.

Error correction: Define each label one time only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
JMP label;
label:

label:

--> C0116: The label 'LABEL' is a duplicate
```

Compiler error C0117

Message: No such label '<jump label>' within the scope of the 'JMP' statement

Possible error cause: A jump is made to a label that does not exist.

Error correction: Define the label that you specify as the destination.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
JMP A;


--> C0117: No such label 'A' within the scope of the 'JMP' statement
```

Compiler error C0118

Message: The label '<jump label>' has not been referenced.

Possible error cause: A jump label is defined that is not referenced.

Error correction: Remove the unused jump labels.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
LABEL:


--> C0118: The label ' LABEL' has not been referenced
```

Compiler error C0119

Message: An 'FB_init'-Method of a functionblock or struct needs two inputs 'bInitRetains' and 'bInCopyCode' of type BOOL

Possible error cause: One or both of the inputs 'bInitRetains' and 'bInCopyCode' of type BOOL is missing.

Error correction: Define the missing inputs.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR


FUNCTION_BLOCK FB
METHOD FB_init
VAR_INPUT
END_VAR


--> C0119: An 'FB_init'-Method of a functionblock or struct needs two
inputs 'bInitRetains' and 'bInCopyCode' of type BOOL
```

Error correction:

```
Example:
METHOD FB_init
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode : BOOL;
END_VAR
```

Compiler error C0120

Message: An 'FB_Exit'-Method of a functionblock or struct needs an input 'bInCopyCode' of type BOOL.

Possible error cause: The input 'bInCopyCode' of type BOOL is missing.

Error correction: Define the input.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR


FUNCTION_BLOCK FB
```

```
METHOD FB_exit
VAR_INPUT
END_VAR


--> C0120: An 'FB_Exit'-Method of a functionblock or struct needs an
input 'bInCopyCode' of type BOOL.
```

Error correction:

```
Example:
METHOD FB_exit
VAR_INPUT
    bInCopyCode : BOOL;
END_VAR
```

Compiler error C0122

Message: Expression 'SUPER' is not allowed in this context

Possible error cause: "SUPER^" is used outside of derived function blocks.

Error correction: Use "SUPER^" in function blocks only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

SUPER^.METH(TRUE, TRUE);


--> C0122: Expression 'SUPER' is not allowed in this context
```

Compiler error C0124

Message: 'Initialization' is no valid initialization for an enumeration

Possible error cause: A data type that is not ANY_INT is used for the enum initialization.

Error correction: Use only ANY_INT for enum initializations.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : DUT;
END_VAR

TYPE DUT :
(
  enum_member := 1.5
) DWORD;
END_TYPE


--> C0032: Cannot convert type 'LREAL' to type 'DUT'
--> C0124: 'Initialization' is no valid initialization for an
enumeration
```

Compiler error C0125

Message: The constant <constant value> is assigned to more than one enumeration.

Possible error cause: The same value is assigned to two or more enumerations.

Error correction: Assign different values to the enumerations.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : DUT;
END_VAR

TYPE DUT :
(
    enum_member := 0,
    enum_member2 := 0
);
END_TYPE


--> C0125: The constant 0 is assigned to more than one enumeration
```

Compiler error C0126

Message: Variable of type '<data type>' requires exactly 1 Index

Possible error cause: Multiple indexes are assigned to a variable with one index.

Error correction: Assign only one index.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pi : POINTER TO INT;
END_VAR
pi[0,1] := 0;


--> C0126: Variable of type 'POINTER TO INT' requires exactly 1 Index
```

Error correction:

```
Example:
pi[0] := 0;
```

Compiler error C0130

Message: <object> '<object name>' referenced without parentheses '()'

Possible error cause: A method is referenced without parentheses.

Error correction: Always reference methods by means of parentheses.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
inst.METH1


--> C0130: METHOD 'METH1' referenced without parentheses '()'
```

Error correction:

```
Example:
inst.METH1();
```

Compiler error C0131

Message: '<value>' is not allowed as operand for 'ADR'

Possible error cause: A constant is passed as an operand to the operator ADR.

Error correction: Use only valid operands for ADR.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    pt : POINTER TO INT;
END_VAR

pt := ADR(1);


--> C0131: '1' is not allowed as operand for 'ADR'
```

Error correction:

```
Example:
pt := ADR(i);
```

Compiler error C0132

Message: No enclosing loop of which to EXIT

Possible error cause: EXIT is used outside of a loop.

Error correction: Use EXIT inside of a loop only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

EXIT
;


--> C0132: No enclosing loop of which to EXIT
```

Compiler error C0136

Message: ambiguous use of name '<variable name>'

Possible error cause: A variable is declared in multiple GVLs.

Error correction: Qualify the variable with the desired GVL.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    j : INT := g_i;
END_VAR

GVL1:
VAR_GLOBAL
    g_i : INT;
END_VAR

GVL2:
```

```
VAR_GLOBAL
    g_i : INT;
END_VAR

--> C0136: ambiguous use of name 'g_i'
```

Error correction:

```
Example:
j : INT := GVL1.g_i;
```

Compiler Error C0138

Message: No matching 'FB_Init' method found for instantiation of POU.

Possible error cause: No `FB_Init` method exists that accepts the passed parameters.

Error correction: Check which arguments `FB_Init` has to receive and adjust the passed arguments.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    myPOU : POU(arg1 := 1, arg2 := 2);
END_VAR


--> C0138: No matching 'FB_Init‘ method found for instantiation of
POU.
```

Compiler error C0139

Message: The code <code> has no effect. Is this the intent?

Possible error cause: The written code is syntactically correct but does not do anything.

Error correction: Write code that has a purpose.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR

i;

--> C0139: The code 'i;' has no effect. Is this the intent?
```

Compiler error C0140

Message: Reference assign is only allowed to variables of Reference type

Possible error cause: An attempt is made to assign a reference value to a variable not defined as a reference type.

Error correction: Define the variable as a reference type.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    I_r : INT;
END_VAR
```

```
I_r REF= i;

--> C0140: Reference assign is only allowed to variables of Reference
type
```

Error correction:

```
Example:
I_r : REFERENCE TO INT;
```

Compiler error C0141

Message: Reference assign needs variable with write access

Possible error cause: A constant is assigned to the reference assignment.

Error correction: Assign a writable variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    I_r : REFERENCE TO INT;
END_VAR

I_r REF= 314;

--> C0141: Reference assign needs variable with write access
```

Error correction:

```
Example:
I_r REF= i;
```

Compiler error C0142

Message: A local variable named '<variable name>' is already defined in '<pou name>'

Possible error cause: The same variable name is used two times.

Error correction: Use different variable names.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    i : INT;
END_VAR

--> C0142: A local variable named 'i' is already defined in 'PLC_PRG'
```

Compiler error C0143

Message: The property '<property name>' cannot be used in this context because it lacks the get accessor

Possible error cause: The property does not have Get access.

Error correction: Make sure that the property has a Get access definition.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    inst: FB;
END_VAR

i := inst.Prop;

FUNCTION_BLOCK FB
VAR
END_VAR

PROPERTY Prop : INT
Set;

--> C0143: The property 'Prop' cannot be used in this context because
it lacks the get accessor
```

Compiler error C0144

Message: Inheritance only allowed in Functionblocks, Interfaces and Structures

Possible error cause: An attempt is made to use inheritance in an object that does not permit inheritance.

Error correction: Use `EXTENDS` in function blocks, interfaces, and structures only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : DUT_1;
END_VAR

TYPE DUT:
(
    enum_member := 0
);
END_TYPE

TYPE DUT_1 EXTENDS DUT:
(
    enum_memberX := 0
);
END_TYPE

--> C0144: Inheritance only allowed in Functionblocks, Interfaces and
Structures
```

Compiler error C0145

Message: Interfaces can only be implemented by Functionblocks

Possible error cause: An attempt is made to implement an interface outside of a function block.

Error correction: Implement interfaces only in function blocks.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

POU();

FUNCTION POU IMPLEMENTS ITF
VAR
END_VAR

--> C0145: Interfaces can only be implemented by Functionblocks
```

Compiler error C0149

Message: Variable declarations are not allowed in interfaces

Possible error cause: An attempt is made to define a variable in an interface.

Error correction: Do not define variables in interfaces.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : ITF;
END_VAR

INTERFACE ITF
VAR_INPUT
    i : INT;
END_VAR

--> C0149: Variable declarations are not allowed in interfaces
```

Compiler error C0161

Message: Border <array bound> of array is no constant value

Possible error cause: A variable is specified as an array bound.

Error correction: Use constants for the array bounds.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT := 3;
    arr1 : ARRAY[1..i] OF INT;
END_VAR

--> C0161: Border 'i' of array is no constant value
```

Error correction:

```
Example:
arr1 : ARRAY[1..3] OF INT;
```

Compiler error C0162

Message: Number <number of array values> of array initialisations is no constant value

Possible error cause: The initialization `[Wert1,AnzahlWert2(Wert2)]` works only with a constant for `AnzahlWert2`.

Error correction: Use constants only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT := 3;
    arr1 : ARRAY[1..4] OF INT := [1,i(7)];
END_VAR

--> C0162: Number 'i' of array initialisations is no constant value
```

Error correction:

```
Example:
arr1 : ARRAY[1..4] OF INT := [1,3(7)];
```

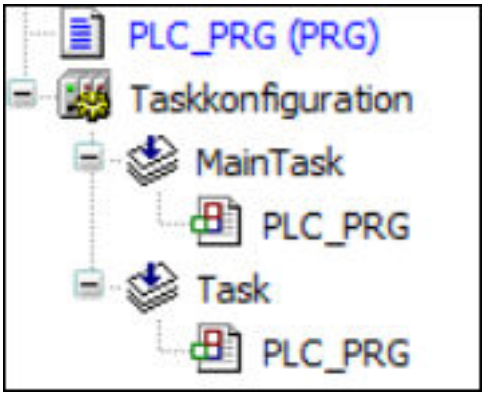
Compiler Error C0164

Message: POU <name> writes to output <name> and is called in several tasks.

Possible error cause: The device setting `codegenerator\check-multiple-task-output-write` is set and multiple tasks access the same output.

Error correction: Do not call a program that changes outputs in multiple tasks.

Example of the error:



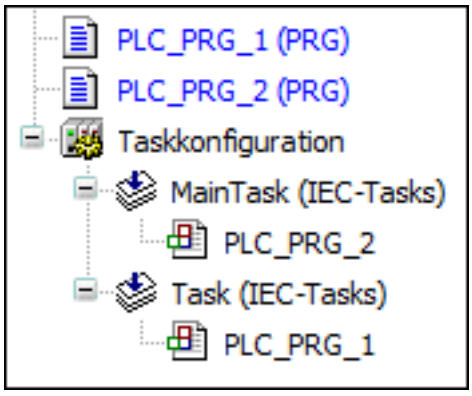

```
PROGRAM PLC_PRG
VAR
    Output AT %QB7 : BYTE
END_VAR

Output := 0;


--> C0164: POU 'PLC_PRG' writes to output 'QB7' and is called in
several tasks
```

Compiler Error C0165

Message: Variable '<variable name>‘, which is mapped on address '<address>‘ is written in different tasks.

Possible error cause: The device setting `codegenerator\check-multiple-task-output-write` is set and multiple tasks access the same output.

Error correction: Write an output in one fixed task only. If multiple tasks need to calculate data for one output, then you should try to transfer this information by means of global variables to one fixed task, which then writes the data to one output.



Example of the error:

```
PROGRAM PLC_PRG_1
VAR
    Output AT %QB7 : BYTE;
END_VAR
Output := 0;

PROGRAM PLC_PRG_2
VAR
    Output AT %QB7 : BYTE;
END_VAR
Output := 1;


--> C0165: Variable 'Output', which is mapped on address 'QB7' is
written in different tasks.
```

Compiler error C0168

Message: 'VAR_CONFIG' declaration only allowed in VAR_CONFIG - list

Possible error cause: 'VAR_CONFIG' is used outside of a VAR_CONFIG list.

Error correction: Use 'VAR_CONFIG' only in VAR_CONFIG lists.

Example of the error:

```
PROGRAM PLC_PRG
VAR_CONFIG
    i : INT;
END_VAR

--> C0168: 'VAR_CONFIG' declaration only allowed in VAR_CONFIG - list
```

Compiler error C0169

Message: 'VAR_GLOBAL' declaration only allowed in Global variable list

Possible error cause: 'VAR_GLOBAL' is used outside of global variable lists.

Error correction: Use 'VAR_GLOBAL' in global variable lists only.

Example of the error:

```
PROGRAM PLC_PRG
VAR_GLOBAL
    i : INT;
END_VAR

--> C0169: 'VAR_GLOBAL' declaration only allowed in Global variable
list
```

Compiler Error C0173

Message: '<keyword>' not allowed in this place

Possible error cause: A declaration keyword (example: `VAR_INPUT`, `VAR_OUTPUT`, or `VAR`) is not allowed at this location.

Error correction: Correct the declaration: Inputs and outputs are not useful or necessary in type definitions or global variable lists.

```
TYPE DUT :
STRUCT
    member : INT;
END_STRUCT
END_TYPE
```

Example of the error:

```
TYPE DUT :
STRUCT
    VAR_INPUT
        member : INT;
    END_VAR
END_STRUCT
END_TYPE


--> C0173: 'VAR_INPUT' not allowed in this place.
```

Error correction

```
Example:
TYPE DUT :
STRUCT
    member : INT;
END_STRUCT
END_TYPE
```

Compiler error C0174

Message: 'VAR_TEMP' declaration not allowed in this place

Possible error cause: 'VAR_TEMP' is used outside of a program or function block.

Error correction: Use 'VAR_TEMP' inside of programs and function blocks only.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

FUN();

FUNCTION FUN
VAR_TEMP
END_VAR

--> C0174: 'VAR_TEMP' declaration not allowed in this place
```

Compiler error C0175

Message: 'RETAIN' or 'PERSISTENT' not allowed in this place

Possible error cause: 'RETAIN' or 'PERSISTENT' is used in a function.

Error correction: Use 'RETAIN' or 'PERSISTENT' at the intended locations.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

POU_1();

FUNCTION POU_1
VAR RETAIN
END_VAR

--> C0175: 'RETAIN' or 'PERSISTENT' not allowed in this place
```

See also

- *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537*
- *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537*

Compiler error C0177

Message: '<object>' is of type 'type' and cannot be instantiated

Possible error cause: An attempt is made to instantiate a function.

Error correction: Instantiate only objects that can be instantiated.

Example of the error:

```
PROGRAM PLC_PRG
VAR
     inst : POU;
END_VAR

FUNCTION POU
VAR
END_VAR

--> C0177: 'POU' is of type 'FUNCTION' and cannot be instantiated
```

Compiler error C0178

Message: No external access to 'VAR_IN_OUT' parameter '<parameter name>' of '<object name>'

Possible error cause: An attempt is made to remotely access a 'VAR_IN_OUT' parameter.

Error correction: Do not remotely access 'VAR_IN_OUT' parameters.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
    i : INT;
END_VAR
i := inst.in_out;

FUNCTION_BLOCK FB
VAR_IN_OUT
    in_out : INT;
END_VAR

--> C0178: No external access to 'VAR_IN_OUT' parameter 'in_out' of
'FB'
```

Compiler Error C0179

Message: '<identifier>' is no output of 'Function block'

Possible error cause: The initialization of a function block instance must not contain `VAR_IN_OUT` variables.

Error correction: Use `VAR_IN_OUT` variables in function block calls only. When initializing a function block instance, only assign the inputs of a function block.

Example of the error:

```
Example:
FUNCTION_BLOCK MyFB
VAR_IN_OUT
    inOut : INT;
END_VAR

PROGRAM PLC_PRG
VAR
    iValue : INT;
    fb : MyFB := (inOut := iValue);
END_VAR


--> C0179: 'inOut' is no output of 'MyFB'
```

Compiler Error C0180

Message: Ambiguous namespace '<library 1>' defined by library '<library 2>'

Possible error cause: The namespace of the library <library 1> is not unique. It is already used for <library 2>.

Error correction: Change the namespace of the library accordingly (*"Properties"* button in the Library Manager).

Example of the error:


```
--> C0180: Ambiguous namespace 'STANDARD' defined by library
'Standard, 3.5.15.0 (System)'
```

Compiler error C0182

Message: Return type is only possible for POUs of Type FUNCTION and METHOD

Possible error cause: An attempt is made to define a return value in a program.

Error correction: Define a return value only in methods and functions.

Example of the error:

```
PROGRAM PLC_PRG : BOOL
VAR
END_VAR

--> C0182: Return type is only possible for POUs of Type FUNCTION and
METHOD
```

Compiler Error C0183

Message: Global scope operation '.' is not valid on expression '<expression>'

Possible error cause: The '.' operator is used to access a global variable. However, at this location it is not followed by a valid IEC identifier, but for example a character such as ";" or a reserved identifier such as `FUNCTION`, or an operator such as `TO_STRING`.

Error correction: Use a valid IEC identifier for a global variable.

Example of the error:

```
PROGRAM PLC_PRG
...

iVar := .FUNCTION;// ERROR: C0183 because ; is not a valid identifier
strVar := .TO_STRING;
-->  C0183: Global scope operation '.' is not valid on expression
'<expression>'
```

Error correction

```
Example: globalValue is declared in a GVL.
PROGRAM PLC_PRG
iVar := .globalValue;
```

Compiler error C0185

Message: It is not possible to perform component access '.', index access '[]' or call '()' on result of function call. Assign result to help variable first.

Possible error cause: Component or index access to the result of a function call is performed.

Error correction: Assign the result to a variable in order to access.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    it : ITF;
END_VAR

POU_1()[0].METH1();

FUNCTION POU_1 : ARRAY[0..0] OF ITF

INTERFACE ITF

METHOD METH1

--> C0185: It is not possible to perform component access '.', index
access '[]' or call '()' on result of function call. Assign result to
help variable first.
```

Compiler Error C0186

Message: It is not possible to compare interface that is return value of call. Assign to variable first.

Possible error cause: A comparison operation is applied to an interface that is returned by a function.

Error correction: First assign the result of the function call to a variable and then compare the value of the variable. This will also reduce the number of function calls that are required.

Example of the error:

```
INTERFACE MyInterface

FUNCTION GetInterface : MyInterface

PROGRAM PLC_PRG
IF GetInterface() <> 0 THEN
    // ...
END_IF

--> C0186: It is not possible to compare interface that is return
value of call. Assign to variable first.
```

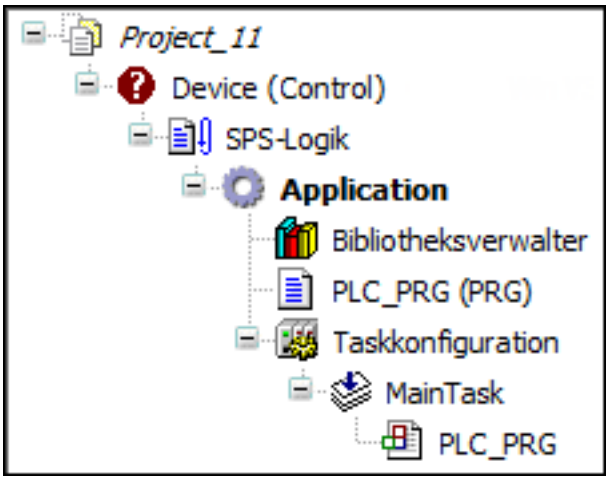
Error correction:

```
Example:
PROGRAM PLC_PRG
VAR_TEMP
    tempInterface : MyInterface;
END_VAR
tempInterface := GetInterface();
IF tempInterface <> 0 THEN
    // ...
END_IF
```

Compiler Error C0188

Message: Device not installed to the system. No code generation possible.

Possible error cause: The desired device is not installed.



Error correction: Install the missing device in the device repository, or replace the existing device already inserted in the device tree with another existing device (*"Update Device"*).

Compiler error C0189

Message: ';' expected instead of '<token>'

Possible error cause: Syntax error

Error correction: Make sure that the syntax is correct.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    INT
END_VAR

--> C0009: Unexpected Token '<Token>' found
--> C0189: ';' expected instead of 'INT'
```

Compiler error C0190

Message: ';' expected instead of end of POU

Possible error cause: Syntax error in the POU

Error correction: Make sure that the syntax is correct.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
i := 5

--> C0190: ';' expected instead of end of POU
```

Compiler error C0191

Message: The operator 'INDEXOF' is no longer supported. Use ADR instead. ADR on a POU-Name returns a Pointer to a Pointer to the function code.

Possible error cause: The outdated operator 'INDEXOF' is used.

Error correction: Use the operator 'ADR'.

Compiler error C0195

Message: Implicit conversion from signed Type '<data type 1>' to unsigned Type '<data type 2>' : possible change of sign

Possible error cause: A sign conflict may have been missed in the implicit conversion.

Error correction: Convert only data types with the same sign implicitly.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    b : UINT;
END_VAR

b := i;

--> C0195: Implicit conversion from signed Type 'INT' to unsigned
Type 'UINT' : possible change of sign
```

Compiler error C0196

Message: Implicit conversion from unsigned Type '<data type 1>' to signed type '<data type 2>' : possible change of sign

Possible error cause: A sign conflict may have been missed in the implicit conversion.

Error correction: Use explicit conversions.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    b : UINT;
END_VAR

i := b;

--> C0196: Implicit conversion from unsigned Type 'UINT' to signed
type 'INT' : possible change of sign
```

Compiler error C0197

Message: Implicit conversion from '<data type 1>' to '<data type 2>': possible loss of information

Possible error cause: An attempt is made to convert a variable from data type DINT or LINT to data type REAL.

Error correction: For DINT, use the data type LREAL, and when converting from LINT to LREAL make sure that the value of LINT does not exceed the capacity of LREAL.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : DINT;
    b : REAL;
END_VAR
b := i;

--> C0197: Implicit conversion from 'DINT' to 'REAL': possible loss
of information
```

Compiler error C0198

Message: String constant '<string value>' too long for destination type '<data type>'

Possible error cause: The string constant has too many characters.

Error correction: Use shorter string constants or declare larger strings.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    str : STRING(4) := '12345';
END_VAR

--> C0198: String constant '12345' too long for destination type
'STRING(4)'
```

Compiler error C0199

Message: Interface '<interface name>' must be instantiated to be accessed

Possible error cause: An attempt is made to access an interface method without the interface being instantiated.

Error correction: Instantiate the interface.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

ITF.METH();

INTERFACE ITF

METHOD METH
VAR_INPUT
END_VAR

--> C0199: Interface 'ITF' must be instantiated to be accessed
```

Error correction:

```
Example:
itest: ITF;
```

Compiler error C0201

Message: Type '<data type 1>' is not equal to type '<data type 2>' of VAR_IN_OUT 'Variable'

Possible error cause: The data type that is passed to the function as a VAR_IN_OUT parameter does not match the data type defined in it.

Error correction: Pass a variable with the correct data type.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    Inst: POU;
    b : BOOL;
END_VAR

inst(in_out := b);

FUNCTION_BLOCK POU
VAR_IN_OUT
    in_out : INT;
END_VAR

--> C0201: Type 'BOOL' is not equal to type 'INT' of VAR_IN_OUT
'Variable'
```

Compiler error C0203

Message: Only Structures and Function Blocks can contain variables of type BIT.

Possible error cause: An attempt is made to declare a variable of type BIT outside of structures and function blocks.

Error correction: Declare variables of type BIT only in structures and function blocks.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BIT;
END_VAR

--> C0203: Only Structures and Function Blocks can contain variables
of type BIT.
```

Compiler error C0204

Message: Variables of type BIT must be declared within a VAR_INPUT-, VAR_OUTPUT or VAR-block

Possible error cause: An attempt is made to define a variable of type BIT or as a VAR_IN_OUT parameter.

Error correction: Define variables of type BIT only within a VAR_INPUT, VAR_OUTPUT or VAR block.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR


FUNCTION_BLOCK FB
VAR_IN_OUT
    b : BIT;
END_VAR

--> C0204: Variables of type BIT must be declared within a
VAR_INPUT-, VAR_OUTPUT or VAR-block
```

Compiler error C0205

Message: POINTER TO BIT is not allowed

Possible error cause: An attempt is made to declare a POINTER TO BIT.

Error correction: Do not declare POINTER TO BIT.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pt : POINTER TO BIT;
END_VAR

--> C0205: POINTER TO BIT is not allowed
```

Compiler error C0206

Message: BIT is not allowed as base type of an array

Possible error cause: An attempt is made to declare a BIT array.

Error correction: Do not declare BIT arrays.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr : ARRAY[1..2] OF BIT;
END_VAR

--> C0206: BIT is not allowed as base type of an array
```

Compiler Error C0207

Message: There is no system definition for '<identifier>'

Possible error cause: An attempt was made to access a variable in `__SYSTEM` that does not exist.

Error correction: Check and correct the specified identifier of the respective variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    Value : INT;
END_VAR
Value := __SYSTEM.UnknownVariable;

--> C0207: There is no system definition for 'UnkownVariable'
```

Compiler error C0208

Message: 'MOD' is not defined for 'REAL'

Possible error cause: An attempt is made to perform a modulo operation with a variable of type `REAL`.

Error correction: Modulo operations are only possible with variables of type `ANY_INT`.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    r1 : REAL;
END_VAR
r1 := r1 MOD 2;

--> C0208: 'MOD' is not defined for 'REAL'
```

Compiler Error C0209

Message: You have defined '<number>' applications for device '<device name>'. The maximum number is '<number>'. So you will not be able to download all applications.

Possible error cause: Some devices only support a specific number of applications (device description). If a project contains more applications, then not all will be downloaded to the device.

Error correction: Remove applications from your project or use another device.

Compiler error C0211

Message: Variable declaration expected instead of <expression>

Possible error cause: Syntax error

Error correction: Make sure that the syntax is correct.

Example of the error:

```
PROGRAM PLC_PRG
VAR
   VAR

   END_VAR
END_VAR

--> C0211: Variable declaration expected instead of VAR END_VAR
```

Compiler error C0212

Message: VAR, VAR_INPUT, VAR_OUTPUT or VAR_INOUT expected instead of <expression>

Possible error cause: Syntax error

Error correction: Make sure that the syntax is correct.

Example of the error:

```
PROGRAM PLC_PRG
i : INT;

--> C0212: VAR, VAR_INPUT, VAR_OUTPUT or VAR_INOUT expected instead
of i : INT;
```

Compiler Error C0215

Message: Direct address declaration is not possible in persistent list

Possible error cause: Persistent variables are not allowed to have a direct address.

Error correction: Remove the direct address assignment in the persistent variable list.

Example of the error:

```
VAR_GLOBAL PERSISTENT RETAIN
    directAddressVar AT %QB7 : BYTE;
END_VAR


--> C0215: Direct address declaration is not possible in persistent
list.
```

Compiler error C0216

Message: Case label duplicate

Possible error cause: A CASE label is used multiple times.

Error correction: Use each CASE label only one time.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR

CASE i OF
  1: i := i+1;
  1: i := i+2;
ELSE
  i := i+10;
END_CASE;

--> C0216: Case label duplicate
```

Compiler error C0217

Message: Case label <case label> also contained in range <case range begin> .. <case range end>

Possible error cause: A CASE label is part of the range of another CASE label.

Error correction: Make sure that there is no intersecting.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
CASE i OF
  3..5: i := i+2;
  4: i := i+2;
ELSE
  i := i+10;
END_CASE;

--> C0217: Case label 4 also contained in range 3 .. 5
```

Compiler error C0218

Message: Case label requires literal or symbolic integer constant

Possible error cause: An attempt is made to use a variable as a CASE label.

Error correction: Use only literals and symbolic integer constants.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    a : INT := 2;
END_VAR

CASE i OF
  1: i := i+1;
  a: i := i+2;
ELSE
  i := i+10;
END_CASE;

--> C0218: Case label requires literal or symbolic integer constant
```

Compiler error C0219

Message: Case contains overlapping range <case range 1 begin> .. <case range 1 end> and <case range 2 begin> .. <case range 2 end>

Possible error cause: Two branches of CASE markers have the same elements or subsets.

Error correction: Make sure that there is no intersecting.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
CASE i OF
  3..5: i := i+2;
  1..4: i := i+2;
ELSE
  i := i+10;
END_CASE;

--> C0219: Case contains overlapping range 1 .. 4 and 3 .. 5
```

Compiler error C0221

Message: Direct Address '<address>' malformed

Possible error cause: An address is not displayed completely.

Error correction: Make sure that the address is displayed correctly.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    xVar : BOOL;
END_VAR;
xVar := %IX0;

--> C0221: Direct Address '%IXO' malformed
```

Error correction:

```
Example:
xVar := %IX0.2;
```

Compiler error C0222

Message: Outputs can't be of type 'REFERENCE TO'

Possible error cause: An attempt is made to define `REFERNCE TO` as an output parameter.

Error correction: Do not use `REFERENCE TO` as an output parameter.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

FUNCTION_BLOCK FB
VAR_OUTPUT
    re : REFERENCE TO INT;
END_VAR

--> C0222: Outputs can't be of type 'REFERENCE TO'
```

Compiler error C0224

Message: Call Recursion: <recursion>

Possible error cause: A function calls itself.

Error correction: Make sure that functions are not recursive.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

POU();

FUNCTION POU
VAR
END_VAR

POU();

--> C0224: Call Recursion: POU -> POU
```

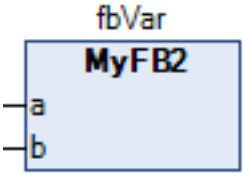
Compiler Error C0225

Message: '<name>' is not an instance of '<name>'

Possible error cause: A function block in a graphical programming language has been assigned with an explicitly specified type that does not match the declared type.

Error correction: Replace the explicit type with the one used in the declaration part, or remove the specification of the explicit type from the POU.

Example of the error:

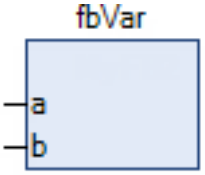



```
PROGRAM PLC_PRG
VAR
    fbVar : MyFB;
END_VAR
```

```
--> C0225: 'fbVar‘ is not an instance of 'MyFB2‘
```

Error correction:

or



Compiler error C0227

Message: Initialisation of constant variable <constant name> not constant

Possible error cause: A constant is initialized with a variable.

Error correction: Initialize constants only with constant values.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
VAR CONSTANT
    k : INT := i;
END_VAR
```

```
--> C0227: Initialisation of constant variable 'k' not constant
```

Compiler error C0228

Message: No initial value for constant variable '<constant name>'

Possible error cause: A constant is not initialized.

Error correction: Initialize the constants.

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
VAR CONSTANT
    k : INT;
END_VAR

--> C0228: No initial value for constant variable 'k'
```

Error correction:

```
Example:
k : INT := 1;
```

Compiler Error C0230

Message: Type name '<data type>' not expected in this place

Possible error cause: The data type name of an enumeration is used at an invalid position.

Error correction: Check whether the data type name is used correctly at this location. Maybe there is a spelling error.

Example of the error:

```
TYPE MyEnum :
(
    enum_member := 0
);
END_TYPE

PROGRAM PLC_PRG
VAR
    value : INT;
END_VAR
value := MyEnum;
MyEnum := value;


--> For PLC_PRG, the error message is issued 2x:
C0230:  Type name 'MyEnum' not expected in this place
```

Error correction:

```
Example:
value := MyEnum.enum_member;
MyEnum.enum_member := value;
```

Compiler Error C0232

Message: Array initialisation expected

Possible error cause: An array of arrays is initialized, but the initialization values are not nested.

Error correction: Use a nested array initialization as shown in the example below.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    value : ARRAY[0..2] OF ARRAY[0..2]  OF INT := [1,2,3];
END_VAR


--> C0232: Array initialisation expected
```

Error correction:

```
Example:
value : ARRAY[0..2] OF ARRAY[0..2]  OF INT := [
[1,2,3],
[4,5,6],
[7,8,9]];
```

Compiler Error C0233

Message: Initialisation list for {0} <data type> expected

Possible error cause: An array of the type of a structure is initialized with elements that are not structure initializations or variables.

Error correction: As shown in the example below, use structure initializations or existing variables to initialize arrays of structures.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    values : ARRAY[0..2] OF COLOR := [1,2,3];
END_VAR


--> C0233: Initialisation list for COLOR expected
```

Error correction:

```
Example:
PROGRAM PLC_PRG
VAR
      colorVariable : COLOR := (red:=0, green:=0, blue:=255);
      value : ARRAY[0..2] OF COLOR := [
            colorVariable,
            (red:=255, green:=0, blue:=0),
            (red:=0, green:=255, blue:=0)];
END_VAR
```

Compiler error C0234

Message: First Operand of __QueryInterface must be an interface reference or the instance of a function block

Possible error cause: Incorrect operands are passed to the operator `__QueryInterface`.

Error correction: Pass an interface reference or the instance of a function block.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    a : INT;
    ITFref, ITFref2 : ITF;
    ITFref2 : ITF2;
END_VAR

__QueryInterface(a ,ITFref);
```

```
INTERFACE ITF EXTENDS __SYSTEM.IQueryInterface
INTERFACE ITF2 EXTENDS ITF

--> C0234: First Operand of __QueryInterface must be an interface
reference or the instance of a function block
```

Error correction:

```
Example:
__QueryInterface(ITFref2, ITFref);
```

Compiler error C0235

Message: Second Operand of __QueryInterface must be an interface reference

Possible error cause: Incorrect operands are passed to the operator `__QueryInterface`.

Error correction: Pass an interface reference.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    a : INT;
    ITFref, ITFref2 : ITF;
    ITFref2 : ITF2;
END_VAR

__QueryInterface(ITFref2, a);

INTERFACE ITF EXTENDS __SYSTEM.IQueryInterface
INTERFACE ITF2 EXTENDS ITF

--> C0235: Second Operand of __QueryInterface must be an interface
reference
```

Error correction:

```
Example:
__QueryInterface(ITFref2, ITFref);
```

Compiler error C0236

Message: Wrong type definition for VAR_EXTERNAL <variable name>

Possible error cause: The variable is declared in `VAR_GLOBAL` / `VAR_EXTERNAL` as different types.

Error correction: Use the same type definition in `VAR_GLOBAL` and `VAR_EXTERNAL`.

Example of the error:

```
PROGRAM PLC_PRG
VAR_EXTERNAL
    ig : STRING;
END_VAR

VAR_GLOBAL
    ig : INT;
END_VAR

--> C0236: Wrong type definition for VAR_EXTERNAL ig
```

Compiler error C0237

Message: No global definition found for VAR_EXTERNAL '<variable name>'

Possible error cause: An attempt is made to declare a variable in `VAR_EXTERNAL` which does not exist in `VAR_GLOBAL`.

Error correction: Make sure that the identifiers match.

Example of the error:

```
PROGRAM PLC_PRG
VAR_EXTERNAL
    i : INT;
END_VAR

VAR_GLOBAL
    ig : INT;
END_VAR

--> C0237: No global definition found for VAR_EXTERNAL 'i'
```

Compiler error C0238

Message: No initial value allowed for VAR_EXTERNAL <variable name>

Possible error cause: An attempt is made to initialize a variable in `VAR_EXTERNAL`.

Error correction: Do not initialize variables in `VAR_EXTERNAL`.

Example of the error:

```
PROGRAM PLC_PRG
VAR_EXTERNAL
    ig : INT := 2;
END_VAR

VAR_GLOBAL
    ig : INT;
END_VAR

--> C0238: No initial value allowed for VAR_EXTERNAL ig
```

Compiler error C0239

Message: Interface <interface name 1> does not extend <interface name 2>

Possible error cause: The used interface does not extend another interface.

Error correction: Extend the interface.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    ITFref : ITF;
    ITFref2 : ITF2;
END_VAR

__QueryInterface(ITFref2,ITFref);

INTERFACE ITF
INTERFACE ITF2 EXTENDS ITF

--> C0239: Interface ITF__Union does not extend
__System.IQueryInterface
```

Error correction:

```
Example:
INTERFACE ITF EXTENDS __System.IQueryInterface
```

Compiler error C0240

Message: First Operand of __QueryPointer must be an interface reference or the instance of a function block

Possible error cause: Incorrect operands are passed to the operator `__QueryPointer`.

Error correction: Pass an interface reference or the instance of a function block.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    a : INT;
    ITFref : ITF;
    pt : POINTER TO FB;
END_VAR

__QueryPointer(a,pt);

--> C0240: First Operand of __QueryPointer must be an interface
reference or the instance of a function block
```

Error correction:

```
Example:
__QueryPointer (ITFref, pt);
```

Compiler error C0241

Message: Second Operand of __QueryPointer must be pointer

Possible error cause: Incorrect operands are passed to the operator `__QueryPointer`.

Error correction: Pass a pointer.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : INT;
    ITFref : ITF;
    pt : POINTER TO FB;
END_VAR

__QueryPointer(ITFref,b);

INTERFACE ITF EXTENDS __System.IQueryInterface

--> C0241:  Second Operand of __QueryPointer must be pointer
```

Error correction:

```
Example:
__QueryPointer (ITFref, pt);
```

Compiler error C0242

Message: Operand of __DELETE must be pointer

Possible error cause: An incorrect operand is passed to the operator `__DELETE`.

Error correction: Pass a pointer.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    a : INT;
    pt : POINTER TO INT;
END_VAR

__DELETE(a);

--> C0242: Operand of __DELETE must be pointer
```

Error correction:

```
Example:
__DELETE (pt);
```

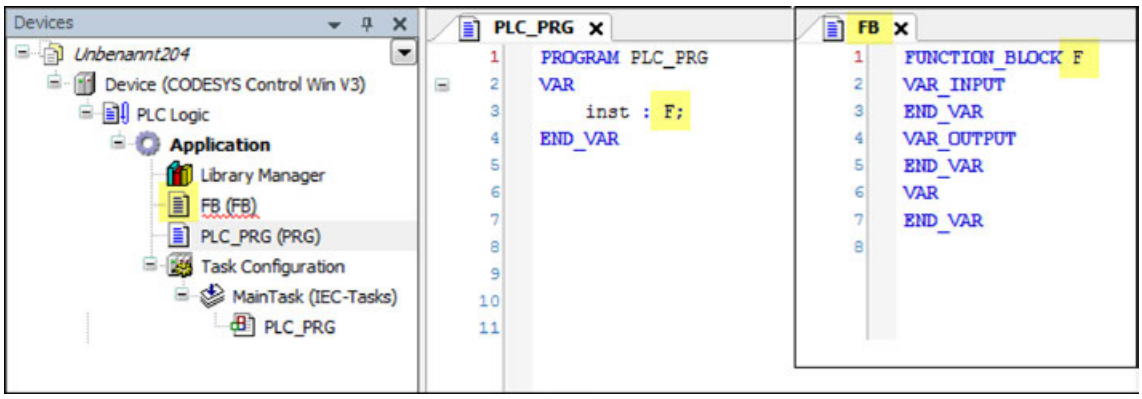
Compiler error C0243

Message: The name used in the signature is not identical to the object name

Possible error cause: The object name differs from the name used in the code.

Error correction: make sure that the names are the same.

Example of the error:



Compiler Error C0380

Message: The Operators LOWER_BOUND and UPPER_BOUND are only supported for arrays of variable length.

Possible error cause: One of the two operators LOWER_BOUND or UPPER_BOUND is not used for an array of variable length.

Error correction: Use the operators LOWER_BOUND and UPPER_BOUND only for an array of variable length.

> *For compiler version 3.5.14.0 and higher, the operators can also be used for static arrays. As a result, the error C0380 occurs only in the case of earlier compiler versions.*

Example of the error:

```
FUNCTION_BLOCK POU
VAR_IN_OUT
    arrin : ARRAY [*] OF INT;
END_VAR
VAR
    arrtest : ARRAY [0..5] OF INT;
    test1: DINT;
    test2: DINT;
```

```
END_VAR

test1 := UPPER_BOUND(arrin, 1);
test2 := UPPER_BOUND(arrtest, 1);

--> C0380: The operators LOWER_BOUND and UPPER_BOUND are supported
only for arrays with variable length.
```

Compiler error C0509

Message: Multiple assignments for operator '__New' not allowed

Possible error cause: In one line of code, the assignment operator ":=" is called a multiple number of times with the `__New` operator.

Error correction: Program the memory allocation with the `__New` operator in a separate line of code for each pointer that points to dynamically allocated memory.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pbAlpha : POINTER TO BYTE; // Typed pointer to Alpha
    pbBeta: POINTER TO BYTE;    // Typed pointer to Beta
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pbBeta := pbAlpha := __NEW(BYTE); // Incorrect code for memory
allocation
END_IF

pbBeta := pbAlpha := 16#01;

IF (xDelete) THEN
    __DELETE(pbAlpha); // Frees memory of pointer
END_IF

--> C0509: Multiple assignments for operator '__New' not allowed
```

Error correction:

```
PROGRAM PLC_PRG
VAR
    pbAlpha : POINTER TO BYTE; // Pointer to Alpha
    pbBeta: POINTER TO BYTE;    // Pointer to Beta
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pbAlpha := __NEW(BYTE); // Allocates memory for Alpha
    pbBeta :=  __NEW(BYTE); // Allocates memory for Beta
END_IF

pbBeta := pbAlpha := 16#01; // Multiple assignment

IF (xDelete) THEN
    __DELETE(pbAlpha); // Frees memory of pointer
END_IF
```

See also

- *Chapter 1.4.1.19.3.58 “Operator '__NEW'” on page 614*
- *Chapter 1.4.1.19.6.2.12 “Attribute 'enable_dynamic_creation'” on page 695*

Compiler error C0511

Message: The function block '<function block name>' is ABSTRACT and cannot be used as a target for an assignment.

Possible error cause: A value was assigned to an abstract function block. The concrete function blocks may have different types and therefore cannot be copied.

Error correction: In order to copy the data of the function block, concrete function blocks have to be used.

Example of the error:

```
PROGRAM PLC_PRG
VAR
refAbstract1 : REFERENCE TO AbstractPOU;
refAbstract2 : REFERENCE TO AbstractPOU;
END_VAR

refAbstract1 := refAbstract2;
--> C0511: The function block 'refAbstract1' is ABSTRACT and cannot
be used as a target for an assignment.
```

Error correction:

Use the reference assignment `REF=` to assign the reference `refAbstract1` to the same function block as `refAbstract2`.

Compiler Error C0542

Message: Inheritance is not intended for the data type "UNION" <data type name>.

Possible error cause: A structured data type (`DUT`) is derived from a `UNION` by extending with `EXTENDS`, or a `UNION` is derived from a `DUT`. This kind of derivation is not permitted. However, for reasons of compatibility only a warning is issued.

Example of the error

```
TYPE U_StringExt EXTENDS U_StringBase :
UNION
    str10 : STRING(10);
END_UNION
END_TYPE TYPE U_StringBase :
UNION
    str20 : STRING(20);
END_UNION
END_TYPE PROGRAM PLC_PRG
VAR
    uStringExt : U_StringExt;
END_VAR

uStringExt.str20 := 'a234567890b234567890'; -> C0542
```

Compiler Error C0543

Message: The name <keyword> is a reserved keyword in the IEC 1131-3 standard. An error will be issued in future versions.

Possible error cause: A reserved keyword was assigned as the name of a variable.

Error correction: Rename the variable.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    char : BYTE;
END_VAR
```

```
--> C0543: The name 'char' is a reserved keyword in the IEC 1131-3
standard. An error will be issued in future versions.
```

Note: For violations in compiled libraries, only a text message (information) will be issued instead of a warning.

The following keywords are reserved:

- CHAR
- WCHAR
- ANY_DERIVED
- ANY_ELEMENTARY
- ANY_MAGNITUDE
- ANY_SIGNED
- ANY_DURATION
- ANY_CHARS
- ANY_CHARS
- CHAR_TO
- TO_CHAR
- WCHAR_TO
- TO_WCHAR
- ATAN2
- USING
- CLASS

See also

- *Chapter 1.4.1.19.9 “Keywords” on page 747*

1.4.1.20 Reference, User Interface

1.4.1.20.1 Notifications

Notifications inform you about important information, such as available updates or security notices.

To open the *“Notifications”* view, click the icon in the upper right corner of the frame window of CODESYS. All received notifications are displayed in this view. Notifications marked as *“read”* are deleted from the list the next time CODESYS is started.

The red icon 3 indicates that new notifications are available, as well as how many.

1.4.1.20.2 Objects

Objects in CODESYS provide special functionalities to create applications. Examples: Application, program, function, Library Manager, devices, image pool. Objects are managed in tree structures in the views *“Devices”*, *“POUs”* and *“Modules”*.

You can add an object to the belonging "tree" by use of the command *“Project ➔ Add Object”*. The possible insert positions depends on the position within the tree.

Each object provides properties, which can be viewed and accessed with the command from the context menu of the object.

See also

- *Chapter 1.4.1.20.3.3.22 “Command 'Properties'” on page 1000*

Object 'Application'

Symbol:

The object is displayed as a node in the device tree. It comprises the objects which are required for a controller program to run.

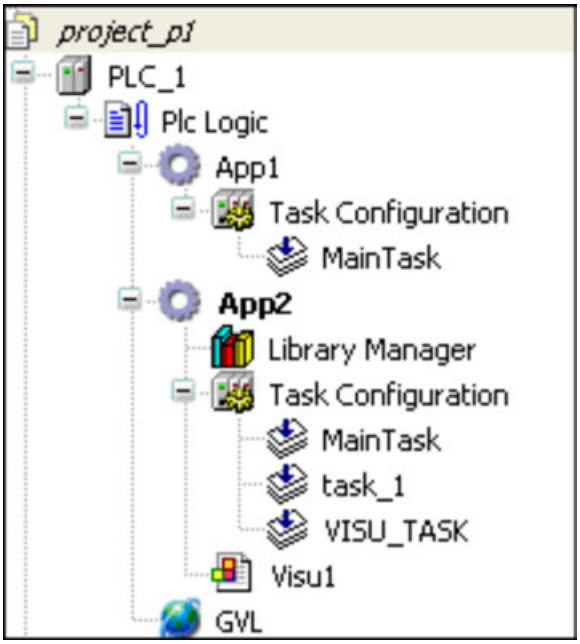
You can insert an application object below a *"PLC Logic"* node (below a programmable device) or as a child application below an existing application (parent application).

Below each application, there has to be a *"Task Configuration"* where you configure which program of the application will be called by which task and using which settings.

Furthermore, you insert the POUs of your controller program below an application, for example POUs, global variables lists, and the Library Manager. These POUs are available only for this application and its child applications.

In addition, the application can also use instances of project-global POUs. You manage project-global POUs in the *"POUs"* view. The use of these kinds of instances follows the thinking behind object-oriented programming.

Multiple applications can be inserted below a PLC device object. To do this, they have to have unique names.



NOTICE!
An online change after a changing the parent application will remove the child application from the PLC.

When multiple applications are directly below a device object, for the I/O handling of the device you have to define the application whose variables CODESYS should use for communication with the target system. The settings are configured on the *"PLC Settings"* tab of the device editor.

The application that you want to work with in online mode has to be set as the "active application" (see *"App2"* in the figure above).

You can set special properties for an application on the *"Application Build Options"* tab of the *"Properties"* dialog of the application object. Example: Activation of dynamic memory allocation.

When downloading the application to the PLC, you can include information about the application contents. This is also a setting on the *"Application Build Options"* tab. Then later you can compare the application on the controller with the active application in CODESYS.

If you want to add individual information about the author, version, and an individual short description, you can modify the information in the general *“Project Information”* on the *“Information”* tab of the *“Properties”* dialog.

When you want to log in to the target system (PLC or simulation) with an application, it will first be checked which applications are currently on the PLC and whether or not the application parameters on the controller match those in the project configuration. Corresponding messages will notify you about mismatches and possible options for further action. In this step, you can also delete applications from the PLC.

On the *“Application”* tab of the device editor, you can see which applications currently exist on the device. There you can also delete applications from the target system. It is possible that you also see additional applications which are not represented by a separate object in the device tree, for example the `<application>_symbols.app`, which contains a symbol list created for the application (see “Symbol Configuration”).

See also

- *Chapter 1.4.1.20.2.26 “Object 'Task Configuration'” on page 937*
- *Chapter 1.4.1.20.4.10.9 “Dialog 'Properties - Application Build Options'” on page 1162*
- *Chapter 1.4.1.10 “Downloading an Application to the PLC” on page 379*
- *Chapter 1.4.1.20.2.8.9 “Tab 'PLC Settings'” on page 850*
- *Chapter 1.4.1.9.2 “Symbol Configuration” on page 357*
- *Chapter 1.4.1.13.1 “Executing the online change” on page 439*
- *Chapter 1.4.1.20.3.4.13 “Command 'Project information'” on page 1007*

Object 'POU Locations'

This object is available only for specific controllers. It is displayed automatically in the device tree. The object cannot be added or removed manually. The object can be used for mapping the executable code of an application in different code areas on the controller. Specifically small controllers often have limited internal code areas (flash memory). If one or more additional code areas (for example, external flash memory) are available on the controller, then the location of the code POUs of an application can be changed specifically.

If there are no specific requirements, then the code POUs are stored sequentially in the code areas (*“default”*). This means that the next code POUs are stored in the next areas only when the first code area is filled. In the *“POU Locations”* editor, you will see the current location of the POUs in the memory areas and you can change them specifically.

Editor 'POU Locations'

Double-clicking the *“POU Locations”* object in the device tree of the controller opens the editor. Then it receives the entry *“<application>”*. After a code generation, all program blocks of the application are displayed with the respective object type, current location in the memory, and code size.

In the *“Configured Location”* column, you can set one of the memory areas other than the *“Current Location”* for each POU or library.

In order to move the POUs to the recently configured memory locations, you first have to *“Clean”* and then *“Generate Code”* again.

Make sure to pay attention to the messages in the category “POU Locations”. This also shows when a code POU cannot be moved as expected.

"Clean"	Deletion of the compile information for the application. Corresponds to the menu command *"Build ➔ Clean"*. Requirement for moving the POUs to the configured memory locations.
"Generate code"	Starting of the code generation for the application. Corresponds to the menu command *"Build ➔ Generate Code"*. Requirement for moving the POUs to the configured memory locations.
"Objects"	Objects of the application, including the objects from the referenced libraries
"Type"	Object type; examples: *"Function block"*, *"Method"*, *"Library"*
"Current location"	Current memory location of the POU: `area_<n>`.
"Configured location"	Configured memory location where the POU is moved at the next code generation. Possible values: ● *"default"*: Automatically assigned area. ● *"area_<n>"*: Explicitly assigned memory area (n=number)
"Code size"	Code size of the POU (in bytes)

- *Chapter 1.4.1.20.3.5.2 "Command 'Clean'" on page 1021*
- *Chapter 1.4.1.20.3.5.1 "Command 'Generate Code'" on page 1021*

Objects for Alarm Management

The help pages for alarm management are summarized in the help for CODESYS Visualization. So please see there for help on the following objects::

- Object *"Alarm Configuration"*
- Object *"Alarm Class"*
- Object *"Alarm Group"*
- Object *"Alarm Storage"*
- Object *"Remote Alarms"*

Object 'Data Source Manager'

Symbol: 

The object is used as a node for data sources below it. At least one data source has to exist. An application with the data source manager communicates with remote devices.

See also

- *Chapter 1.4.1.9.4 "Data Link with Data Sources" on page 363*

Command 'Add Object' > 'Data Source'

Function: The command opens the *"Add Data Source"* dialog.

Call

- Menu bar: *"Project"*
- Context menu in the *"Devices"* view of the CODESYSperspective
- Context menu in the *"Data Sources"* view of the *"HMI"* perspective

Requirement: The *"Data Source Manager"* object is selected that should have an additional data source.

Dialog 'Add Data Source'

"Name"	Example: `Data_Source_A`
"Select data source type"	Data source type that matches the controller configuration in order to establish communication. ● *"CODESYS Symbolic"*: Requirement: The local device is a CODESYS HMI. The data is transmitted by means of symbolic monitoring. This requires that symbols are configured in the symbol configuration of the remote PLC application. Note: As long as the symbol configuration is not impacted by an application change, you have the advantage that the application in the local device does not have to be updated. Hint: Use this communication connection unless there are no resources available in the remote PLC for a symbol configuration. ● *"CODESYS ApplicationV3"*: The data is transmitted via the CODESYS address protocol. This requires that the address information between the remote PLC and the local device match. Otherwise a connection cannot be established. Advantage: A symbol configuration is not required in the remote application. Note: For changes to the remote application, the local application has to be updated (for example, the HMI application). Hint: Use this communication for embedded or mini PLCs when there are no available resources for the symbol configuration. ● *"OPC UA Server"*: Data is transferred from an OPC UA server to the local controller via a TCP connection.
"Add"	Opens the *"Initialize Data Source - Provider settings"* dialog. The contents of the dialog depend on the selected data source type.

NOTICE!
The remote PLC should be running and the remote PLC application loaded and started.

See also

- *Chapter 1.4.1.20.2.4 "Object 'Data Source Manager'" on page 821*
- *Chapter 1.4.1.20.2.5.1 "Tab 'Variables'" on page 824*

Dialog 'Initialize Data Source Wizard - Provider settings' (for 'CODESYS Symbolic')

The settings of this dialog are described in the following chapter: Object 'Data Source' - Tab 'Communication".

The dialog is used to configure the connection initially when you have selected *"CODESYS Symbolic"* as the data source type. The communication is done by means of symbolic monitoring. The configuration can be modified later in the editor of the data source on the *"Communication"* tab.

See also

- *Chapter 1.4.1.20.2.5.3 "Tab 'Communication' via CODESYS Symbolic" on page 826*

Dialog 'Initialize Data Source Wizard - Provider settings' (for 'CODESYS ApplicationV3')

The settings of this dialog are described in the following chapter: Object 'Data Source' - Tab 'Communication".

The dialog is used to configure the connection initially when you have selected *“CODESYS ApplicationV3”* as the data source type. The communication is done by means of address monitoring.

See also

- *Chapter 1.4.1.20.2.5.4 “Tab 'Communication' via CODESYS ApplicationV3” on page 831*

Dialog 'Initialize Data Source Wizard - Provider settings' (for 'OPC UA Server')

The settings of this dialog are described in the following chapter: Object 'Data Source' - Tab 'Communication".

The dialog is used to configure the connection initially when you have selected *“OPC UA Server”* as the data source type. The communication takes place over a TCP connection.

See also

- *Chapter 1.4.1.20.2.5.5 “Tab 'Communication' via OPC UA Server” on page 834*

Dialog 'Initialize Data Source Wizard - Browse data items'

The settings of this dialog are described in the following chapter: Object 'Data Source' - Dialog 'Choose Variables'.

Function: You can select the variables for data transmission from the variables of the remote PLC. By clicking *“Finish”*, the data source is initialized and the data types and variables (data interface) are declared below the folder *“DataSources_Objects”*. You can modify the settings in the editor of the data source object.

Call: Automatic

See also

- *“Dialog 'Choose Variables'” on page 824*

Object 'Data Source'

Symbol: 

In the editor (object type *“Data Source”*), the access to the data of a remote device is managed on the *“Variables”*, *“Type Mappings”*, *“Communication”*, and *“General and Diagnosis”* tabs.

Status bar

The status bar which is always visible notifies you about the data source type and the most important communication settings. When the communication is established by means of the data source type CODESYS Symbolic, the name of the data source type, the connection type, and the network name of the remote device are displayed. When the communication is established by means of data source type CODESYS ApplicationV3 , then the name of the data source type, the location of the remote project, and the instance name of the remote application.

Example:

```
CODESYS Symbolic (CODESYS V3): PLC_Name
CODESYS ApplicationV3 (D:\Projects\Project_A): Project_A.App_A
```

See also

- *Chapter 1.4.1.9 “Working with Controller Networks” on page 352*

Tab 'Variables'

The variables for the data originating from the remote source are declared in the global variable list `<name of data source>`. The global variable list acts as a data interface to the remote PLC. The object is located below the application and below the *“DataSources_Objects”* folder.

“Update variables”	Establishes a connection to the remote device and opens the *“Choose Variables”* dialog.

“Local variable”	Variable in the local application. Contains the remote data.
“Access rights”	Access rights of the variables. The respective remote variable has the same access rights. ● : Write access. Every time the values changes, the variable is updated on the controller. ● : Read access. Every time the values changes on the controller, the variable is updated in the application. ● : Read/Write access Note: If you change the access rights, then a download is required for the change to go into effect.
“Update always”	: The controller data is updated automatically (via the data source). A variable is updated automatically if it is used in the visualization, trend, recipe, or as an alarm. Note: This is the recommended setting type. : The variable is updated in each cycle. Note: Select the option only when the variable is used exclusively in IEC code. If a variable is used in the visualization code, then it is updated automatically. Note: When an instance of a function block or a data type is updated in this way, the instance is always transferred completely.
“Create or map”	Mapping type for how the remote variable is mapped to the local variable. ● : Mapping to a specific created variable with the data type of the remote variable. The control data is mapped 1:1. That is the recommended mapping type. The variable is declared in the GVL `<name of data source>`. ● : Mapping to an existing variable. This requires that the existing variable has the same data type. ● : Mapping to a specific created variable with type-conforming data type to the remote data type: remote and local data types are not the same, but compatible. For example, a type-conforming data type can be available in a library. The variable is declared in the GVL `<name of data source>`.
“Type mapping”	Data type of the remote variable. If the variable is not a scalar type, then the type is listed on the *“Type Mappings”* tab.
“Remote variable”	Variable in the remote PLC

Dialog 'Choose Variables'

Symbol:

Function: The dialog lists the remote variables that are accessed by means of the configured connection.

Call: *“Update variables”* command on the *“Variables”* tab.

Requirement: The remote PLC is running. The control application is downloaded.

“Variables”	The remote variables are listed in the tree view. The top node is identified by the remote application name. Its variables are listed below that. Structured data is listed with all of its subordinate elements. Example: `appControl_A` ☑: The variable is selected for transferring to the local device. When the variable is structured, it is applied with all subelements. If the variables themselves are subelements, then only this subelement is applied without accepting the structure completely. Red font: When a variable is displayed in a red font, the variable is not available (anymore) in the remote PLC. Note: You can click *“Uncheck unavailable variables”* to remove the variable from the list. ☐: The variable is not selected for the transfer.
⊞	The variable has expandable elements. By clicking the symbol, the variable is extended by their elements.
“Insert the items structured”	☑: The selected variables are transferred with this structure if they are structured. ☐: The variable is transferred unstructured with a scalar data type.
“Uncheck unavailable variables”	Requirement: The link is visible when previously are no longer available in the variable available on the remote PLC. These variables are marked in red in the window above. The symbol configuration or the application presumably changed in the remote PLC. By clicking the command, the red variables are removed from the list box.

Tab 'Type Mappings'

The tab lists the non-scalar data types as they are currently available in the *“DataSources_Objects”* folder. You can edit or delete the data type declaration by selecting a data type and then the declared elements in the lower window. Moreover, you can modify the name, reset access rights, map another type, or select another remote variable.

Tab 'Type Mappings'

“Local type”	Data type in the local application
“Create or map”	• : Mapping to a new created data type. Declared in the *“DataSources_Objects”* folder. • : Mapping to an existing data type • : Mapping to a type-conforming data type. Declared in the *“DataSources_Objects”* folder.
“Mapping name”	Name of the data type
“Remote type”	Data type of the remote PLC

List with the subordinate elements of the selected data type.	
“Local variable”	Local variable name of the element of the selected data type
“Access right”	Access rights to the element

“Type mapping”	Data type of the element
“Remote variable”	Remote variable name of the element of the selected data type

[Del]	Removes the selected element

See also

- *Chapter 1.4.1.9.4.2 “Editing data source variables” on page 370*

Tab 'Communication' via CODESYS Symbolic

The tab includes the communication settings via CODESYS Symbolic for the remote data source.

When initially adding a data source, you have selected the *“CODESYS Symbolic”* data source type, and depending on that the communication settings to the data source were configured. Afterwards, the communication settings are outdated on this tab. You can only initially set the *“Data source type”* setting.

CODESYS Symbolic means that in the case of an active connection the communication is done via symbolic monitoring. This kind of symbolic access is possibly for CODESYS V2 and CODESYS V3 controller variants. In addition, the runtime system has to support the symbol configuration.

You can develop a local application offline based on the symbol information without a connection to the data source. To do this, you refer to a symbol file in the configuration settings in which all required variable information has been stored. Then no active connection is established.

Tab 'Communication' via CODESYS Symbolic

"Variable information"	Source of the variable information ● *"From connection settings"* A connection is established actively according to the communication settings specified below (*"Connection type"* and *"Connection Settings"*). The variable information is read from the remote controller application. ● *"<device name>.<application name>.symbol configuration"* The variable information is read from the symbol configuration. The symbol configuration is part of the active project and located in the device tree at the object of the remote controller below the application. An active connection is not established to the controller. ● *"From symbol file"* The variable information is read from a symbol configuration file that is stored on the development system. In the *"Choose symbol file"* field, specify this data. An active connection is not established to the controller.
"Choose symbol file"	The path of the symbol file for the *"Variable information"* selection is *"From symbol file"*. The symbol file is stored on the development system and contains the required variable information. By default, a symbol file path is created in the project directory in the following structure: `<project folder>\<project name>.<device name>.<application name>.xml`. Example: `D:\Projects\Project_A\VisualizeWithHMI.Device.Application.xml` Note: If the *"Alarm Table"* element or *"Trend"* element is used in the visualization, then the symbol file required for symbolic access and the respective project **must** both be saved in the same folder. The project contains the configuration for the alarm table element or the trend recording for the trend element. This is the default case for automatically generated symbol files. Example: `D:\Projects\Project_A\VisualizeWithHMI.project`

"Connection type"	Connection type between the remote PLC and the local device. Depending on the selected connection type, the following settings below change. Note: Whenever possible, avoid a direct connection without a gateway. ● *"CODESYS V2"* The devices exist in the same network. The V2 runtime on the remote PLC provides a communication interface. ● *"CODESYS V2 (Via gateway)"* The devices do not exist in the same network. They are connected via a V2 gateway. ● *"CODESYS V3"* The devices exist in the same network. The V3 runtime on the remote PLC provides a communication interface. ● *"CODESYS V3 (Via gateway)"* The devices do not exist in the same network. They are connected via a V3 gateway.

Connection settings for connection type 'CODESYS V2'

"PLC"	
"Driver type"	● *"Tcp/Ip (Level 2 Route)"* ● *"Tcp/Ip (Level 2)"* ● *"Tcp/Ip"*
"Address"	Example: `localhost` (for the currently used system on your computer)
"Port"	Example: `1200`
"Block size"	Example: `128` Requirement: The driver type is *"Tcp/Ip (Level 2)"*.
"Target ID"	Example: `0` Requirement: The driver type is *"Tcp/Ip (Level 2 Route)"*.
"Motorola byte order"	☑: Byte order on the PLC in big endian (Motorola format) ☐: Byte order in little endian (Intel format)

Connection settings for connection type 'CODESYS V2 (Via gateway)'

"Gateway"	The gateway settings are configured in addition to the PLC settings. Note: For this connection, a *"CoDeSys V2.3 Gateway Server"* (V2 Gateway) also has to be installed on the development computer where CODESYS V3 is running.
"IP address"	Example: `localhost`
"Port"	Example: `1217`

Connection settings for connection type 'CODESYS V3'

"PLC"	
"Name or address of device"	The setting that you make here varies according to the selection in the *"Type of name or address"* list box. For options that are derived automatically, you do not have to specify the setting here. The setting can remain empty. Example: Nothing specified for *"...(automatically derived)"* Example: `PLC_A` for *"Node name"* Example: `[ABCD]` for *"Node address"* Example: `192.168.1.5:11741` for *"IP address"* Example: `POU.dssCommVar` with data type `DatasourceSym.ConnectionSetup` for *"Dynamic from variable"* Hint: …: Opens the input to select the program variables for dynamic configuration. This variable has to be the data type `DatasourceSym.ConnectionSetup`.

"Type of name or address"	● *"Node name (automatically derived)"* ● *"Node address (automatically derived)"* ● *"IP address (automatically derived)"* ● *"Node name"* ● *"Node address"* ● *"IP address"* ● *"Dynamic from variable"*
"Dynamic from variable"	The device name or address is configured dynamically at runtime by means of an IEC variable of data type `DatasourceSym.ConnectionSetup`. The data type `DatasourceSym.ConnectionSetup` (`STRUCT`) is defined in the `Datasource Symbolic Access` library. For the configuration, the structure member `xDataValid` first has to be set to `FALSE`. If the address data has been specified, then `xDataValid` has to be set back to `TRUE`. Use case: The device name or address is not available when a project is being created. The dynamic configuration can also be used to change the settings at runtime without restarting the HMI application. Note: For this connection type, the connection is also not done dynamically via gateway.

Connection settings for connection type 'CODESYS V3 (Via gateway)'

"Gateway"	The gateway settings are configured in addition to the PLC settings.
"IP address"	Example: `localhost`
"Port"	Example: `1217`

Extending the communication settings for the PLCHandler interface

NOTICE!
It is not recommended to configure the PLCHandler manually.

The connection to the controller is established via the CODESYS PLCHandler communication interface. In this case, the configuration is performed in the PLCHandler INI format and allows for advanced parameterization.

"Advanced"	
"Used as"	● *"Don't use"* Recommended setting The *"INI content"* property as well as any specified configuration settings there are ignored. ● *"Extend the configuration by the following content"* As a rule, the configuration settings are used which are specified in the *"Connection Settings for CODESYS V3 (Via gateway)"* property. Moreover, the configuration settings are used in the *"INI content"* property. ● *"Configure completely with the following content"* The configuration settings of the *"Configuration Settings for CODESYS V3 (Via gateway)"* property are ignored. Instead, only the configuration settings are used in the *"INI content"* property.

"INI content"	Requirement: *"Used as"* is set to *"Extend the configuration by the following content"*.
	Example:
	`logfilter=16#000000FF`
	Example:
	`parameter0=EncryptCommunication`
	`value0=1`
	Note: If the parameters are generic, then they can be specified as 0-based (`parameter0` and `value0`). When extending, the numbering is automatically adjusted so that the extended parameters connect to the existing ones. The number of parameters (`parameters=<n>`) is also set to the correct value.

"INI content"	Requirement: *"Used as"* is set to *"Configure completely with the following content"*.
	Example:
	`[PLC:PLC_IdArti]`
	`interfacetype=ARTI`
	`active=1`
	`logevents=1`
	`motorola=0`
	`nologin=0`
	`timeout=10000`
	`precheckidentity=0`
	`tries=3`
	`waittime=12`
	`reconnecttime=10`
	`buffersize=0`
	`device=Tcp/Ip (Level 2 Route)`
	`instance=PLCWinNT_TCPIP_L2Route`
	`parameters=4`
	`parameter0=Address`
	`value0=localhost`
	`parameter1=Port`
	`value1=1200`
	`parameter2=TargetId`
	`value2=0`
	`parameter3=Motorola byteorder`
	`value3=No`

Communication settings for controllers with visualization user management

"Login Configuration"	If a visualization user management is configured on the remote device, then valid credentials are required at login.
"Type"	Defines how the visualization user management gets credentials ● *"Login using the following credentials"* The credentials are hard-coded into the *"User name"* and *"Password"* settings. They are used each time a connection is attempted. ● *"Login using the credentials determined at runtime"* At runtime, a dialog opens and prompts the user to specify a user name and password. Hard-coded credentials, which have nonetheless been specified in *"User name"* and *"Password"*, are ignored.

“User name”	Example: `max.smith`
“Password”	Example: •••

See also

- *Chapter 1.4.1.9.4.1 “Initially Adding a Data Source” on page 365*
- *Chapter 1.4.1.20.2.4 “Object 'Data Source Manager'” on page 821*

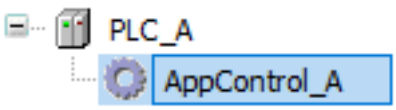
Tab 'Communication' via CODESYS ApplicationV3

The tab includes the communication settings for a remote data source.

When initially adding a data source, you have selected the CODESYS ApplicationV3 data source type, and depending on that the communication settings to the data source were configured. Afterwards, the communication settings are outdated on this tab. You can only initially set the *“Data source type”* and *“Select the project type”* settings.

CODESYS ApplicationV3 means that in the case of an active connection the communication is done via address monitoring. In this case, the remote PLC is configured by directly specifying the device address or automatically via network scan.

Tab 'Communication' via CODESYS ApplicationV3

“Select the project type”	The project type indicates where the controller is configured: in the same project as the HMI application or in a separate project. ● *“Current project”* The control application is part of the currently open project. The communication settings can be updated automatically or manually. ● *“Other Project”* The control application is part of a separate project whose location is specified in *“Choose file”*. The communication settings are done manually. In the initial setting of the data source object, this option is fixed and influences which settings are available for *“Target device”*.
“Choose file”	Name and path of the project that contains the control application (source project) Example: `D:\PLCs\PLC_A.project` Requirement: The *“Select the project type”* is *“Other Project”*.
Window area for controllers of the project	Controllers and their subordinate applications, read from the selected project Example: 

Settings for 'Select the project type' == 'Current project'

"Target device"	Note: The following settings are available when *"Select the project type"* is set to *"Current project"*.
"Automatic configuration"	: The configuration is read automatically from the source project. This is the recommended setting. Example: *"[DEVICE_A]"* Example: *"[undetermined]"*: No configuration can be read. Note: Make sure that the application is running on the controller and the network path is active. The communication settings of the controller are applied only then. These are the communication settings that were configured in the source project in the device editor on the *"Communication Settings"* tab.
"Manual configuration"	: More configuration settings are displayed. See "Manual configuration" below.

Settings for 'Select the project type' == 'Other Project'

The communication setting is done only manually.

See "Manual configuration" below.

Manual configuration

"Dynamic from variable"	: The communication parameters are configured at application runtime by means of an IEC variable of data type `DatasourceAppV3.ConnectionSetup`. ...: Opens the input for selecting the IEC variables for a dynamic configuration. The data type `DatasourceAppV3.ConnectionSetup` (`STRUCT`) is defined in the `Datasource ApplicationV3 Access` library. For the configuration, the structure member `xDataValid` first has to be set to `FALSE`. If the address data has been specified, then `xDataValid` has to be set back to `TRUE`. Use case: The communication parameters are not available yet when a project is being created.
"Use device address"	: The communication is done via the address specified here. Example: `0101` Hint: Click *"From device"* for an automatic address setting.
"From device"	The data of the currently connected data source device is read automatically and specified in *"Use device address"*. The address corresponds to the setting of the device in the device editor in *"Communication Settings"*.
"Search for the target device using the network scan"	: The data source manager starts the network scan for devices in the network. The scan is successfully when controllers are found whose communication settings match the selected search criteria. The result is displayed in the input fields.
"Node name"	: Search for the specified node name Example: `WST06`
"Target type"	: Search for the specified target type Example. `4096`
"Target ID"	: Search for the specified target ID Example: `0000 0001`
"Target version"	: Search for the specified target version Example: `1.0.0.0`

"Network location"	● *"Direct child of the data sources PLC"*: The scanned remote PLC has an address that is running with the address of the local controller (of the data source manager). Example: Data sources PLC: `0000.0001`; remote source PLC: `0000.0001.0001` ● *"Direct child of node with address"*: Specify the address of the parent node ● *"Direct child of the data source PLC or of the node with address"*: Combination of both options above.
"Search type"	● *"First found device"*: The first controller in the device tree is selected that fulfills the specified criteria. ● *"Exactly found device"*: The controller is selected that fulfills the specified criteria exactly. Note: The data source manager waits until the network scan is complete. This usually takes about 10 seconds.

Communication settings for controllers with visualization user management

"Login Configuration"	If a visualization user management is configured on the remote device, then valid credentials are required at login.
"Type"	Defines how the visualization user management gets credentials ● *"Login using the following credentials"* The credentials are hard-coded into the *"User name"* and *"Password"* settings. They are used each time a connection is attempted. ● *"Login using the credentials determined at runtime"* At runtime, a dialog opens and prompts the user to specify a user name and password. Hard-coded credentials, which have nonetheless been specified in *"User name"* and *"Password"*, are ignored.
"User name"	Example: `max.smith`
"Password"	Example: ●●●

Specific settings of the communication buffer

"Advanced"	☑: The subsequent settings are changed.
"Default communication buffer size"	Default setting: `50000`

See also

- *Chapter 1.4.1.9.4.1 "Initially Adding a Data Source" on page 365*
- *Chapter 1.4.1.20.2.4 "Object 'Data Source Manager'" on page 821*

Tab 'Communication' via OPC UA Server

“Server URI”	URI of the OPC UA Server Editable
“Information Model Source ” The information model defines how the data is structured and organized on the OPC UA Server.	
“ Online”	The client connects to the server and detects the existing variables and types. Requirement: There exists an unencrypted connection to the server.
“Offline”	The client reads the variables and types from the information model. A connection to the server is not required. The list box includes the OPC UA information models which are installed in the *“OPC UA Information Model Repository”*.
“Security”	
“Messages Security Mode”	Type of encryption ● *“None”*: No encryption and no signing Note: if you select this option, there can be no guarantee who receives the data. Therefore, *“None”* should be used exclusively in closed networks. ● *“Sign and Encrypt”*: The transferred data will be signed and encrypted. Signing makes sure that the data is not manipulated and the receiver is correct. ● *“Sign”*: The transferred data will be signed. Signing and encryption work only for certificates.
“Security Policy”	List box for the encryption method to be used: ● Basic256sha256 Requirement: Either *“Sign and Encrypt”* or *“Sign”* was selected for *“Messages Security Mode”*.

See also

- ↳ *Chapter 1.4.1.9.4.7 “Establishing an Encrypted Connection of a Data Source OPC UA Client to an OPC UA Server” on page 377*
- ↳ *Chapter 1.4.1.20.2.4 “Object 'Data Source Manager'” on page 821*

Tab 'General and Diagnosis'

The *“General and Diagnosis”* tab provides information about the status of the data source communication.

“Update Configuration”	
“Update rate (ms)”	Example: `200`

“Connection Information”	
“Connection status”	Example: `online`
“Error information”	Example: `OK`

See also

- ↳ *Chapter 1.4.1.9.4.4 “ Updating data interfaces” on page 373*

Object 'DUT'

Symbol:

- for a DUT without text list support
- for an enumeration data type with text list support

A DUT (Data Unit Type) declares a user-specific data type.

You can add this kind of object below the application or in the *“POUs”* view. When the object is created, the *“Add DUT”* dialog opens. There you select among the *“Structure”*, *“Enumeration”*, *“Alias”*, or *“Union”* data types.

Moreover, enumerations can have a text list stored to localize the enumeration values. Then the object also has a localization view.

Syntax

```
TYPE <identifier> : <data type declaration with optional
initialization>
END_TYPE
```

How the data type declaration has to be done syntactically depends in detail on the selected data type.

Examples

Declaration of a structure

```
TYPE S_POLYGONLINE :
STRUCT
    aiStart : ARRAY[1..2] OF INT := [-99, -99];
    aiPoint1 : ARRAY[1..2] OF INT;
    aiPoint2 : ARRAY[1..2] OF INT;
    aiPoint3 : ARRAY[1..2] OF INT;
    aiPoint4 : ARRAY[1..2] OF INT;
    aiEnd : ARRAY[1..2] OF INT := [99, 99];
END_STRUCT
END_TYPE
```

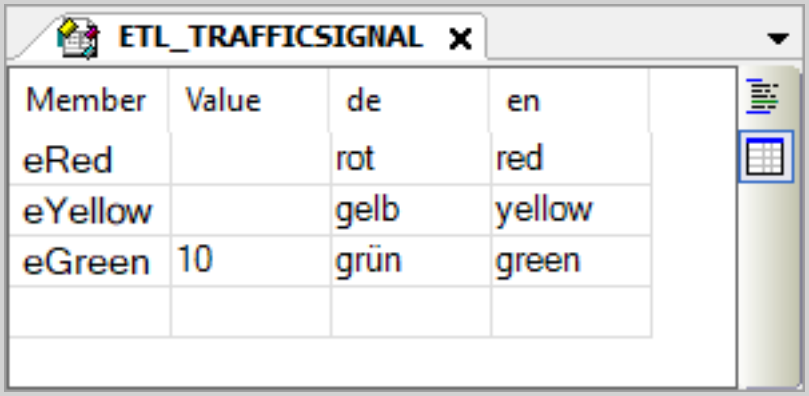
Extension of a structure

```
TYPE S_PENTAGON EXTENDS S_POLYGONLINE :
STRUCT
    aiPoint5     : ARRAY[1..2] OF INT;
END_STRUCT
END_TYPE
```

Declaration of an enumeration

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_TRAFFICSIGNAL :
(
    eRed,
    eYellow,
    eGreen := 10
);
END_TYPE
```

Enumeration with text list support in the localization view



The *"Textual View"* and *"Localization View"* buttons are located on the right edge of the editor. Click the buttons to toggle between the views.

Declaration of an alias

```
TYPE A_MESSAGE : STRING[50];
END_TYPE
```

Declaration of a union of components with different data types

```
TYPE U_DATA :
UNION
    lrA : LREAL;
    liA : LINT;
    dwA : DWORD;
END_UNION
END_TYPE
```

Dialog 'Add DUT'

Function: The dialog is used to configure a new DUT (Data Unit Type).

Call: Menu bar: *"Project ➔ Add Object ➔ DUT"*; context menu of the application object.

“Name”	Name of the new DUT data type Example: `S_POLYGONLINE`

Table 46: “Type”

“Structure”	Creates an object which declares a structure that combines multiple variables with different data types into a logical unit. The variables declared within the structure are called members. Example: `S_POLYGONLINE`
“Extends”	☑: Extends an existing structure by more members. In the input field, specify an existing structure. The members of the existing structure are automatically available in the new structure. Example: `S_PENTAGON`
“Enumeration”	Creates an object which declares an enumeration that combines multiple integer constants into a logical unit. The constants declared within an enumeration are also called enumeration values. Example: `E_TRAFFICSIGNAL`
“Add Text List Support”	☐: Enumeration that does not have any text list support ☑: Enumeration with additionally stored text list for the enumeration values. The text list allows you to localize the names of the enumeration values. Example: `ETL_TRAFFICSIGNAL` Note: In the case of an existing enumeration type, text list support can be added or removed at any time. As a result, the *“Add Text List Support”* and *“Remove Text List Support”* commands are provided in the context menu of the object. Hint: The localized texts can be displayed, for example, in a visualization. In this case, the text output of a visualization element displays the symbolic enumeration values in the current language instead of the numeric enumeration values. When an enumeration with text list support is specified in the *“Text variable”* property of a visualization element, it gets the additional property `< < enumeration name> >`. Example: In a visualization, you use the variable `PLC_PRG.eTrafficLight` of type `ETL_TRAFFICSIGNAL`. `ETL_TRAFFICSIGNAL` is an enumeration with text list support. Then the entry in the properties editor of the visualization element looks like this: `PLC_PRG.eTrafficLight <ETL_TRAFFICSIGNAL>`. Hint: When you edit the enumeration type in the application, a prompt opens when you close the application and asks whether the affected visualizations should be updated automatically. See also: Help for "Enumerations" with information about the declaration syntax
“Alias”	Creates an object which declares an alias with which an alternative name is declared for a base type, data type, or function block
“Union”	Creates an object which declares a union that combines multiple members with mostly different data types into a logical unit. All members have the same offset so that they are occupy the same memory. The memory requirement of a union is determined by the memory requirement of its "largest" component.
“Add”	Closes the dialog and creates the new object The object is displayed with the symbol in the device tree or in the *“POUs”* view. When a text list is also stored for the object, the symbol is displayed.

See also

- *Chapter 1.4.1.19.5.18 “Alias” on page 680*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*
- *Chapter 1.4.1.19.5.16 “Structure” on page 674*
- *Chapter 1.4.1.19.5.19 “Data type 'UNION'” on page 681*
- *Chapter 1.4.1.20.3.20.12 “Command 'Add Text List Support'” on page 1136*
- *Chapter 1.4.1.20.3.20.13 “Command 'Remove Text List Support'” on page 1136*
- *Help for CODESYS Visualization: Using Texts*

Object 'External File'

An *“External File”* is any file that you add to the project in the *“POUs”* view or *“Devices”* view. Click *“Project ➔ Add Object”* to open the *“Add External File”* dialog and define how the file belongs to the project.

An external file which was inserted in the *“POUs”* view is never downloaded to the controller.

An external file which was added in the *“Devices”* view is always downloaded to the controller when an online change or a download is performed due to an IEC code change.

When an external file is downloaded to the controller, it is not updated in the project.

Dialog 'Add External File'

“File path”	The ... button opens a dialog for selecting a file in the local file system.
“Name”	Object name for the file in CODESYS. If you do not type anything, the file will have its previous name.

Table 47: “File Handling”

“Remember the link”	The file is available in the project only as long as it exists in the defined file path.
“Remember the link and embed into project”	CODESYS saves an internal copy of the file in the project, as well as the link to the defined file path. The update option selected below applies as long as the external file exists there. Otherwise CODESYS uses the version saved in the project.
“Embed into project”	CODESYS saves only one copy of the file in the project. There is no longer a link to the external file.

Table 48: “Change Tracking”

“Reload the file automatically”	If the external file changes, then CODESYS updates the file in the project.
“Prompt whether to reload the file”	If the external file changes, then a dialog prompt opens whether CODESYS should also update the file in the project.
“ Do nothing”	The file remains unchanged in the project, even if the external file changes.

“Display File Properties”	Clicking this button opens the default *“Properties of <file name>”* dialog, which you can also open in the Windows file system by right-clicking the file.
“Open”	The file object is inserted into the device tree (*“Devices”* or *“POUs”* view) and opened in the editor for the matching file format.

See also

- *Chapter 1.4.1.20.3.4.1 “Command ‘Add Object’” on page 1001*
- *Chapter 1.4.1.20.4.10.7 “Dialog 'Properties' - 'External file'” on page 1161*

Object 'Device' and Generic Device Editor

Symbol: 

A device object represents a type of hardware; examples: control device, fieldbus node, bus node, drive, I/O module, monitor. The arrangement of the device objects in the device tree, that is the view *“Devices”* in CODESYS, maps the hardware structure. In the device object configuration editors inter alia you connect the controller I/Os with project variables.

Use command *“Add Device”* or *“Insert Device”* to insert a device object in the device tree. Depending on the insert position CODESYS always offers the currently matching devices.

A double-click on a device object in the device tree opens the associated device editor. The editor provides generic and device-specific tabs for the device configuration.

See also

- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*

Generic device editor

The generic device editor contains tabs for the configuration of a PLC device in CODESYS. Additionally there are device-specific tabs, so that the configuration editor consists of many different dialogs, depending on the device.

The editor opens after a double-click the device object in the device tree (*“Devices”* view).

You can make general settings for a device editor in the CODESYS *“Options”* in the *“Device Editor”* category. For example, you can show and hide the tabs of the generic device editor.

A device editor is given the name of the device. The following tabs of the generic device editor can be included:

- *“Communication”*: Configuration of the connection between the development system and a programmable device (PLC). Not available in the case of pure I/O devices.
- *“Applications”*: List of the applications on the controller.
- *“<device> Parameters”*: Display and configuration of device parameters.
- *“Files”*: Configuration of the file transfers between a host file system and the device.
- *“Log”*: Display of the PLC log file.
- *“PLC Settings”*: Configuration of the handling of the I/Os: which application, behavior in the stop state, updating, bus cycle options, etc.
- *“PLC Shell”*: Text-based control monitor for interrogating certain information from the controller.
- *“Users and Groups”*: User management with regard to the device at runtime.
- *“Access Rights”*: Rights for access to objects and files on the device.
- *“Symbol Rights”*: Access rights of individual user groups to symbols (symbol sets) on the device.
- *“Task List”*: Overview of all inputs and outputs, which are assigned to tasks – useful for troubleshooting.
- *“Status”*: Device-specific status and diagnostic messages.
- *“Information”*: General information about the device (name, vendor, version etc.)

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*

Tab 'Communication Settings'

On this tab of the generic device editor, you define the connection between CODESYS and the device on which your application(s) should run.

If you prefer the classic mode of display for the dialog, then select it in the CODESYS “Options” (“Device Editor” category).

You select a gateway and a target device from the list boxes. The possible selections depend on the entries in the *“Manage Gateways”* and *“Manage Favorite Devices”* dialogs (see the *“Gateway”* menu).

You can also specify the target directly with the IP address (example: "192.168.101.109"), device address (example: "[056D]"), or device name (example: "MyDevice"). After the device is entered, CODESYS searches for the device in the network of the gateway.

The option of searching by device name requires unique device names in the network.

The solid circle on the lower right corner of the gateway symbol provides information about the connection status:

- Red: CODESYS cannot establish the connection.
- Green: The connection is established.
- Black: The connection status is unknown.

Some communication protocols allow regular checking of the gateway so that the status cannot be displayed.

Clicking the solid circle of the target device starts a network scan for the device. This works only if the network is not already being scanned.

“Scan Network”	This button opens the *“Select Device”* dialog. It lists all configured gateways with the associated devices. You can select one target device from this list. If the name of the selected device is unique, then the name will be used in the connection settings. Otherwise, the unique device address is applied. For details about this dialog, see the description of the classic view below.
“Gateway ”	This menu includes the following commands: ● *“Add New Gateway”*: Opens the *“Gateway”* dialog for defining a new gateway channel. ● *“Manage Gateways”*: Opens the *“Manage Gateways”* dialog with an overview of all gateways. You can add or delete entries here or change their order. ● *“Configure Local Gateway”*: Opens the *“Gateway Configuration”* dialog. You can configure the block drivers for the local gateway.
“ Device”	This menu includes the following commands: ● *“Add Current Device to Favorites”*: Adds the currently set device to the list of favorite devices. ● *“Manage Favorite Devices”*: Opens the favorites dialog with a list of all preferred devices. In this dialog, you can add or delete entries or change their order. The top device is the default. ● *“Rename Active Device”*: Opens the *“Change Device Name”* dialog. ● *“Wink Current Device”*: Devices that support this function illuminate a flashing signal. ● *“Send Echo Service”*: CODESYS sends five echo services to the controller. These are used to test the network connection, similar to the ping function. The services are sent first without a payload and then with a payload. The scope of the payload depends on the communication buffer of the PLC. A message view opens with information about the average echo service delay and the scope of the sent payload. ● *“Store Communication Settings in Project”*: ☑: CODESYS saves the communication settings in the project for reuse on the same computer. Note: If you use the project on another computer, then you have to reset the active path. ☐: CODESYS saves the communication settings in the options of the local installation for reuse on the same computer. Note: When using CODESYS SVN, the option should be cleared in order to prevent blocking the device object. ● *“Confirmed Online Mode”*: ☑: For security reasons, CODESYS requires you to confirm the following when calling the following online commands: Force Values, Write Values, Multiple Loading, Release Force List, Single Cycle, Start, Stop. ● *“Filter Network Scans by Target ID”*: ☑: The display is limited on the devices that have the same target ID as the current device configured in the project. ● *“Encrypted Communication”*: ☑: The communication to this controller is encrypted. A certificate of the controller is required in order to log in to the controller. If the certificate is not available, then an error message opens prompting whether or not the certificate should be displayed and installed. If the *“Enforce encrypted communication”* option is selected as *“Security level”* in the *“Security Screen”* view, then the *“Encrypted Communication”* command is disabled here. ● *“Change Communication Policy”* Opens the *“Change Communication Policy”* dialog for changing the device setting for the encryption of communication.

Table 49: Dialog “Change Communication Policy”

If a new communication policy is selected in this dialog, then the configuration on the controller is changed.	
“Communication Settings”	
“Current policy”	Shows the currently selected policy for the encryption of communication
“New policy”	List box for the new policy for encryption ● *“No encryption”*: The controller does not support encrypted communication. ● *“Optional encryption”*: The controller supports encrypted and unencrypted communication. ● *“Enforced encryption”*: The controller supports encrypted communication only.
“Device User Management”	
“Current policy”	Shows the currently selected policy for user management
“New policy”	List box for the new policy for user management ● *“Optional user management”*: It is the responsibility of the user to enable user management on the device or leave the device unprotected. ● *“Enforced user management”*: The user management on the device is enabled and cannot be disabled by the user.
“Allow anonymous login”	☑: Specific registered components (for example, OPC UA) can connect to the controller without the providing any credentials. Even if anonymous access to the OPC UA is permitted, the created device user management for the controller remains active.

See also

- *Chapter 1.4.1.10.2 “Encrypting Communication, Changing Security Settings” on page 381*
- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*
- *Chapter 1.4.1.20.3.4.5 “Command 'Scan for Devices'” on page 1003*
- *Chapter 1.4.1.20.3.18.1 “Command 'Add New Gateway'” on page 1124*
- *Chapter 1.4.1.20.3.4.3 “Command 'Insert Device'” on page 1002*
- *Chapter 1.4.1.20.3.18.2 “Command 'Configure the Local Gateway'” on page 1125*

Communication Settings - Classic Mode

In the CODESYS options, you can activate the classic mode of the dialog (*“Tools ➔ Options”*, "Device Editor" category).

“Select the network path to the controller”	Gateway channel for the connection. Select the channel from the lower part of the view.

Table 50: "View displaying configured gateway channels and network devices"

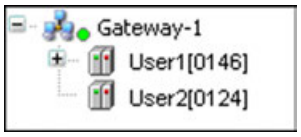
Left side of view	Tree structure of the configured gateway channels with the connected devices in the local network:  Note: CODESYS saves these entries on the local system, not in the project. The device entries are preceded by a device symbol ( ). Entries with a target ID that are different from those currently configured in the project are displayed in gray. Click *"Scan Network"* to refresh the list. Note: If you created the first project on the local system, then the local gateway is listed as an entry in the tree by default. CODESYS starts this gateway automatically on system boot. The solid circle on the lower right corner of the gateway symbol provides information about the connection status: ● Red: CODESYS Development System cannot establish the connection. ● Green: The connection is established:  ● Black: The connection status is unknown. Note: Some communication protocols allow regular checking of the gateway so that the status cannot be displayed. Each of the device entries in the tree consists of a symbol followed by the *"Device name"*> [*"Device address"*]. On the right side of the view, you also see the *"Target ID"*, *"Target Name"*, *"Target Type, Target Vendor"*, and *"Target Version"*.
Right side of view	Information about the gateway channel of device selected on the left side of the view. When a gateway channel is selected in the left view, the following information is displayed: *"Device name"*, *"IP address"*, *"Port"*, *"Driver"* When a device is selected in the left view, the following information is displayed (depending of the device): *"Device name"*, *"Device address"*, *"Number of channels"*, *"Block driver"*, *"Serial number"*, *"Encrypted communication"*, *"Target vendor"*, *"Target ID"*, *"Target name"*, *"Target type"*,*" Target version"*.

Table 51: "Filter and sorting functions on the right side of the dialog"

"Filter"	You can reduce the displayed list of devices that have the same *"Target ID"* as the current device configured in the project.
"Sorting order"	You can sort the list by *"Name"* or *"Device Address"* in alphabetical or ascending order.

Table 52: Command buttons on the right side of the dialog

"Set Active Path"	The command sets the selected communications channel as active. Double-clicking the entry in the channel tree achieves the same result.
"Add Gateway"	The command opens the *"Gateway"* dialog where you can define a gateway that CODESYS should add to the current configuration.
"Add Device"	The command opens the *"Add Device"* dialog. Here you can manually define a device that is to be inserted under the gateway entry currently selected in the tree. Note the functionality of *"Scan Network"* as well.
"Scan Network"	The command starts a search for available devices in the local network. The configuration tree of the gateway is refreshed accordingly.

Table 53: “Commands in the context menu of the gateway tree and device tree in the dialog”

“Scan for Device by Address”	The command searches the network for devices with a unique address as given in the configuration tree. CODESYS displays the detected devices with the given address below the gateway. The search always applies to the devices below the selected gateway or below the selected entry.
“Scan for Device by Name”	The command searches the network for devices with the same name as given in the configuration tree. Capitalization is ignored. CODESYS displays the detected devices below the gateway with the given name together with its unique device address. The search always applies to the devices below the selected gateway or below the selected entry.
“Scan for Device by IP Address”	The command searches the network for devices with a unique IP address as given in the configuration tree. CODESYS displays the detected devices with the given address below the gateway together with its name. The search always applies to the devices below the selected gateway or below the selected entry.
“Send Echo Service”	CODESYS sends five echo services to the controller. These are used to test the network connection, similar to the ping function. The services are sent first without a payload and then with a payload. The scope of the payload depends on the communication buffer of the PLC. A message view opens with information about the average echo service delay and the scope of the sent payload.
“Delete Selected Device”	The command deletes the selected device from the channel tree.
“Edit Gateway”	The command opens the *“Gateway”* dialog for editing the settings for the selected gateway.
“Configure the Local Gateway”	The command opens a dialog for configuring a local gateway. This provides an alternative to manually editing the `Gateway.cfg` file.

Table 54: Options in the lower part of the dialog

“Don't store communication settings in project”	• ☑: CODESYS saves the communication settings in the options of the local installation for reuse on the same computer. Note: When using CODESYS SVN, the option should be selected in order to prevent blocking the device object. • ☐: CODESYS saves the communication settings in the project for reuse on the same computer. Note: If you use the project on another computer, then you have to reset the active path.
“Confirmed Online Mode”	☑: For security reasons, CODESYS requires you to confirm the following when calling the following online commands: *“Force Values”*, *“Write Values”*, *“Multiple Loading”*, *“Release Force List”*, *“Single Cycle”*, *“Start”*, *“Stop”*.

See also

- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*
- *Chapter 1.4.1.20.3.4.5 “Command 'Scan for Devices'” on page 1003*
- *Chapter 1.4.1.20.3.18.1 “Command 'Add New Gateway'” on page 1124*
- *Chapter 1.4.1.20.3.4.3 “Command 'Insert Device'” on page 1002*
- *Chapter 1.4.1.20.3.18.2 “Command 'Configure the Local Gateway'” on page 1125*

Tab 'Parameters'

This dialog is intended for test purposes. Its values should be changed only by experts.

The device-specific parameters are displayed in a table on this tab of the generic device editor. The device description defines which parameters you can edit in this dialog.

You can sort the entries in alphabetically ascending or descending order or in the default order by clicking the column header.

“Parameter”	Parameter name, not editable
“Type”	Data type of the parameter, not editable
“ Value”	Initially displays the default value of the parameter, directly or the corresponding symbol name. Non-editable parameters are displayed in light-gray. If the parameter is editable you can open an input field, a drop-down list or a file selection dialog with a double-click the table field and use it to change the value.
“Default value”	Default value of the parameter defined by the device description, not editable
“Unit”	Unit of measure for the value (example: "ms" for milliseconds; not editable)
“Description”	Short description of the parameter specified by the device description, not editable

See also

- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*

Tab 'Applications'

On this tab of the generic device editor you can see which applications exist on the device. Depending on the system you can delete the applications from the device or retrieve detailed information about the application.

“Applications on the PLC”	List of the applications found via *“Refresh list”* during the last scan of the control device.
“Delete” *“Delete All”*	Deletes the application selected in the list or all listed applications on the controller Note: If a safety controller is inserted below a PLC, then this command can **permanently** interrupt the communication links of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system. The safe field devices and the other safety controller can enter the safe state as a reaction. The connection to the development system is affected only in the case of a safety controller that is connected to the main controller via a fieldbus. For more information, refer to the section "Subordinate Safety Controller".
“Details”	Opens the dialog box *“Details”*. It displays information defined for the application on the *“Information”* tab of the dialog box *“Properties”*.
“ Contents”	Requirement: The *“Download the application info”* option is activated in the *“Properties”* of the application object on the *“Application generation options”* tab. This causes information about the contents of the application to be additionally loaded to the PLC. The *“Contents”* button opens a dialog box with additional information about the differences between the latest generated code and the application code that exists on the controller. The different modules are displayed in a comparison view.
“Refresh List”	The controller is scanned for applications and the list is refreshed accordingly

You can configure the commands “Remove Application from Device” and “Remove Applications from Device” by means of the dialog box form “Tools ➔ Customize”. These commands correspond to the “Delete” and “Delete All” buttons.

See also

- *Chapter 1.4.1.20.2.8 “Object 'Device' and Generic Device Editor” on page 839*
- *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*
- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*
- *Chapter 1.4.1.20.2.8.19 “Tab 'Information'” on page 870*
- *Chapter 1.4.1.20.4.10.9 “Dialog 'Properties - Application Build Options'” on page 1162*
- *Chapter 1.4.1.20.3.6.2 “Command 'Login'” on page 1028*
- *Chapter 1.4.1.9.5 “Subordinate safety controller” on page 378*

Tab 'Backup and Restore'

In this tabbed page of the generic device editor, you can backup and restore the application-specific file on the PLC by saving and reading a zip archive.

Requirement: The communication settings are correct for connection to the device. The application for backup is available on the PLC.

Table 55: Menu Bar

“Backup”	This button opens a menu with the following commands: ● *“Read Backup Information from Device”*: This command searches for application-specific files from the `$PlcLogic$` directory of the PLC and lists them in a table in the lower part of the tabbed page. ● *“Create Backup File and Save to Disk”*: Requirement: The *“Read Backup Information from Device”* command was used for determining the backup-related files. These files are located in the table in the lower part of the tabbed page. This command compresses the files in the table set as *“Active”* and the `meta.info` information file into a backup zip file. The file extension is `tbf` (="Target Backup File"). ● *“Save Backup File to Device”*: Requirement: The backup file has been saved to the disk. This command saves the backup file to the `TBF` directory of the PLC.
“Restore”	This button opens a menu with the following commands: ● *“Load Backup File from Disk”*: This command opens the *“Open”* dialog box for navigating the file system for a saved backup file. The included files are listed in a table in the lower part of the tabbed page. ● *“Load Backup File from Device”*: This command generates a list of all backup files found on the PLC. Select one of these files to view its contents in a table on the tabbed page. For the restore operation, you can deactivate optional components and edit the comments. ● *“Restore on Device”*: This command is available if at least one component of the backup file that is currently loaded in the tabbed page is set to active. It prompts for restoring the application status on the device. The user interface is blocked during restore. You can cancel the operation.

Table 56: “Target Information”

“ID”	ID of the PLC (example: `0000 0001`)
“Type”	Device type (example: `4096`)
“Version”	Device version (example: `3.5.8.0`)

Table 57: “Backup Information”

“File name”	Storage path of the backup file. Clicking the symbol () opens the file system dialog box. Example: `PlcLogic$/Application/Application.crc`
“Size of active files”	(in kilobytes) Total size of the files set as active in the table (example: `206 KB (210965 bytes)`).
“Mode”	Defines the scope of the backup: *“Application”*. The application-related files are added to the archive.
“Comment”	Optional entry for comments to be saved in the `meta.info` file of the backup and reading when the files are restored.

Table 58: Table of files for backup

“Active”	: Optional files can be deactivated here for exclusion in the backup file. Required components are shown here with a green check mark (no check box).
“Component”	Affected components (example: `file system`)
“File”	Name of the component file to back up (example: `$PlcLogic$/Application/Application.app`)
“Size”	File size in bytes (example: `43280`)
“Requires STOP”	 : For components, the application must be stopped before backup and restore. A dialog prompt will open to warn you of any backup or restore conflicts.

See also

- *Chapter 1.4.1.12.9 “Backup and restore” on page 438*

Tab 'Synchronized Files'

The tab of the generic device editor lists the files that are downloaded to the PLC when the application is downloaded. For example, these are external files that were added to an application.

Implicit files, such as the source code archive file, are displayed here only if their time of download is configured for this and the *“Show implicit files for application download on the editor of a PLC”* option is activated in the CODESYS options (*“Device Editor”* category).

“Refresh”	Refreshes the view
“Download 'on-demand' files”	For internal use only.
“File Name”	Name of the file below the application, or direct name of the implicitly transferred file (example: `archive.prj`). Double-click the file name to open the file.
“Host Path”	Location or original location of the file (example: `D:\Proj1\Files`). Double-click the path to open the directory in the file explorer.
“Timing”	Time interval of the file update on the PLC (example: *“After application download/online change”*).
“Information”	Object-dependent additional information (example: *“Object: External File”*).
“Provider”	General origin type of the file (example: *“External File Objects”*, *“Source code download provider”*).

See also

- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*
- *Chapter 1.4.1.20.2.7 “Object 'External File'” on page 838*

Tab 'Files'

In this tab of the generic device editor, you can transfer files between CODESYS (host) and the PLC. If the communication settings are correct and the PLC is online, then CODESYS establishes the connection automatically to the PLC for the duration of the file transfer.

Table 59: “Host” / “Runtime”

	Access to the file system of the host with the functionalities of a standard file manager
“Location”	Current directory for the file transfer on the host side
	Opens a dialog to create a new directory in the set path
 “”	Deletes the selected files or directories
	Updates the list of files and directories there for the set location
	Copies the selected files and directories to the respective other file system from the host and runtime system If a file is not already available in the target directory, then it is created. If it is already available and not write-protected, then it is overwritten. Then a corresponding message is displayed.  corresponds to the *“Write File to Controller”* command.  corresponds to the *“Write File from Controller”* command.

By default, the “Write File to Controller” and “Write File from Controller” commands are not included in any menu. You can add it to a menu by means of the “Tools ➔ Customize” dialog, in the “Online” command category.

See also

- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*
- *Chapter 1.4.1.14 “Copying files to/from PLC” on page 441*

Tab 'Log'

You can view the PLC log on this tab of the generic device editor. It lists the events that were recorded on the target system. This concerns the following:

- Events during the startup and shutdown of the system (components loaded, with version)
- Application download and loading of the boot application
- Custom entries
- Log entries from I/O drivers
- Log entries from data sources

The “Log” tab also opens when you click “Open Log Page”. You can configure this as a menu command in the “Customize” dialog.

Table 60: Menu bar

	Refreshes the list of log events for all runtime system component
"Components"	Filters the display of log events by the runtime system components selected in the list box Example: `CmpApp` displays all events which occur in these components, for example "`Application [ <name> ] loaded via [OnlineChange]`". *"<all components>"*: Displays the reported events of all components
	Loads the next page with newer log messages
	Loads the previous page with older log messages
	Loads the page with the newest log entries and enables automatic scrolling
	 : Indicates that there are new log messages which have not been displayed yet. Hint: This is also displayed on the status bar as *"Auto-Scroll: ON"*.
	Loads the page with oldest log messages
	Filters events with the severity *"Warning"* and notifies about how many Blue-outlined button: Warnings are displayed.
	Filters events with the severity *"Error"* and notifies about how many Blue-outlined button: Errors are displayed.
	Filters events with the severity *"Exception"* and notifies about how many Blue-outlined button: Exceptions are displayed.
	Filters events with the severity *"Information"* and notifies about how many Blue-outlined button: Information is displayed.
	Filters events with the severity *"Debug"* and notifies about how many Blue-outlined button: Debug messages are displayed.
Logger	Enables a logger for displaying its recorded events By default, the `<default logger>` defined by the system is set. For example, that is the logger `PlcLog` for a CODESYS Control Win V3 runtime system.
"UTC time"	 : Converts the times displayed below *"Timestamp"* to the local time of the development system. The conversion is based on the time zone of the operating system where the CODESYS is running. (default setting)  : Displays the original time stamp of the runtime system If you change the option, then the displayed time stamp is converted automatically.
	Exports the list contents to an xml file. You can select the file name and location.
	Imports an XML file with log messages stored in the file system . A separate window opens to display the log messages.

Table 61: Display window with log file

Tabular display of the log messages Ten thousand log messages are displayed per page.	
"Severity"	• : Warning • : Error • : Exception • : Information • : Debug message
"Time Stamp"	Date and time of the development system or of the runtime system) Example: `01/12/07 09:48`
"Description"	Description of the event Example: `PLC started`
"Component"	Runtime component where the reported event occurred

Table 62: Status bar

"Auto-Scroll"	Displays whether auto-scrolling is enabled (*"ON"*) or disabled (*"OFF"*) Hint: Click the button to enable *"Auto-Scroll"*. • *"ON"*: The log list is refreshed automatically when changes occur. • *"OFF"*: When a new log event occurs, it is displayed next to *"Off"*. Moreover, the button is decorated on the menu bar: .

Note for error checking

For exceptions with the description `*SOURCEPOSITION*`, the affected function opens in the editor by double-clicking it or by means of the "Display Source Code in Editor" command in the context menu. The cursor jumps to the line that is causing the error. You can also perform this diagnosis when you have the CODESYS project archive, including the download information files and the exported log file. When the affected function is protected, then the following message appears: "`The source code is not available for <function name>`".

If a `VendorException` is reported, then a manufacturer-specific exception error has occurred in the CODESYS runtime. Contact the PLC manufacturer for more information.

See also

- *Chapter 1.4.1.20.2.8.1 "Generic device editor" on page 839*
- *Chapter 1.4.1.20.2.8 "Object 'Device' and Generic Device Editor" on page 839*
- *Chapter 1.4.1.12.6 "Reading the PLC log" on page 435*
- *Chapter 1.4.1.20.4.14.1 "Dialog 'Customize' - 'Menu'" on page 1206*

Tab 'PLC Settings'

On this tab of the generic device editor you make the basic settings for the configuration of the PLC, for example the handling of inputs and outputs and the bus cycle task.

“Application for I/O handling”	Application that is responsible for the I/O handling.

Table 63: “PLC Settings”

“Update IO while in stop”	☑: CODESYS refreshes the values of the input and output channels even if the PLC is in the stop state. If the watchdog detects a malfunction, the outputs are set to the predefined default values. ☐: CODESYS does not refresh the values of the input and output channels when the PLC is in the stop state.
“Behavior for outputs in stop”	Handling of the output channels when the controller enters the stop state: ● *“Keep current values”*: The current values are retained. ● *“Set all outputs to default”*: The default values resulting from the I/O mapping are assigned. ● *“Execute program”*: You can control the handling of the output values via a program contained in the project, which CODESYS executes at "STOP". Enter the name of the program in the field on the right.
“Always update variables”	Global setting that defines whether or not CODESYS updates the I/O variables in the bus cycle task. This setting is effective for I/O variables of the slaves and modules only if 'deactivated' is defined in their update settings. ● *“Disabled (update only if used in a task)”*: CODESYS updates the I/O variables only if they are used in a task. ● *“Enabled 1 (use bus cycle task if not used in any task)”*: CODESYS updates the I/O variables in the bus cycle task if they are not used in any other task. ● *“Enabled 2 (always in bus cycle task)”*: CODESYS updates all variables in each cycle of the bus cycle task, regardless of whether they are used and whether they are mapped to an input or output channel.

Table 64: “Bus Cycle Options”

“Bus cycle task”	Task that controls the bus cycle. By default the task defined by the device description is entered. By default the bus cycle setting of the superordinate bus device (use cycle settings of the superordinate bus) applies, i.e. the device tree is scanned upwards for the next valid bus cycle task definition. Pay strict attention to the following notes!

NOTICE!
Before you select the *“<unspecified>”* setting for the bus cycle task, you should be aware that "<unspecified>" means that the default setting given in the device description goes into effects. You should therefore check this description. Use of the task with the shortest cycle time may be defined as the default there, but use of the task with the longest cycle time could equally well be defined!

NOTICE!
For fieldbuses, a fixed cycle matrix is necessary to assure a determined behavior. Therefore, do not use the type 'free-running' for a bus cycle task.

Table 65: “Additional Settings”

“Generate force variables for IO mapping ”	This setting is available only if it is supported by the device. ☑: When compiling the application CODESYS creates two global variables for each I/O channel that is mapped to a variable in the dialog *“I/O Mapping”*. You can use these variables for the forcing of the input or output value on this channel, for example via an HMI visualization.
“Enable Diagnosis for devices”	☑: CODESYS automatically integrates the library `CAA Device Diagnosis` in the project and creates an implicit function block for each device. If there is already a function block for the device, then either an extended FB is used (for example with EtherCAT) or a further FB instance is added. This then contains a general implementation of the device diagnostics. By means of the FB instances you can determine the status of all devices in the application and evaluate errors. In addition, the library contains functions for the programmatic editing of the device tree. Example: Scanning of all children of a bus system, jumping to the parent element.
“Create additional parameters”	This setting is available only if it is supported by the device. Create additional parameters.
“ Show I/O warnings as errors ”	Warnings concerning the I/O configuration are displayed as errors.

See also

- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*
- *Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*
- *Chapter 1.4.1.20.3.5.4 “Command 'Build'” on page 1022*
- PDF document 'CAA Device Diagnosis', which is a component of the library.

Tab 'PLC Shell'

This tab of the generic device editor includes a text-based control monitor for querying specific information from the controller. You can specify device-dependent commands for this and receive the response from the controller in a result window.

Table 66: ABB AG standard commands

Command with Possible Parameters	Description
?	List of available PLC shell commands with possible parameters and short description
`getcmdlist`	List of names of available PLC shell commands
`mem <memory address> [<size>]`	Provides a hex dump of the defined memory range. The size parameter is optional and describes the number of bytes that are output. Default value: 16 Example: `mem 16x0422139C 8`
`reflect`	Repeats the given command (for testing the connection)
`applist`	Provides a list of all loaded applications The order in the list defines the application index beginning with `0`.
`pid [<application name>| <application index>]` *	Provides the GUID (application index) of one or all loaded applications
`pinf [<application name>| <application index>]` *	Provides the contents of the following fields from the project information: title, version, author, and description. Requirement: The option *“Create POU for properties access automatically”* in the *“Project Information”* dialog is activated.

Command with Possible Parameters	Description
`startprg [<application name>|<application index>]` *	Starts the given application, or all loaded applications if no application is given
`stopprg [<application name>|<application index>]` *	Stops the given application, or all loaded applications if no application is given
`resetprg [<application name>|<application index>]` *	Resets the given application, or all loaded applications if no application is given
`resetprgcold [<application name>|<application index>]` *	Executes a cold boot of the given application, or all loaded applications if no application is given
`reload[<application name>|<application index>]` *	Loads the boot application of the given application, or the boot projects of all loaded applications if no application is given
`getprgstat [<application name>|<application index>]` *	Provides the program status of the given application, or the program status of all loaded applications if no application is given
`plcload`	Shows the processor load of the controller (in percent)
`rtsinfo`	Provides information about the runtime system, for example the processor and version of the runtime system
`channelinfo`	Provides information about the communication channel
`rtc-get`	Provides the universal time (UTC) via the `DataTime` string
`rtc-set`	Sets the universal time (UTC) via the `DataTime` string (see ISO 8601)
`listpcicards [<VendorID>]`	Provides a list of PCI adapters (all or by <VendorID>)
`gettaskgroups`	Provides a list of all task groups, their tasks, and the CPU core binding
`cert-getapplist`	Provides all registered and used certificates (ID of the component and usage).
`cert-genselfsigned [<number for search result by "cert-getapplist"> <expdays=>]`	Generates self-signed certificates The validity period of the certificate can be specified by means of `expdays=`. Default value: 365 days
`cert-gendhparams [length in bits]`	Generates the parameters for the Diffie-Hellman key exchange Caution: This operation can take several minutes to complete.
`cert-getcertlist [<trust level>]`	Lists all certificates of the specified trust level If a trust level is not given, then all certificates are listed. Possible trust levels • `untrusted`: Untrustworthy certificates • `trusted`: Trustworthy certificates • `own`: Certificate of the controller • `quarantine`: Certificates whose trust level (trusted, untrusted) cannot be determined by validation. Incoming connections were therefore denied.
`cert-createcsr [<number for search result by "cert-getapplist">]`	Generates CSR files for all applications
`cert-import <trust level> <file name.cer>`	Imports the specified certificate
`cert-export <trust level> [<number for search result by "cert-getcertlist">]`	Exports the specified certificate

Command with Possible Parameters	Description
`cert-remove <trust level> <number for search result by "cert-getcertlist" or "all">`	Removes the specified certificate
`cpuload`	Shows the processor load of the CPU (for multicore, each processor core)
`gettaskgroups`	Provides a list of defined task groups The assigned tasks are shown for each task group.
`getmulticoreinfo`	Shows whether or not multicore is supported and the number of available processor cores
`sessinfo-list`	Provides a list of all currently logged in clients/users
`sessinfo-getcnt`	Provides the number of currently logged in clients/users

* Application name: Name of the application in the device tree

Application index: Results from the list of all applications on the controller that you can call with the *"applist"* command. Index `0` stands for the first application in the list, `1` for the second, and so on.

See also

- *Chapter 1.4.1.12.7 "Using PLC shell for requesting information" on page 436*
- *Chapter 1.4.1.20.2.8.1 "Generic device editor" on page 839*
- *Chapter 1.4.1.20.3.4.13 "Command 'Project information'" on page 1007*

Tab '<device name> I/O Mapping'

Devices with I/O channels

This tab is displayed in device editors for devices with I/O channels. It shows the available channels and allows for the mapping of input, output, and memory addresses of the controller to variables or entire function blocks of the application. In this way, you create the 'I/O Mapping'.

The application that is to take care of the I/O handling is defined on the *"PLC Settings"* tab.

You can use the "Online Configuration Mode" if the device supports it. In this mode, you can access the I/Os of the hardware without having to download an actual application to the device beforehand.

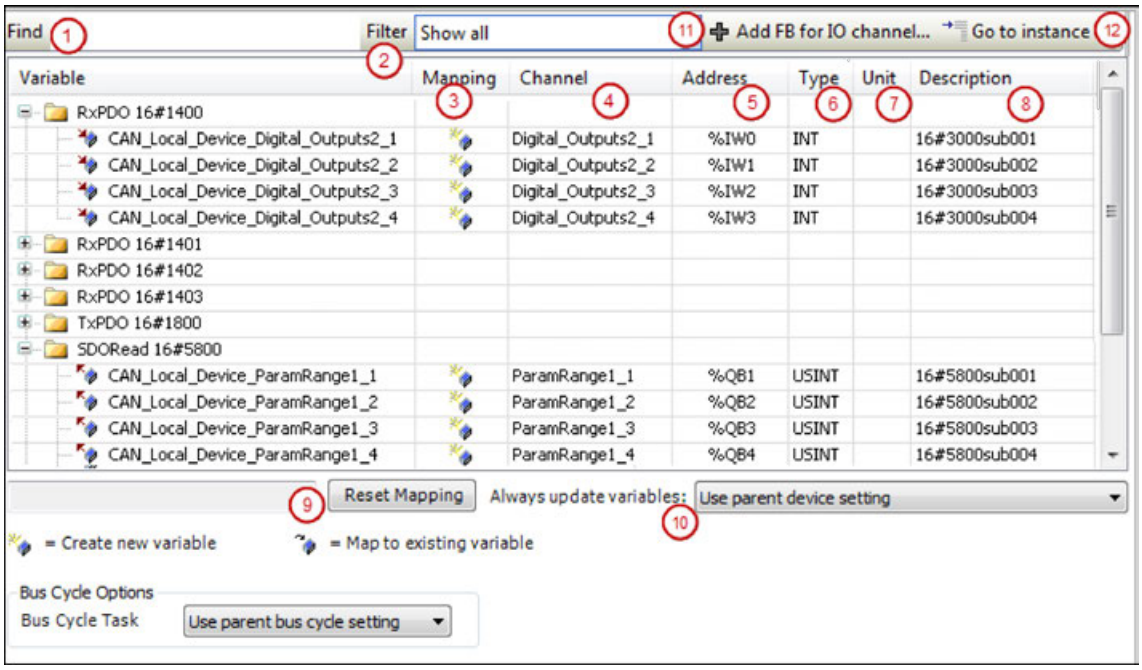
You can also create the I/O mapping in the "Edit I/O Mapping" dialog. Here you get a mapping list with search and filter functions for an entire device tree.

NOTICE!

Mapping 'too large' data types

If a variable of a data type that is larger than a byte is mapped to a byte address, the value of the variable will be truncated to byte size there. For monitoring the variable value in the *"I/O Mapping"* dialog, this means that, in the root element of the address, the value is displayed which the variable currently has in the project. The current individual bit values of the byte are displayed in succession in the bit elements below that, but this may not be sufficient for the entire variable value.

Example of the *"<device name> I/O Mapping"* tab for a CAN bus slave:



The tab contains a table for editing the I/O mapping. The information displayed for the inputs and outputs originates from the device description.

"Find" (1)	Input field for a search string for the mapping table. The search results are marked in yellow.
"Filter" (2)	List box with filters for the I/O mappings displayed in the mapping table: ● *"Show all"* ● *"Show only outputs"* ● *"Show only inputs"* ● *"Show only unmapped variables"* ● *"Show only mapped variables"* ● *"Show only mapping to existing variables"* ● *" Show only mapping to new variables"*
"Add FB for IO channel" (11)	Depending on the device, available if the channel entry is selected in the mapping table. Opens the *"Select Function Block"* dialog for selecting the function block that should be linked directly to the channel.
"Go to instance" (12)	Available if the entry is selected in the mapping table. Jumps to the corresponding entry on the *"<device name> IEC Objects"* tab.

"Variable"	Depending on the device, the inputs and outputs of the device are displayed as nodes and below them, indented, the associated channels or, depending on the device, only the implicitly created device instance. The symbol indicates the type of channel: : Input : Output Double-click the cell to open an input field. ● Option 1: The variable already exists; specify complete path: <application name>.<module name>.<variable name>; example: `app1.plc_prg.ivar`; input assistance via ... ● Option 2: The variable does not exist yet; enter a simple name; automatically created internally as a global variable. Depending on the device, inputs or outputs can be linked directly to a function block. In this case, the *"Add FB for IO channel"* button can be clicked. See above.
"Mapping" (3)	Type of mapping: ● : Existing variable ● : New variable ● : Mapping to function block instance
" Channel" (4)	Symbolic name of the channel.
"Address" (5)	Address of the channel (example: `%IW0`). Address strikethrough: Indicates that you should not assign any more variables to this address. Reason: Although the variable specified here is managed – as an existing variable –at a different memory location, ambiguity could result when the values are written, particularly with outputs. : Indicates that this address has been edited and fixed. If the arrangement of the device objects in the device tree changes, then CODESYS does not adapt this address automatically.
"Type" (6)	Data type of the channel (example: `BOOL`). Structures or bit fields defined in the device description are displayed only if they are part of the IEC standard and are identified as IEC data types in the device description. Otherwise the table cell remains empty. When mapping structured variables, the editor prevents you from specifying both the structure variable (example: `%QB0`) and individual structure elements (example: `%QB0.1` and `QB0.2`). Therefore, if there is a main entry with a subtree of bit channel entries in the mapping table, then the following applies: You can input a variable either into the line of the main entry, or into the lines of the subelements (bit channels), but not into both.
"Default value"	Default value of the parameter that applies to the channel: Appears only if the option *"Set all outputs to default"* is selected in the *"PLC Settings"* for the behavior of the outputs at stop. Note: For compiler version V3.5 SP11 and higher, the initialization value of the variables is used automatically as the default value when mapping to an existing variable. You can edit the *"Default value"* field only if you map to a new created variable or if no mapping is specified. In older versions, users had to specify explicitly that the default value and initialization value were identical.
"Unit" (7)	Unit for the parameter value (example: `ms` for milliseconds).
"Description" (8)	Short description of the parameter.
"Current value"	Actual value of the parameter applied to the channel; displayed in online mode only.

The change of the default value by an online change is allowed, however the value is applied only after a "Reset cold" or "Reset warm".

"Reset Mapping" (9)	CODESYS resets the mapping settings to the default values as defined in the device description file.
"Always update variables" (10)	Definition for the device object about updating I/O variables. The default value is defined in the device description: ● *"Use parent device setting"*: Update according to the setting of the superordinate device. ● *"Enabled 1 (use bus cycle task if not used in any task)"*: CODESYS updates the I/O variables in the bus cycle task if they are not used in any other task. ● *"Enabled 2 (always in bus cycle task)"*: CODESYS updates all variables in each cycle of the bus cycle task, regardless of whether they are used and whether they are mapped to an input or output channel.

If a UNION is represented by I/O channels in the mapping dialog, it depends on the device whether mapping to the root element is also possible.

Devices with I/O drivers

For devices with I/O drivers, you can set the bus cycle task here in the *"I/O Mapping"* tab if the general settings should not be used (*"PLC Settings"* tab).

Table 67: Bus Cycle Options

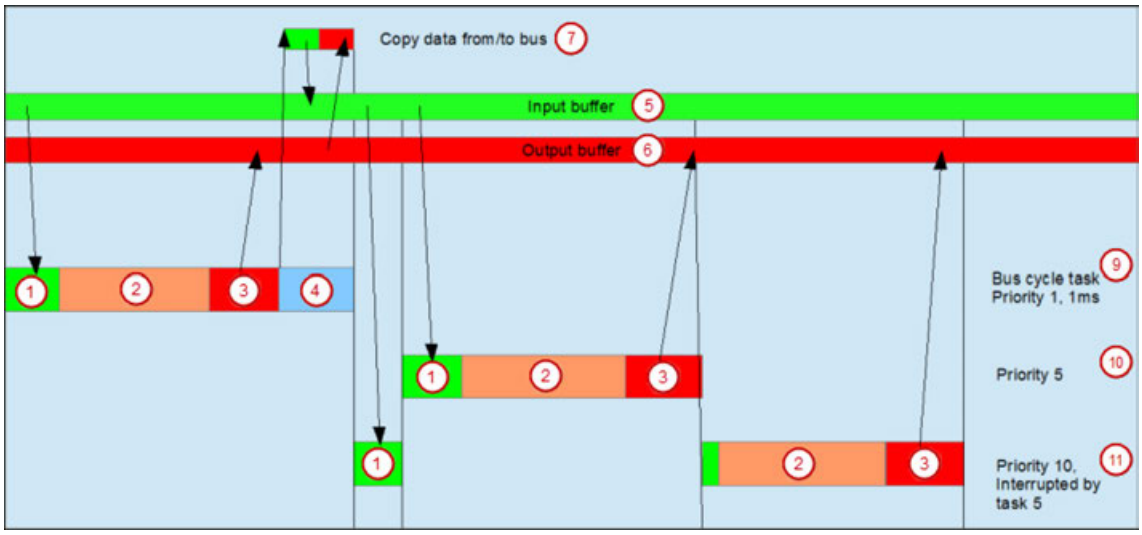
"Bus Cycle Task "	The list box provides all tasks which are defined in the task configuration of the active application (example: *"MainTask"*. In case of *"Use parent bus cycle setting"*, the settings of the parent node will be used.

General information about the bus cycle task

Generally, for each IEC task, the used input data is read at the start of each task (1) and the written output data is transferred to the I/O driver at the end of the task (3). The implementation in the I/O driver is decisive for additional transfer of the I/O data. It is responsible for the time frame and time point that the actual transfer to the corresponding bus system occurs.

The bus cycle task of the PLC can be defined globally for all fieldbuses in the PLC settings. For some fieldbuses, however, you can change this independent of the global setting. The task with the shortest cycle time is used as the bus cycle task (setting: *"unspecified"* in the PLC settings). The messages are normally sent on the bus in this task.

Other tasks copy only the I/O data from an internal buffer that is exchanged only with the physical hardware in the bus cycle task.



(1) Read inputs from input buffer (2) IEC task
(3) Write outputs to output buffer (4) Bus cycle
(5) Input buffer (6) Output buffer
(7) Copy data to/from bus
(9) Bus cycle task, priority 1, 1 ms
(10) Bus cycle task, priority 5
(11) Bus cycle task, priority 10, interrupted by task 5

Task usage

The *"Task Deployment"* tab provides an overview of used I/O channels, the set bus cycle task, and the usage of channels.

WARNING!
If an output is written in various tasks, then the status is undefined, as this can be overwritten in each case.

If the same inputs are used in various tasks, then it is possible for the input to change during the processing of a task. This happens when the task is interrupted by a task with a higher priority and causes the process image to be read again. Solution: At the beginning of the IEC task, copy the input variables to variables and then work only with the local variables in the rest of the code.

Conclusion: Using the same inputs and outputs in several tasks does not make any sense and can lead to unexpected reactions in some cases.

See also

- *Chapter 1.4.1.7.1 "Configuring Devices and I/O Mapping" on page 213*
- *Chapter 1.4.1.20.2.8.1 "Generic device editor" on page 839*
- *Chapter 1.4.1.20.3.4.35 "Command 'Edit I/O Mapping'" on page 1018*
- *Chapter 1.4.1.20.2.8.12 "Tab '<device name> IEC Objects'" on page 859*
- *Chapter 1.4.1.20.4.3 "Dialog 'Select Function Block'" on page 1150*
- *Chapter 1.4.1.20.3.4.39 "Command 'Online Config Mode'" on page 1019*
- *Chapter 1.4.1.20.2.8.9 "Tab 'PLC Settings'" on page 850*

Tab '<device name> IEC Objects'

> **NOTICE!**
> Please note that manually creating another instance of the device object can lead to malfunctions.

In this tab of the generic device editor, "objects" are listed that allow for access to the device from the IEC application. In online mode, this is used as the monitoring view.

For devices for which a function block instance is created implicitly, at least this instance is listed as an object here in the table. This instance can be used, for example, in order to restart a bus or to query information from the application. The device type determines whether this kind of device instance is available and which access options it has. Please refer to the help for the special device configuration.

Instances of function blocks that are linked with inputs or outputs of the device are also displayed here. The mapping of a function block to a channel is defined in the *"<device name> I/O Mapping"* tab. The *"Go to Instance"* command takes you directly to the affected object from there.

In addition, you can create more objects in the table here that are not yet linked with a device channel.

In online mode, you can use the table of IEC objects as a monitoring view. It also shows the current value, the address, and the comment for the function block variable at the channel. Finally, it provides the capability of writing and forcing values.

"Add" *"Edit"*	Opens the *"Select Function Block"* dialog for creating a new instance or for editing the instance selected in the table.
"Delete"	Deletes the selected entry.
"Go to Variable"	Jumps from the selected entry directly to the corresponding mapping in the *"<device name> I/O Mapping"* tab.
"Variable"	The object name comprises the device name and the function block name. Example: `EL2004_Relay`. Changing the device name has an immediate effect. The part of the name after the device name is editable here.
"Mapping"	Mapping type, as in the *"<device name> I/O Mapping"* tab
"Type"	Data type: Here it is the name of the function block.
"Value" *"Prepared value"* *"Address "* *" Comment"*	In online mode only: Display of the current value, the address, and the comment for the variable at the channel. Moreover, the option of specifying a value for writing or forcing the variable.

See also

- *Chapter 1.4.1.20.2.8.11 "Tab '<device name> I/O Mapping'" on page 854*

Tab 'Users and Groups'

NOTICE!

Recommendations regarding data security

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running. Whenever possible, avoid exposing the PLC and control networks to open networks and the Internet. Use additional data link layers for protection, such as a VPN for teleaccess, and install firewall mechanisms. Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

On this tab of the generic device editor, you edit the device user management of the PLC.

Depending on how it is supported by the device, you can define user accounts and user groups. In combination with the configuration on the *“Access Rights”* tab, you thus control access to control objects and files at runtime.

Requirements: The controller has a user management and allows it to be edited. You have login data in order to be able to log in to the controller.

It is possible to apply user account definitions from the project user management into the device user management (see below: “Import” button).

Table 68: Toolbar of the tab

Synchronization	Switches on and off the synchronization between the editor and the user management on the device. If the button is not pressed, then the editor is blank or it contains a configuration that you loaded from the hard disk. When the button is pressed, CODESYS synchronizes the display in the editor continuously with the current user management on the connected device. When you enable the synchronization while the editor contains a user configuration that is not synchronized with the device yet, you are prompted what should happen to the editor contents. Options: ● *"Upload from the device and overwrite the editor content"*: The configuration on the device is loaded into the editor, overwriting the current contents. ● *"Download the editor content to the device and overwrite the user management there"*: The configuration in the editor is transferred to the device and applied there.
Import from disk	CAUTION: The import of a device user management by means of a `*.dum2` file completely overwrites the existing user management on the device. In order to log in to the device again afterwards, you need authentication data from the new user management. This means that you have to log in as a user from the imported user management after the import. ● When you click the button on the *"Users and Groups"* tab to import a *"Device user management file *.dum2"*, the default dialog for selecting a file opens to select a device user management file from the hard drive. After you select the file, the *"Enter Password"* dialog opens. You have to specify the password that was assigned when the file was exported. Then the user management is enabled. Note: Before V3.5 SP16, the *"Device user management files (*.dum)"* file type was used which did not require any encryption. ● When you click the button on the *"Access Rights"* tab to import a *"Device rights management file *.drm"*, the default dialog for selecting a file opens to select a corresponding file from the hard drive. The existing configuration in the dialog is overwritten by the imported file.
Export to disk	● When you click the button on the *"Users and Groups"* tab, first the *"Enter Password"* dialog opens for assigning a password to the device user management file. Note: This password has to be repeated later when this file is imported to enable this user management on the controller. After the password assignment dialog is closed, the default dialog for selecting and importing a user management configuration from the hard disk opens. In this case, the file type is *"Device user management files (*.dum2)"*. Note: Before V3.5 SP16, the *"Device user management files (*.dum)"* file type was used which did not require any encryption. ● When you click the button on the *"Access Rights"* tab, the file type is *"Device rights management files (*.drm)"*. In this case, a password does not have to be assigned for the file before saving.
"Device user"	User name of the user currently logged in on the device

Table 69: "User"

All currently defined users, and below them their memberships of user groups, are listed in a tree structure.	
"Add"	Opens the *"Add User"* dialog for creating a new user account.
"Import"	Opens the *"Import User"* dialog. It displays all the user accounts defined in the project user management. Select the desired entries and click *"OK"* in order to import them into the device user management. CAUTION: The passwords are NOT applied.

“Edit”	Opens the *“Edit User <user name>”* dialog. It corresponds to the *“Add User”* dialog and you can change the settings of the user account.
“Delete”	Deletes the account of the currently selected user.

Table 70: “Groups”

All currently defined groups, and below them the users assigned to them, are listed in a tree structure.	
“Add”	Opens the *“Add Group”* dialog. Define a new group name. From the list of defined users, select those that are to belong to the group. Click *“OK”* to confirm the selection. The group is displayed in the tree.
“Import”	Opens the *“Import User”* dialog. It displays all the user groups defined in the project user management. Select the desired entries and click *“OK”* in order to import them into the device user management.
“Edit”	Opens the *“Edit Group <group name>”* dialog. It corresponds to the *“Add Group”* dialog where you can change the group definition.
“Delete”	Deletes the currently selected group.

Table 71: “Add Dialog 'Add User'”

“Name”	Name of the new user
“Default group”	List box with all configured user groups. Every user has to belong to at least one group. You define this here as a default group.
“ Password”	
“Confirm password”	
“Password strength”	Password security in a range from *“Very weak”* to *“Very good”*.
“Hide password”	☑: The password is shown only with asterisks **"*"** when it is typed in.
“Password can be changed by the user”	
“Password must be changed at first login”	

See also

- *Chapter 1.4.1.20.2.8 “Object 'Device' and Generic Device Editor” on page 839*
- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*
- *Chapter 1.4.1.20.2.8.14 “Tab 'Access Rights'” on page 863*
- *Chapter 1.4.1.20.3.6.16 “Command 'Add Device User'” on page 1041*

Tab 'Access Rights'

NOTICE!

Recommendations regarding data security

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running. Whenever possible, avoid exposing the PLC and control networks to open networks and the Internet. Use additional data link layers for protection, such as a VPN for teleaccess, and install firewall mechanisms. Restrict access to authorized persons only, and change any existing default passwords during the initial commissioning, and change them regularly.

NOTICE!

Detailed information on the concept and use of device user management is provided in "Handling of Device User Management".

There you will also find the following instructions on how to use the editor:

- First-time login to the controller for editing and viewing its user management
- Setting up a new user in the user management of the controller
- Changing of access rights to controller objects in the user management of the controller
- Loading user management from a *.dum file, modifying it, and downloading it to the controller in offline mode

On this tab of the device editor, you define the device access rights of device users to objects on the controller. As in the project user management, users must be members of at least one user group and only user groups can be granted certain access rights.

Requirements for the *“Access Rights”* tab to be displayed:

- In the CODESYS options, in the *“Device Editor”* category, the *“Show access rights page”* option has to be selected.
  Note that this CODESYS option can be overwritten by the device description.

Requirements for the access rights to be granted to user groups

- A component for the user management has to be available on the controller. That is the primary requirement.
- Users and user groups have to be configured on the *“Users and Groups”* tab.

Table 72: Toolbar of the tab

Synchronization	Switches on and off the synchronization between the editor and the user management on the device. If the button is not pressed, then the editor is blank or it contains a configuration that you loaded from the hard disk. When the button is pressed, CODESYS synchronizes the display in the editor continuously with the current user management on the connected device. When you enable the synchronization while the editor contains a user configuration that is not synchronized with the device yet, you are prompted what should happen to the editor contents. Options: ● *"Upload from the device and overwrite the editor content"*: The configuration on the device is loaded into the editor, overwriting the current contents. ● *"Download the editor content to the device and overwrite the user management there"*: The configuration in the editor is transferred to the device and applied there.
Import from disk	CAUTION: The import of a device user management by means of a `*.dum2` file completely overwrites the existing user management on the device. In order to log in to the device again afterwards, you need authentication data from the new user management. This means that you have to log in as a user from the imported user management after the import. ● When you click the button on the *"Users and Groups"* tab to import a *"Device user management file *.dum2"*, the default dialog for selecting a file opens to select a device user management file from the hard drive. After you select the file, the *"Enter Password"* dialog opens. You have to specify the password that was assigned when the file was exported. Then the user management is enabled. Note: Before V3.5 SP16, the *"Device user management files (*.dum)"* file type was used which did not require any encryption. ● When you click the button on the *"Access Rights"* tab to import a *"Device rights management file *.drm"*, the default dialog for selecting a file opens to select a corresponding file from the hard drive. The existing configuration in the dialog is overwritten by the imported file.
Export to disk	● When you click the button on the *"Users and Groups"* tab, first the *"Enter Password"* dialog opens for assigning a password to the device user management file. Note: This password has to be repeated later when this file is imported to enable this user management on the controller. After the password assignment dialog is closed, the default dialog for selecting and importing a user management configuration from the hard disk opens. In this case, the file type is *"Device user management files (*.dum2)"*. Note: Before V3.5 SP16, the *"Device user management files (*.dum)"* file type was used which did not require any encryption. ● When you click the button on the *"Access Rights"* tab, the file type is *"Device rights management files (*.drm)"*. In this case, a password does not have to be assigned for the file before saving.
"Device user"	User name of the user currently logged in on the device

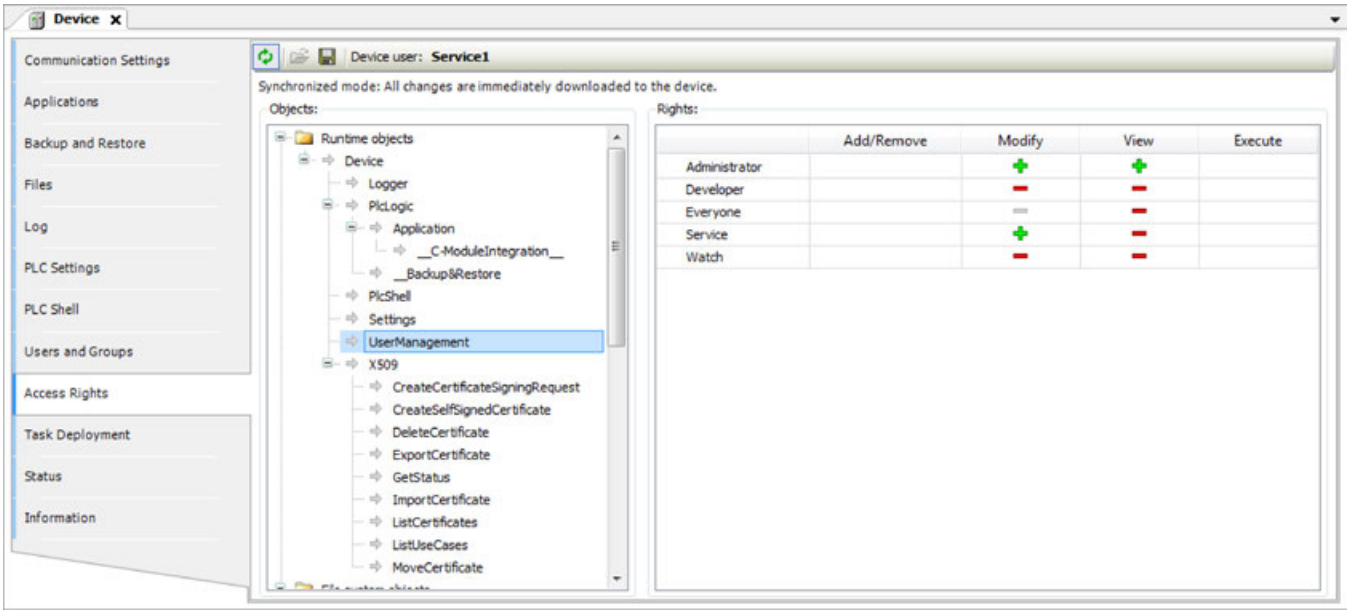
Table 73: “Objects”

In the tree structure, the objects are listed to which actions can be executed at runtime. The objects are each assigned by their object source and partially sorted in object groups. In the *“Rights”* view, you can configure the access options for a user group to a selected object.
Object source (root node) ● *“File system objects ➔ Device”*: In these objects, the rights can be granted to folders of the current execution directory of the controller. ● *“Runtime objects ➔ /”*: In these objects, all objects are managed that have online access in the controller and therefore have to control the access rights. A description of the objects is located in the table. *“Overview of the objects” on page 867*
Object groups and objects (indented) Example: *“Device”* with child nodes *“Logger”*, *“PlcLogic”*, *“Settings”*, *“UserManagement”*.

Table 74: “Rights”

In general, the access rights are inherited from the root object (also *“Device”* or *“/”* to the subobjects. This means that if a permission of a user group is denied or explicitly granted to a parent object, then this first affects all child objects.

The table applies for the object that is currently selected in the tree. For every user group, it shows the rights currently configured for the possible actions on this object.



Possible actions on the object:

- *“Add/Remove”*
- *“Modify”*
- *“View”*
- *“Execute”*

When an object is clicked, a table on the right side shows the access rights of the available user groups for the selected object.

This allows you to quickly see:

- Which access rights are evaluated by an object
- Which user group has which effective rights to which object

Meanings of the symbols

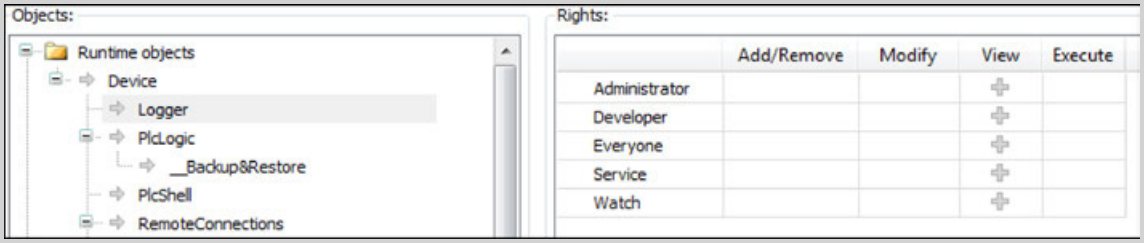
- ✚: Access right granted explicitly
- ━: Access right denied explicitly
- ✚: Access right granted through inheritance
- ━: Access right denied through inheritance
- ✖: The access right was not granted or denied explicitly and also not inherited by the parent object. Access is not possible.
- No symbol: Multiple objects are selected that have different access rights.

Change the permission by clicking the symbol.

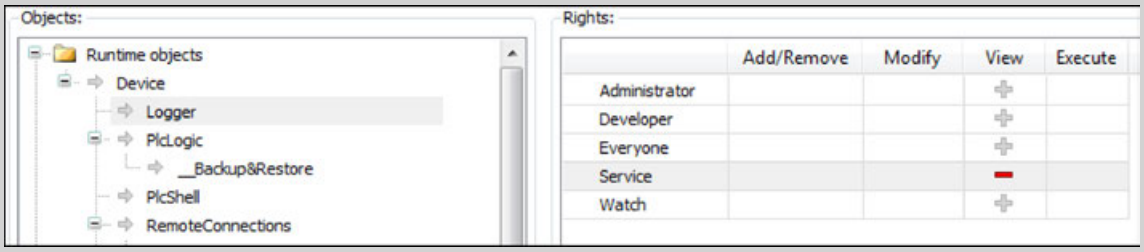
Example

The *"Logger"* object on the *"Access Rights"* tab was created by the "Logger" component and controls its access rights. It is located directly below the *"Device"* runtime object.

The possible access rights for this object can be granted only for the *"View"* action.



Initially, each object has a read access. This means that every user can read the "Logger" of a controller. If this access right should be denied for a single user group (*"Service"* in the example), then the read access to the logger object has to be denied explicitly.



Overview of the objects

"Runtime objects ➔ Device"						
"Logger"	Online access to the logger is read only. Therefore, only the *"View"* access right can be granted or denied here.					
"PlcLogic"	All IEC applications are inserted here automatically as child objects during download. When an application is deleted, it is removed automatically. This allows specific control of online access to the application. Access rights can be assigned centrally over all applications in the *"PlcLogic"* The *"Administrator"* and *"Developer"* user groups have full access to the IEC applications. The *"Service"* and *"Watch"* user groups only have read access (for example for read-only monitoring of values).					
	The following table shows which action is affected in particular when a specific access right is granted for an IEC application. **x** : The right has to be set explicitly. **–** : The right is not relevant.					
	"Application"	**Operation**	**Access Rights**			
			"Add/ Remove"	*"Execute"*	*"Modify"*	*"View"*
		Login	–	–	–	x
		Create	x	–	–	–
		Create child object	x	–	–	–
		Delete	x	–	–	–
		Download / online change	x	–	–	–
		Create Boot Application	x	–	–	–

		Read variable	–	–	–	x
		Write Variable	–	–	x	x
		Force variable	–	–	x	x
		Set and delete break-point	–	x	x	–
		Set Next Statement	–	x	x	–
		Read call stack	–	–	–	x
		Single cycle	–	x	–	–
		Switch on flow control	–	x	x	–
		Start / Stop	–	x	–	–
		Reset	–	x	–	–
		Restore retain variables	–	x	–	–
		Save retain variables	–	–	–	x
“PlcShell”	Only the *“Modify”* permission is evaluated at this time. This means that only when the *“Modify”* permission has been granted to a user group can PLC shell commands also be evaluated.					
“RemoteConnections”	Additional external connections to the controller can be configured below this node. Currently, access to the CODESYS OPC UA server can be configured here.					
“Settings”	This is the online access to the configuration settings of a controller. By default, access to *“Modify”* is granted only to the administrator.					
“UserManagement”	This is the online access to the user management of a controller. By default, read/write access is granted only to the administrator.					
“X509”	This controls the online access to the X.509 certificates. Two types of access are distinguished here: ● Read (*“View”*) ● Write (*“Modify”*) Every operation is assigned to one of these two access rights. Each operation is inserted as a child object below X509. Therefore, access per operation can now be fine-tuned even more.					
“File system objects ➔ /”						
	All folders from the execution path of the controller are inserted below the "*“/”*" file system object. This allows you to grant specific rights to each folder of the file system.					

See also

- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*
- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*
- *Chapter 1.4.1.20.2.8.13 “Tab 'Users and Groups'” on page 860*

Tab 'Symbol Rights'

In this tab of the generic device editor, you define the access rights of different user groups (clients) to the individual symbol sets available on the controller.

Requirement: User management must be set up on the PLC. An application was downloaded to the controller for which symbol sets were defined in the CODESYS project. They have access data for logging in to the controller.

In the *“Symbol Sets”* view, all symbol sets are listed below the *“Application”* node whose definition was downloaded with the application to the controller.

In the *“Rights”* view, the user groups defined in the user management of the controller are listed in a table. When a symbol set is selected, you see the access rights of the corresponding user group to the symbols of this set. : Access granted; : Access not granted. You can change the access rights by double-clicking the symbol.

Click the button to save the current access configuration to an XML file The file type is *“Device symbol management files (*.dsm)”*. Click the button to read a file like this from the hard drive.

See also

- *“Creating symbol sets with different access rights for different control clients” on page 359*
- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*
- *Chapter 1.4.1.20.2.8.14 “Tab 'Access Rights'” on page 863*
- *Chapter 1.4.1.20.2.8.13 “Tab 'Users and Groups'” on page 860*

Tab 'Licensed Software Metrics'

The tab of the device editor displays the code sizes of the applications of the open project in a tree structure. The display is refreshed when you click *“Build ➔ Generate Code”* or *“Online ➔ Login”* for the active application. When the compile information is deleted, the displayed code size of the corresponding application is reset.

“Metric”	Applications of the open project
“Size”	● *“Size of User Code”*: Sum of the displayed code sizes of the applications listed below ● Code size of the respective application
“ Unit”	Unit in which the *“Size”* is displayed
“Max. Allowed”	Not implemented yet

Tab 'Task deployment'

This sub-dialog box of the device editor displays a table of inputs and outputs as well as their assignment to the defined tasks.

The information only becomes visible after code has been generated for the application. It is used for troubleshooting, because it shows where inputs or outputs are used in several tasks with different priorities. Multiple use can lead to undefined values through overwriting.

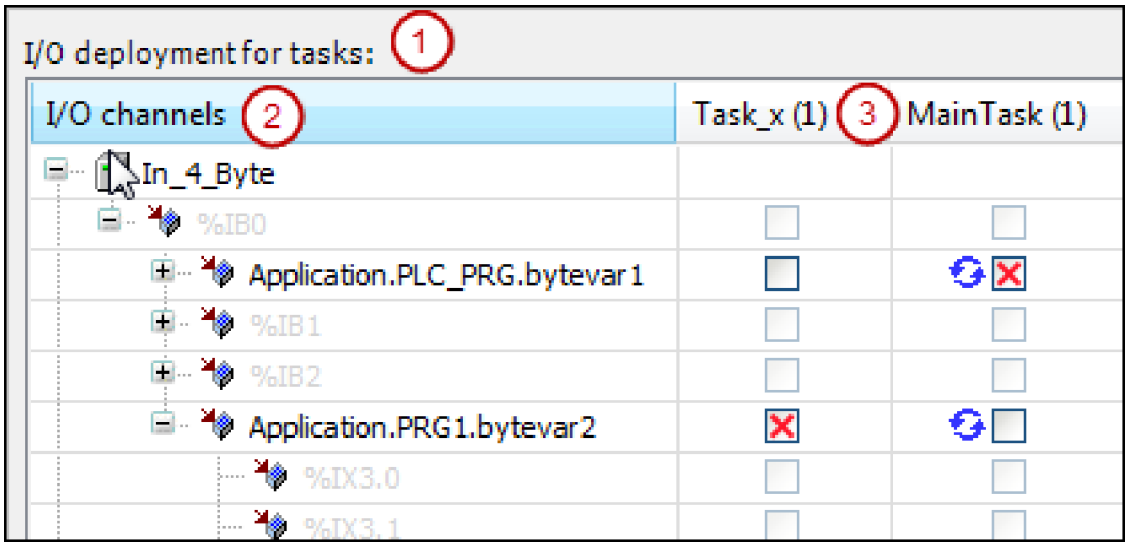


Table 75: "I/O Deployment for Tasks" (1)

"I/O Channels" (2)	All inputs and outputs of the linked devices. The display corresponds to that in the dialog box *"I/O Mapping"* of the device editor. By double-clicking on an input or output you can open the associated I/O mapping editor.
"<task name>" (3)	A column appears for every task defined in the task configuration. The title contains the task name and priority. The priority of the tasks decreases from first to the last column. A red cross appears in the box for inputs and outputs that are written or read by a task: ☒. In addition, the task defined as a *"Bus cycle task"* in the *"PLC Settings"* of the device editor is marked at these points with a blue double arrow symbol ⟳. Following a mouse-click on the title cell, only the I/Os assigned to this task are displayed. Following a mouse-click on the *"I/O Channels"* cell, all channels are shown again.

See also

- *Chapter 1.4.1.20.2.8.1 "Generic device editor" on page 839*
- *Chapter 1.4.1.20.2.8.11 "Tab '<device name> I/O Mapping'" on page 854*
- *Chapter 1.4.1.20.2.8.9 "Tab 'PLC Settings'" on page 850*
- *Chapter 1.4.1.8.16.1 "Creating a task configuration" on page 293*

Tab 'Status'

This tab of the generic device editor displays status information, for example 'Running' or 'Stopped', and specific diagnostic messages from the respective device, also information about the card used and the internal bus system.

See also

- *Chapter 1.4.1.20.2.8.1 "Generic device editor" on page 839*

Tab 'Information'

This tab of the generic device editor displays general information that originates from the device description file: name, vendor, categories, version, order number, description, if necessary an illustration.

See also

- *Chapter 1.4.1.20.2.8.1 “Generic device editor” on page 839*

Object 'GlobalTextList'

Symbol: 

This object is for the management and translation of texts that are written as static text in visualizations in the project. It contains a table with these texts. If you write a text in a visualization in an element under the property *“Texts”*, CODESYS automatically adds a line in the table. You cannot write any new text here, you can only edit an existing text.

In addition CODESYS makes the following commands available, in order to consolidate the *“GlobalTextList”*:

- *“Check Visualization Text IDs”*
- *“Update Visualization Text IDs”*
- *“Remove Unused Text List Entries”*

The object is located in the POUs view and exists once at the most

“ID”	Unambiguous identifier of the text
“Default”	Source text as a character string with one formatting specification at the most, for example `Information A: %i possibilities`. If no translation is written under a language column , CODESYS uses this text. Double-click in the field in order to edit the text.
The table contains as many language columns as you have added. A language column is named with a language code that you entered when creating the column with the command *“Insert Language”*.	
“<Language code>”	Name of the language as a language code, for example `en-US`. This column contains the translation of the text that is written under *“Standard”*. If the language code is selected as a language in the visualization manager, a visualization displays the translation during operation. A running visualization can switch over during operation to another language at the request of a user. Double-click in the field in order to edit the text.

See also

- *Chapter 1.4.1.20.3.20.1 “Command 'Add Language'” on page 1132*
- *Chapter 1.4.1.20.3.20.2 “Command 'Create Global Text List'” on page 1132*
- *Chapter 1.4.1.20.3.20.6 “Command 'Import/Export Text Lists'” on page 1133*
- *Chapter 1.4.1.20.3.20.7 “Command 'Remove Language'” on page 1134*
- *Chapter 1.4.1.20.3.20.9 “Command 'Remove Unused Text List Entries'” on page 1135*
- *Chapter 1.4.1.20.3.20.10 “Command 'Check Visualization Text IDs'” on page 1135*
- *Chapter 1.4.1.20.3.20.11 “Command 'Update Visualization Text IDs'” on page 1135*
- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*
- *Chapter 1.4.1.8.8 “Managing text in text lists” on page 266*

Object 'GVL' - Global Variable List

Symbol: 

A global variable list is used for the declaration, editing and display of global variables.

A GVL is added to the application or the project with the command *“Project ➔ Add Object ➔ Global Variable List”*.

If you insert a GVL under an application in the Device tree, the variables are valid within this application. If you add a GVL in the `POUs` view, the variables are valid for the entire project.

You can apply settings for the editor of the object in the dialog *"Tools ➔ Options"* in the categories *"Declaration Editor"* and *"Text Editor"*.

If the target system supports network functionality, you can convert the variables of a GVL into network variables and thus use them for data exchange with other devices in the network. To do this you must define corresponding properties for the GVL in the *"Network Variables"* tab of the *"Properties"* dialog.

See also

- *Chapter 1.4.1.8 "Programming of Applications" on page 222*
- *Chapter 1.4.1.20.4.10.11 "Dialog 'Properties' - 'Network Variables'" on page 1163*
- *Chapter 1.4.1.20.4.13 "Dialog 'Options'" on page 1186*

Object 'GVL' - Global Variable List (task-local)

Symbol: 

A global variable list (task-local) is used for the declaration, editing and display of global variables. For this special global variable list, the declared variables in the list can be written by one task only. All other tasks have only read-only access. This makes sure that the values of these variables are always consistent, even for multicore projects.

The object is available for compiler version 3.5.13.0 with the corresponding device description.

"Task with write access"	Task that has exclusive write access to the variables.

See also

- *Chapter 1.4.1.8.2.5 "Using Task-Local Variables" on page 230*
- *Chapter 1.4.1.8.2.4 "Declaring global variables" on page 229*
- *Chapter 1.4.1.20.2.10 "Object 'GVL' - Global Variable List" on page 871*
- *Chapter 1.4.1.20.2.26.5 "Tab 'Task Groups'" on page 941*

Object 'Persistent variable list'

Symbol: 

The object contains the declaration of global persistent variables in the declaration section `VAR_GLOBAL PERSISTENT RETAIN .. END_VAR`. The variables are stored in special non-volatile memory.

The persistence editor shows the variables as a list in the usual way. The displayed list does not influence the persistence behavior of the variables, but only the list stored internally in the process image. The list there contains all variables ever declared in chronological order. Variables that you have removed are marked with a placeholder and continue to exist as a gap.

The declaration section can also contain instance paths, which refer to locally declared persistent variables and were created with the command *"Declarations ➔ Add All Instance Paths"*.

NOTICE!
Before you decide how to set up persistence for an application, it would be helpful for you to be familiar with the use cases described in the *"Data Persistence"* section. Moreover, it is helpful if you can differentiate between the mechanisms of persistent variables, retain variables, variables of the Persistence Manager, and recipe variables.

Commands

The following commands are provided in the persistence editor:

- Command *“Declarations ➔ Add All Instance Paths”*
- Command *“Declarations ➔ Reorder List and Clear Gaps”*

See also

- TODO
- *Chapter 1.4.1.20.3.17.1 “Command 'Reorder List and Clean Gaps'” on page 1123*
- *Chapter 1.4.1.8.19.1 “Preserving data with persistent variables” on page 304*
- *Chapter 1.4.1.8.19.2 “Preserving data with retain variables” on page 306*
- *Chapter 1.4.1.20.3.17.4 “Command 'Add all instance paths'” on page 1124*

Object 'Image Pool'

The *“Image Pool”* object contains a table with image ID assignments.

“ID”	ID of the image; you reference this ID, for example in the visualization of the image.
“File name”	File path of the image; if you click for more settings (...), the *“Select Image”* dialog box opens.
“Image”	Show a thumbnail of the image.
“Link type”	Opens the *“Select Image”* dialog box, where you define the link type.

Dialog box 'Select Image'

“Image file”	Name and directory of the image file (example: "C:\Programme\images\logo.bmp") CODESYS supports the following image formats: BMP, EMF, GIF, ICO, JPG, PNG, SVG, and TIFF. Please note that a controller may not support all formats. Whether or not you can use images formatted as scalable vector graphics (*.svg) depends on the operating system. Any necessary information is located in the device description of the hardware vendor.

Table 76: “File Handling”

“Remember the link”	CODESYS saves only the link. CODESYS automatically updates any changes to an image file in the image pool. You must ensure that the path of the image file does not change. When saving the project as an archive, CODESYS embeds the image file in the project archive.
“Remember the link and embed into project”	CODESYS copies the image to the image pool and the link information is retained. In this way, CODESYS recognizes any changes to the image file and then update the image pool can as needed. This behavior is controlled with the options in the next table. Embedded image files increase the memory requirement of the project.
“Embed into project”	CODESYS copies the image to the image pool. If the image file is changed again afterwards, then it is not updated in the project. For libraries, you must embed the image in the project. Embedded image files increase the memory requirement of the project.

Table 77: “Change Tracking”

These options are available only if you have selected the *“Remember the link and embed into project”* check box as described above.	
“Reload the file automatically”	CODESYS automatically updates the image file in the project without prompting.
“Prompt whether to reload the file”	If the image file has changed, you may be prompted whether or not the image file should be updated.
“Do nothing”	CODESYS does not update the image file in the image pool.

See also

- *Chapter 1.4.1.8.9 “Using image pools” on page 274*

Object 'Library Manager'

Symbol: 

The Library Manager lists all libraries that were integrated in the project for creating applications. It provides information about the type of library, its properties, and its contents.

You can expand or collapse the list of integrated libraries, as well as edit library properties for non-dependent libraries.

The Library Manager consists of three views:

- Upper view: List of integrated libraries
- Lower left view: Tree structure with all modules of the library selected in the upper view
- Lower right view: Documentation for the module selected in the tree

See also

- *Chapter 1.4.1.16 “Using Libraries” on page 448*

List of integrated libraries List of all libraries integrated in the project. If a library depends on other libraries, then these referenced libraries are automatically integrated.

Displayed in gray fonts	The library was added to the project automatically by means of a plug-in.
Displayed in black fonts	The library was added to the project automatically by means of the *“Add Library”* command.
“Name”	Display of the integrated library in the following syntax: *“<placeholder name> = <library name>, <version> (<company>)”*: *“<placeholder name>”*: If it is a placeholder library for a library, then the placeholder name is before a *“ = ”*. *“<library name>”*: Name of the library that is used for management in the library repository. *“<version>”*: Version that was referenced at the first time it was integrated. *“ (<company>)”*: Vendor (optional)

"Namespace"	Namespace for unique access to the contents of the library. It is prepended to a module identifier for this purpose: <namespace>.<library module identifier> The namespace usually coincides with the library name. Note: If the library has the property `LanguageModelAttribute "qualified-access-only"`, then you **must** access the library module in the application code by means of the namespace. Qualified (unique) access is enforced. You can modify the standard namespace for local use (within the project) in the *"Properties"* dialog.
" Effective version"	Version of the library after the resolution. This version is used in the project. Requirement: The Library Manager exists in the *"Devices"* view and a placeholder library is selected. Example: `3.5.10.0` A placeholder library that is integrated below an application is resolved by assigning a special resolution to the placeholder library in the *"Placeholders"* dialog. Then the selected library is loaded. Other resolutions are ignored. If no special resolution is given, then a check is performed as to whether or not a resolution is specified in the device description and library profile of the application. The first search hit is applied.
	Symbol with tooltip to notify about the current device-dependent resolution of the selected library. Example when the Library Manager is in the *"Devices"* view: *"This placeholder is explicitly redirected to this version (see the Placeholders dialog)"* Example when the Library Manager is in the *"POUs"* view: *"In the 'Device_1' device, the placeholder is resolved to 'VisuElemsAlarms, 1.0.0.0 (System)"* A placeholder library that is integrated in the *"POUs"* view is resolved by checking depending on the application whether or not a resolution is specified in the device description. Afterwards, the library profile is checked. The first detected resolution is used. If you have assigned a special resolution to the placeholder library in the *"Placeholders"* dialog, then this will **always be ignored**. The result is shown in the tooltip of the symbol.
	Library that is signed with a trusted certificate (compatible with CODESYS >= V3 SP15)
	Library that is signed with a trusted certificate, but references at least one unsigned library
	Library that is signed with a private key and token (compatible with CODESYS < V3 SP15)
	Library that is not signed, or signed with an untrusted or expired certificate. In the case of an untrusted certificate, the *"Trust Certificate"* command is provided in the context menu.
	Library that is defined as optional and not currently available
	Library whose status is being determined
	Licensed library for which no valid license is currently available
	Library symbol for a library that cannot be loaded because its signature (encryption) could not be verified

Table 78: Commands in the Library Manager

"Add Library"	Opens the dialog for selecting a library. All libraries installed in the library repository are offered.
"Delete Library"	Removes the presently selected library from the project
"Properties"	Opens the dialog for the display and editing of the properties of the presently selected library
"Details"	Opens a dialog with details for the presently selected library (general information, contents, properties, license information)
"Try to Reload Library"	If you select a library marked as not found, you can attempt to load it into the project again using this command.
"Download Missing Libraries"	CODESYS scans for the missing libraries in the download servers specified in the project options. After that you can download and install the library.
"Placeholders"	The *"Placeholders"* dialog opens. The current resolution is displayed there and you can edit it.
"Library Repository"	Opens the *"Library Repository"* dialog for installing and uninstalling libraries and for defining library locations
"Icon legend"	Opens the *"Information"* dialog with a legend of the icons that display the current status of a library in the list of integrated libraries (see above)
"Summary"	Opens the *"Library Summary"* dialog. All libraries referenced in the project are displayed in a tree structure in the dialog, and those libraries which reference these libraries. ● Command *"Display all occurrences in library hierarchy and close dialog"*: In the editor of the Library Manager, the libraries in the open tree structure are marked which reference or use this library. Requirement: A library is selected. The *"Information"* dialog is then closed. This command is also executed when you double-click a library. Display of the libraries ● *"Managed Library"*: Name and version of the library ● *"Number of Occurrences"*: Number of locations where this library is referenced by other libraries. When you click *"+"* for a library, the libraries, which reference this library, are displayed in the next level down.

"Trust Certificate"	Only in the context menu of a library selected in the Library Manager, in which the library has been signed with an untrusted certificate. The command turns the untrusted certificate into a trusted certificate and the prepended icon changes from to .
"Export Library"	Only in the context menu of a library selected in the Library Manager: Opens the default dialog for saving the library file in the file system

See also

- *Chapter 1.4.1.20.3.14.1 "Command 'Add Library'" on page 1116*
- *Chapter 1.4.1.20.3.14.5 "Command 'Export Library'" on page 1120*
- *Chapter 1.4.1.20.3.14.3 "Command 'Properties'" on page 1118*
- *Chapter 1.4.1.20.3.8.5 "Command 'Library Repository'" on page 1061*
- *Chapter 1.4.1.20.3.14.4 "Command 'Placeholders'" on page 1120*
- *Chapter 1.4.1.20.4.13.15 "Dialog 'Options' – 'Library Download'" on page 1195*
- *Chapter 1.4.1.20.2.21 "Object 'Project Information'" on page 919*

Tree structure of all modules of a selected library

All library modules that were integrated with the library are listed in the tree structure.

Requirement: A library is selected in the upper view.

The usual sorting and search functions are available in the menu bar.

Documentation for the library module selected in the lower left view

Tab *“Inputs/Outputs”*	Interface (inputs/outputs) of the library module
Tab *“Graphical”*	Graphical display of the module
Tab *“Documentation”*	Documentation for the library module. Note: As a library developer, you have to follow the rules for documentation inclusion in 'Guidelines for library development'.
Tab *“Parameter List”*	Requirement: The library project contains a parameter list. You can change the values of these parameters in the column *“Value (editable)”*.

See also

- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*

Object 'OPC UA Information Model'

Symbol:

The *“OPC UA Information Model”* object is added to the *“Communication Manager”* in the application. When added, an OPC UA publishing object and below that an information model object as a child object are also added.

In the *“Add OPC UA Information Model”* dialog, specify a name for the information model and select the OPC UA information model. The selection includes the OPC UA information models which are installed in the *“OPC UA Information Model Repository”*.

OPC UA information model editor

Symbol:

The editor is used to select the object types and data types of the OPC UA information model which you want to use in the open CODESYS project. The selected OPC UA types are converted to IEC types in the editor.

“Browse Information Model ”	List box The currently used information model and the information models which are referenced by the current model are displayed. The dependencies depend on the respective information model. The OPC UA base model is always displayed.
“Generate IEC declarations ”	Generates an IEC declaration for all OPC UA types converted into an IEC type. The generated IEC types are saved in a folder (example: *“OPC Objects”*) in the *“Devices”* view and can be used in the implementation of the IEC code. When implementing the CODESYS project, you can select them in the *“Input Assistant”* dialog. When the IEC declarations are generated, the appropriate attributes are automatically added to the generated POUs (example: `'opcua.mapping.type','opcua.mapping.member.accesslevel'`). Note: The attributes added by the system should not be changed by the user. When the IEC type cannot be created, the entry `UNKNOWN_TYPE` is displayed in the declaration instead of the data type. The user should delete this variable because in this case it is almost always an OPC UA feature which is not supported yet. OPC UA features which are not supported yet are grayed out in the left area.
“Data Model”	
Left area: OPC UA data model	
“Types”	Display of the OPC UA data types and object types in a tree structure When you drag an OPC UA type to the right area, CODESYS converts the OPC UA type into the corresponding IEC type which can be used in the implementation of the CODESYS project. In this case, only the root node of an OPC UA type can be dragged to the right area. For a detailed description of the assignment of individual OPC UA types to the corresponding IEC types in the mapping operation, see the chapters "Mapping of OPC UA Types to IEC Types" and "Mapping of Reference Types".
“Element Type”	OPC UA element type
“Reference Type”	OPC UA reference types Example: `HasComponent`, `HasProperty` For a description of these reference types, see the chapter "Mapping of OPC UA Types to IEC Types".
“Modelling Rule”	● *“Mandatory”*: For the corresponding OPC UA type, the respective members are generated in the project when the *“Generate IEC declarations”* command is executed. In the right area, the *“Generate member”* field is activated and cannot be deactivated. ● *“Optional”*: Generating an IEC member for this OPC UA type is optional. ● *“Optional placeholder”*: in the right, you can drag another IEC type for this placeholder. For an example as a screenshot, see the chapter "Using OPC UA Companion Information Models".
Right area: Object types and data types of the OPC UA information model which are mapped to IEC types	
“Name”	Name of the IEC POU or data type in the project By default, the name of the type is displayed in the OPC UA information model. OPC UA also supports names which are invalid in IEC. In these cases, CODESYS automatically generates a valid IEC name. You can change the name.
“IEC Type”	IEC type to which the OPC UA type was mapped (example: `BOOL`, *“Method”*).

"OPC UA Type"	Corresponds to the *"Element Type"* displayed in the left area
"Generate member"	● ☑: When the *"Generate IEC declarations"* command is executed, a corresponding member or a placeholder is generated in the project. Only the interfaces are automatically generated here. The implementation still has to be manually created later in a POU. When the *"Modelling Rule"* is *"Mandatory"* for the OPC UA type in the right area, this option cannot be deactivated. ● ☐: When the *"Generate IEC declarations"* command is executed, a corresponding member is not generated in the project. Click this option to activate it.

OPC UA publishing editor

Symbol:

In the editor, the instances (OPC UA objects) of the OPC UA types are configured which should be available to the OPC UA Clients via the controller

"Search for Mapped Instances"	Searches in GVLs and PRGs below the current application for instances of the mapped OPC UA types which have already been declared. The search result is displayed in the list. Note: Instances in the *"POUs"* view and in libraries are not taken into consideration.
"Create New Instance"	Opens the *"Create New Instance"* dialog to select the IEC type for which a new instance should be generated. Instances can be generated for the POUs which have been created in the OPC UA information model editor from OPC UA types. These instances can be used in POUs in the application. Requirement: In the OPC UA information model editor, the *"Generate IEC declarations"* command has been executed after mapping the OPC UA types to the IEC types.
"Root Node"	Selection of directories or the object instance of the server which is displayed on the OPC UA Client for publishing the instances. The list box depends on the applied OPC UA companion specification.
Tabular list of generated instances:	
"OPC UA Variable"	Variable which has been generated as an instance of an OPC UA type. This variable can be published in an OPC UA Client. You can edit the displayed name.
"OPC UA Type"	OPC UA type of the *"OPC UA Variable"*
"Map or Generate"	● : The *"OPC UA Variable"* has been mapped to an existing variable. ● : The *"OPC UA Variable"* has been generated as a new instance.
"IEC Variable"	Full variable name
"IEC Type"	IEC type of the IEC variable
"Access Rights"	Note that an OPC UA Client may have read/write access the OPC UA variable. In the function blocks, the access rights to the variables can be changed by attributes which can also be read from the XML file if necessary. Reading and writing
"Maximal"	Maximum possible permissions for the OPC UA variable

See also

- *Chapter 1.4.1.20.3.8.12 "Command 'OPC UA Information Model Repository'" on page 1069*

Object 'Network Variable List (Sender)'

Symbol: 

A network variable list (sender) is used for declaring and listing global variables that should be sent to network variable lists (receiver) of other devices or network projects.

You add the object to the device tree by clicking *"Add Object ➔ Network Variable List (Sender)"* of an application.

You can configure the protocol and transfer parameters in the *"Add Network Variable List (Sender)"* dialog box or *"Properties"* dialog box of the object in the *"Network Variables"* tab.

Dialog Box 'Add Network Variable List (Sender)'

Function: This dialog box defines the network properties for the sender NVL. When you close the dialog box, CODESYS adds the sender NVL of the application to the device tree.

Call: Main menu *"Project ➔ Add Object ➔ Network Variable List (Sender)"* while the application is selected in the device tree.

This dialog box corresponds to the *"Network Variables"* tab in the *"Properties"* of the network variable list object.

See also

- *Chapter 1.4.1.20.4.10.11 "Dialog 'Properties' - 'Network Variables'" on page 1163*
- *Chapter 1.4.1.20.2.17 "Object 'Network Variable List (Receiver)'" on page 880*
- *Chapter 1.4.1.9.3.1 "Configuring a Network Variable Exchange" on page 361*

Object 'Network Variable List (Receiver)'

Symbol: 

The object is used for listing the received network variables and displaying the information: network and transmit information and sender.

You add the object to an application by clicking *"Add Object ➔ Network Variable List (Receiver)"*.

The network variable list (receiver) shows the received network variables, which were declared in network variable list (sender) of another device or project. You cannot change the network variables in the object editor.

The object editor consists of two parts:

- Information about the sender and transfer log of the list
- List of declarations of network variables

Dialog Box 'Add Network Variable List (Receiver)'

Function: This dialog box defines the receiver NVL to a sender NVL and adds the receiver NVL to the application object in the device tree.

Call: Main menu *"Project ➔ Add Object ➔ Network Variable List (Receiver)"* (when the application object is selected).

"Task"	Task of the current application that controls the variables to be received.
"Sender"	Drop-down list ● Available sender NVLs of another device in the project ● *"Import from file"*: Required if the necessary sender NVL is defined in another project. For this, the necessary sender NVL must have been generated in another project as *"GVL export file *.gvl"* in the properties dialog of the NVL in the *"Link To File"* tab.
"Import from file"	File name in *"GVL export file *.gvl"* format if you have selected *"Import from file"* for *"Sender "*.

See also

- *Chapter 1.4.1.9.3.1 “Configuring a Network Variable Exchange” on page 361*
- *Chapter 1.4.1.20.2.16 “Object 'Network Variable List (Sender)'” on page 880*

Object 'POU'

Symbol: 

An object of the type *“POU”* is a Program Organization Unit in a CODESYS project. You write source code for your controller program in POUs.

There are the following types of POUs:

- Program
- Function
- Function block

A *“POU”* object is inserted by using the command *“Project ➔ Add Object”* in the Device tree or in the *“POUs”* view. When adding a POU you define the POU type and the implementation language.

You can also add other programming objects (method, action, etc.) to these objects.

Calling POUs

Certain POUs can call other POUs. Recursions are not permitted.

When calling POUs via the namespace, CODESYS browses the project for the POU to be called in accordance with the following order:

1. Current application
2. *“Library Manager”* of the current application
3. *“POUs”* view
4. *“Library Manager”* in the *“POUs”* view

If you want to call a POU that exists with the same name in a library used in the application and as an object in the “POUs” view, note the following: There is no syntax that allows you to call the POU in the “POUs” view only by its name. In this case you must shift the library from the application's library manager to the project's library manager (in the “POUs” view). After that you can call the POU object in the “POUs” view purely by its name. If you add the namespace to the library, you can call the POU of the library.

The term “POU” is also used in CODESYS for the “POUs” view in which CODESYS manages the global objects in the project.

See also

- *Chapter 1.4.1.20.2.18 “Object 'POU'” on page 881*
- *Chapter 1.4.1.20.4.10 “Dialog 'Properties'” on page 1157*

Dialog 'Add POU'

Function: The dialog is used to configure a new POU according to the IEC 61131-3 standard. This means that a POU can be a program, a function, or a function block.

Call: *“Project ➔ Add Object”* menu; context menu in the *“Devices”* view when an application is selected; context menu in the *“POUs”* view

"Name"	Name of POU

Table 79: "Type"

"Program"	
"Function Block"	● ☑ *"Extends"*: Specification or selection of a base function module in the sense of object-oriented programming. Specified with the EXTENDS keyword in the function block declaration ● ☑ *"Implements"*: Specification or selection of an interface in the sense of object-oriented programming. Specified with the IMPLEMENTS keyword in the function block declaration. When the POU is created, all methods are created which are defined via the interface. ● ☑ *"Final"*: Derived access is not allowed. This means that you cannot extend the function block with another function block. This allows for optimized code generation. ● ☑ *"Abstract"*: Identifies that the function block has a missing or incomplete implementation and cannot be instantiated. Abstract FBs are used exclusively as base function blocks and the implementation typically occurs in a derived FB. If a non-abstract function block is created, which in turn extends an abstract function block, then all abstract methods of the abstract basic function block are added to the new function block as (non-abstract) methods. ● *"Access specifier"* – *"PUBLIC"*: Corresponds to the specification of no access specifier. – *"INTERNAL"*: Access to the function block is restricted to the namespace (library). ● *"Method implementation language"*: When you select the *"Implements"* option, you can select an implementation language here for all method objects that CODESYS generates by means of the implementation of the interface. The *"Method implementation language"* does not depend on the implementation language of the function block.
"Function"	Note: Not available when *"Sequential Function Chart (SFC)"* is selected as the *"Implementation language"*. *"Return type:"*: Data type of the return value
"Implementation language"	Implementation language of the POU

See also

- *Chapter 1.4.1.20.2.18.2 "Object 'Function Block'" on page 883*
- *Chapter 1.4.1.20.2.18.1 "Object 'Program'" on page 882*
- *Chapter 1.4.1.20.2.18.3 "Object 'Function'" on page 886*
- *Chapter 1.4.1.8.22.1 "Extension of function blocks" on page 310*
- *Chapter 1.4.1.8.22.2 "Implementing interfaces" on page 312*

Object 'Program'

A program is a POU that supplies one or more values during execution. After execution of the program, all values are retained until the next execution. The order of calling the programs within an application is defined in task objects.

A program is added to the application or the project using the command *"Project ➔ Add Object ➔ POU"*. In the Device tree and in the *"POUs"* view the program POUs have the suffix *"(PRG)"*.

The editor of a program consists of the declaration part and the implementation part.

The uppermost line of the declaration part contains the following declaration:

```
PROGRAM <program>
```

Calling a program

Programs and function blocks can call a program. A program call is not permitted in a function. There are no instances of programs.

If a POU calls a program and values of the program change as a result, these changes are retained until the next program call. The values of the program are also retained even if the repeat call takes place by another POU. This differs from the call of a function block. When calling a function block only the values of the respective instance of the function block change. The changes only need to be observed if a POU calls the same instance again.

You can also set the input or output parameters for a program directly when calling.

Syntax: `<program>(<input variable> := <value>, <output value> => <value>):`

If you insert a program call via the input assistant and the *"Insert with arguments"* option in the input assistant is activated at the same time, CODESYS adds input and/or output parameters to the program call in accordance with the syntax.

Examples

Calls:

IL:

```
CAL      PLC_PRG(
         in1:= 2)
LD       PLC_PRG.out2
ST       erg
```

With assignment of the parameters:

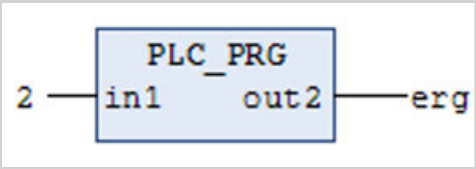
```
CAL      PLC_PRG(
         in1:= 2,
         out2=> erg)
```

ST:

```
PLC_PRG()
                    erg := PLC_PRG.out2;
```

With assignment of the parameters:

```
PLC_PRG(in1:=2, out1=>erg);
```



See also

- *Chapter 1.4.1.20.2.18 "Object 'POU'" on page 881*
- *Chapter 1.4.1.8.16 "Task Configuration" on page 292*

Object 'Function Block'

A function block is a POU that yields one or more values when executed.

The object is added to the application or the project by clicking *"Project ➔ Add Object ➔ POU"*. In the device tree or in the *"POUs"* view, function block POUs have the *"(FB)"* suffix.

It always calls a function block by means of an instance that is a copy of the function block.

The editor of a function block consists of the declaration part and the implementation part.

The values of the output variables and the internal variables remain unchanged after execution until the next execution. This means that the function block does not necessarily return the same output values for multiple calls with the same input variables.

In addition to the functionality described in IEC 61131-3, you can also use function blocks in CODESYS for the following functionalities of object-oriented programming:

- Extension of a function block
- Implementation of interfaces
- Methods
- Properties

The top line of the declaration part contains the following declaration:

```
FUNCTION_BLOCK <access specifier> <function block> | EXTENDS <function block> | IMPLEMENTS <comma-separated list of interfaces>
```

Calling a function block

The call is always made by means of an instance of the function block. When a function block is called, only the values of the respective instance change.

Declaration of the instance:

```
<instance> : <function block>;
```

You access a variable of the function block in the implementation part as follows:

```
<instance> . <variable>
```

NOTICE!

Note the following:

- You can access only input and output variables of a function block from outside the function block instance, not the internal variables.
- Access to a function block instance is restricted to the POU in which the instance is declared, unless you have declared the instance globally.
- You can assign the desired values to the function block variables when you call the instance.

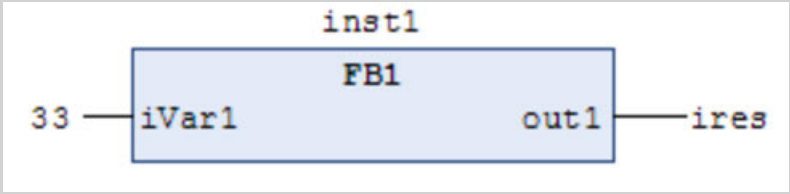
Example

Access to function block variables:

The function block `FB1` has the input variable `iVar1` of type `INT` and the output variable `out1`. In the following, the variable `iVar1` is called from the program `Prog`.

```
PROGRAM Prog
VAR
inst1:FB1;
END_VAR

inst1.iVar1 := 33;  (* FB1 is called and the value 33 is assigned
to the variable iVar1 *)

inst1();            (* FB1 is called, that's necessary for the
following access to the output variable *)

ires := inst1.out1  (* the output variable out1 of the FB1 is read
*)
```

In FBD:



Assigning variable values when calling:

In the textual languages IL and ST, you can assign values directly to input and/or output variables when you call the function block.

A value is assigned to an input variable with `:=` .

A value is assigned to an output variable with `=>` .

Example

The instance `CMD_TMR` of the timer function block is called with assignments for the input variables `IN` and `PT`. Then the output variable `Q` of the timer is assigned to the variable `A`.

```
PROGRAM PLC_PRG
VAR
    CMD_TMR : TOF;
END_VAR

CMD_TMR(IN := %IX5.1, PT := T#100MS);
A := CMD_TMR.Q;
```

When you insert a function block instance by means of the “Input Assistant” and select the “Insert with arguments” option in the “Input Assistant” dialog, CODESYS inserts the call with all input and output variables. Then you only have to insert the desired value assignment. In the example above, CODESYS inserts the call as follows: `CMD_TMR (IN:= ,PT:= , Q=> ).`

You can use the attribute `'is_connected'` *and a local variable to determine at the time of the call in the function block instance whether or not a specific input receives an external assignment.*

See also

- *Chapter 1.4.1.20.2.18 “Object 'POU'” on page 881*
- *Chapter 1.4.1.8.22.1 “Extension of function blocks” on page 310*
- *Chapter 1.4.1.8.22.2 “Implementing interfaces” on page 312*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.19.6.2.23 “Attribute 'is_connected'” on page 707*

Object 'Function'

A function is a POU that supplies precisely one data element when executed and whose call in textual languages can occur as an operator in expressions. The data element can also be an array or a structure.

The object is added to the application or the project by clicking *“Project ➔ Add Object ➔ POU”*. In the device tree or in the *“POUs”* view, function POUs have the *“(FUN)”* suffix.

NOTICE!
Functions have no internal status information, which means that functions do not save the values of their variables until the next call. Calls of a function with the same input variable values always supply the same output value. Therefore, functions must not use global variables and addresses!

The editor of a function consists of the declaration part and the implementation part.

The top line of the declaration part contains the following declaration:

```
FUNCTION <function> : <data type>
```

Below that, you declare the input and function variables.

The output variable of a function is the function name.

NOTICE!
If you declare a local variable in a function as `RETAIN`, this has no effect. In this case, CODESYS issues a compiler error.

NOTICE!
You cannot mix explicit and implicit parameter assignments in function calls in CODESYS V3. This means that you have to use either only explicit or only implicit parameter assignments in function calls. The order of the parameter assignments when calling a function is arbitrary.

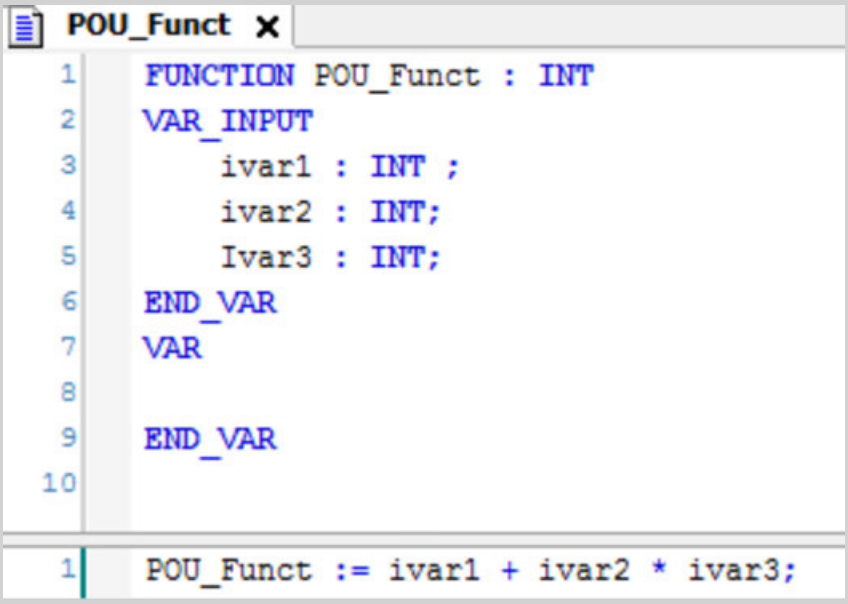
Calling a function

In ST, you can use the call of a function as an operand in expressions.

In SFC, you can use a function call only within step actions or transitions.

Examples

Function with declaration part and a line implementation code



Function calls:

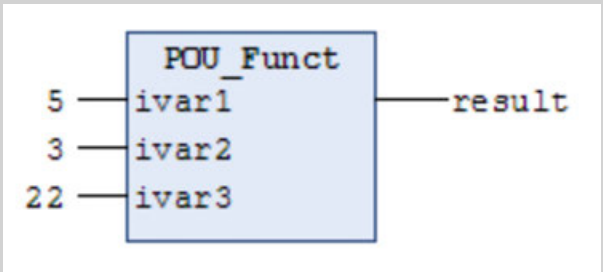
ST:

```
result := POU_Funct(5,3,22);
```

AWL:

LD	5
POU_Funct	3
	22
ST	result

FBD:



Functions with additional outputs

According to the IEC 61131-3 standard, functions can have additional outputs. You declare the additional outputs in the function between the keywords `VAR_OUTPUT` and `END_VAR`. The function is called according to the following syntax:

```
<function> (<function output variable1> => <output variable 1>,
<function output variable n> => <output variable n>)
```

Example

The `fun` function is defined with two input variables `in1` and `in2`. The output variable of the `fun` function is written to the locally declared output variables `loc1` and `loc2`.

```
fun(in1 := 1, in2 := 2, out1 => loc1, out2 => loc2);
```

See also

- *Chapter 1.4.1.20.2.18 “Object 'POU'” on page 881*

Object 'Interface'

Symbol:

Keyword: `INTERFACE`

An interface is a means of object-oriented programming. The object `ITF` describes a set of method and property prototypes. In this context, prototype means that the methods and properties contain only declarations and no implementation.

This allows different function blocks having common properties to be used in the same way. An object *“ITF”* is added to the application or the project with the command *“Project ➔ Add Object ➔ Interface”*.

Table 80: “Adding an interface”

“Inheritance”	
“Name”	Interface name
“Extends”	: Extends the interface that you enter in the input field or via the input assistant . This means that all methods of the interface that extend the new interface are also available in the new interface.

You can add the objects *“Interface property”* and *“Interface Method”* to the object *“ITF”*. Interface methods may contain only the declarations of input, output and input/output variables, but no implementation.

So that you can also use an interface in the program, there must be a function block that implements this interface.

This means:

- the function block contains the interface in its IMPLEMENTS list in its declaration part
- the function block contains an implementation for all methods and property prototypes of the interface

A function block can implement one or more interfaces. You can use the same method with identical parameters, but different implementation code in different function blocks.

Please note the following:

- You may not define variables within an interface. An interface has no implementation part and no actions. Only a collection of methods is defined, in which you may define only input, output and input/output variables.
- CODESYS always treats variables declared with the type of an interface as references.
- A function block that implements an interface must contain implementation code for the methods of the interface. You have named the methods exactly as in the interface and the methods contain the same input, output and input/output variables as in the interface.

NOTICE!

Interface references and online change

The following can happen with a compiler version < 3.4.1.0: if a function block changes its data because variables are added or deleted, or because the type of variables changes, then CODESYS copies all instances of the function block to a new memory location. In this case, however, an interface reference refers not to the new memory location, but still to the old one.

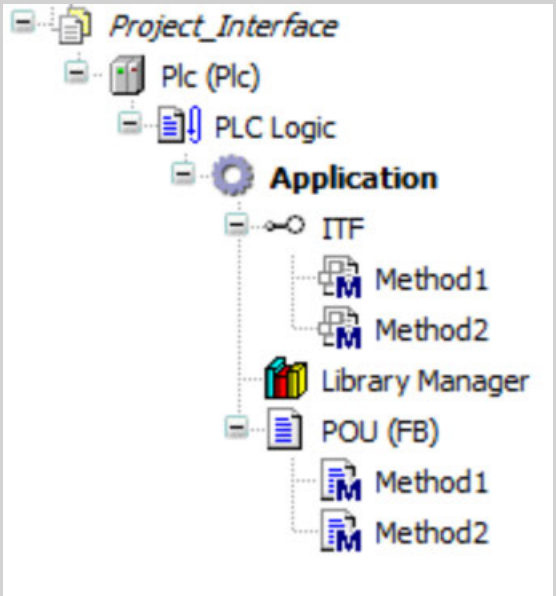
In case of compiler versions >= 3.4.1.0, CODESYS automatically re-addresses the interface references so that CODESYS also references the correct interface in case of an online change. CODESYS requires additional code and more time for this, so that jitter problems can occur depending on the number of objects concerned. Therefore, CODESYS displays the number of variables and interface references concerned before the execution of the online change and you can then decide whether the online change should be executed or aborted.

Example

Definition of an interface and its use in a function block

You have inserted the interface *"ITF"* below the application. The interface contains the methods *"Method1"* and *"Method2"*. *"ITF"*, *"Method1"* and *"Method2"* contain no implementation code. You insert the required variable declarations only in the declaration part of the methods.

If you subsequently insert a function block in the device tree that implements the interface *"ITF"*, CODESYS automatically also inserts the methods *"Method1"* and *"Method2"* under the function block. Here you can implement function-block-specific code in the methods.



- *Chapter 1.4.1.8.22.2 "Implementing interfaces" on page 312*
- *Chapter 1.4.1.8.22.3 "Extending interfaces" on page 314*

Object 'Method'

Symbol:

Keyword: `METHOD`

Methods are an extension of the IEC 61131-3 standard and a tool for object-oriented programming which is used for data encapsulation. A method contains a declaration and an implementation. However, unlike a function, a method is not an independent POU, and it is subordinated to a function block or program. A method can access all valid variables of the superordinate POU.

You can use interfaces for the organization of methods.

You can add a method below a program or a function block. Click *“Project ➔ Add Object ➔ Method”* to open the *“Add Method”* dialog.

Declaration

- The variables of a method contain temporary data that are valid only during the execution of the method (stack variables). All variables that are declared and implemented in a method are reinitialized each time the method is called.
- Like functions, methods can have additional outputs. You have to assign these additional outputs in the method call.
- Depending on the declared access specifier, a method can be called only within its own namespace (`INTERNAL`), only within its own POU and its derivatives (`PROTECTED`), or only within its own POU (`PRIVATE`). For `PUBLIC`, the method can be called from anywhere.

Interface methods can have declared input, output, and `VAR_IN_OUT` variables, but do not contain an implementation.

See also

- *Chapter 1.4.1.20.2.18.6 “Object 'Interface Method'” on page 894*

Implementation

- Access to function block instances or program variables is allowed in the implementation of the method.
- The `THIS` pointer allows for access to its own function block instance. Therefore, the pointer is allowed only in methods that are assigned to a function block.
- A method cannot access `VAR_TEMP` variables of the function block.
- A method can call itself recursively.

NOTICE!
When you copy a method below a POU and add it below an interface, or move the method there, the contained implementation is removed automatically.

Calling a method

Syntax for calls:

```
<return value variable> := <POU name> . <method name> ( <method input
name> := <variable name> (, <further method input name> := <variable
name> )* );
```

For the method call, you assign transfer parameters to the input variables of the method. Respect the declaration when doing this. It is enough to specify the names of the input variables without paying attention to their order in the declaration.

Example

Declaration

```
METHOD PUBLIC DoIt : BOOL
VAR_INPUT
    iInput_1 : DWORD;
    iInput_2 : DWORD;
    sInput_3 : STRING(12);
END_VAR
```

Call

```
bFinishedMethod := fbInstance.DoIt(sInput_3 := 'Hello World ',
iInput_2 := 16#FFFF, iInput_1 := 16);
```

When the method is called, the return value of the method is assigned, for example, to variables declared locally. When you omit the names of the input variables, you have to pay attention to the declaration order.

Example

Declaration

```
METHOD PUBLIC DoIt : BOOL
VAR_INPUT
    iInput_1 : DWORD;
    iInput_2 : DWORD;
    sInput_3 : STRING(12);
END_VAR
```

Call

```
bFinishedMethod := fbInstance.DoIt( 16, 16#FFFF,'Hello World ');
```

Recursive method call

Within the implementation, a method can call itself, either directly by means of the `THIS` pointer, or by means of a local variable for the assigned function block.

- `THIS^. <method name> ( <parameter transfer of all input and output variables> )`
  Direct call of the relevant function block instance with the `THIS` pointer
- `VAR fb_Temp : <function block name>; END_VAR`
  Call by means of a local variable of the method that temporarily instantiates the relevant function block

A compiler warning is issued for a recursive call. If the method is provided with the pragma `{attribute 'estimated-stack-usage' := '<sstimated_stack_size_in_bytes>'}`, then the compiler warning is suppressed. For an implementation example, see the "Attribute 'estimated-stack-usage'" chapter.

To call methods recursively, it is not enough to specify only the method name. If only the method name is specified, then a compiler error is issued: *“Program name, function or function block instance expected instead of”*

See also

- *Chapter 1.4.1.8.22.4 “Calling methods” on page 314*
- *Chapter 1.4.1.19.6.2.13 “Attribute 'estimated-stack-usage'” on page 695*
- *Chapter 1.4.1.19.2.15 “THIS” on page 539*

Special methods of a function block

FB_Init	Declarations automatically implicit, but explicit declaration also possible Contains initialization code for the function block, as is defined in the declaration part of the function block
FB_Reinit	Explicit declaration is necessary. Call after the instance of the function block was copied (as during an online change). It reinitializes the new instance module.
FB_Exit	Explicit declaration is necessary. Call for each instance of the function block before a new download or a reset or during an online change for all shifted or deleted instances.
Properties	Provides Set and/or Get accessor methods.

See also

- *Chapter 1.4.1.19.10 “Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'” on page 748*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.20.2.18.7 “Object 'Interface Property'” on page 894*

Dialog 'Add Method'

Function: Defines a method below the selected POU when the dialog is closed.

Call: Menu bar: *“Project ➔ Add Object ➔ Method”*; context menu

Requirement: A program (PRG) or a function block (FUNCTION_BLOCK) is selected in the *“POUs”* view or the *“Devices”* view.

The interface of a method inserted below a basic function block is copied when a method with the same name is inserted below a derived function block.

“Name”	Example: meth_DoIt. The standard methods FB_Init and FB_Exit are offered in a list box if they are not already inserted below the POU. If it is a derived function block, then the list box also offers all of the methods of the basic function block.
“Return type”	Default data type or structured data type of return value Example: BOOL
“Implementation language”	Example: *“Structured Text (ST)”*
“Access specifier”	Controls access to data. ● *“PUBLIC”* or not specified: Access is not restricted. ● *“PRIVATE”*: Access is restricted to the program, function block, or GVL. The object is marked as (private) in the POU or device view. The declaration contains the keyword PRIVATE. ● *“PROTECTED”*: Access is restricted to the program, function block, or GVL with its derivations. The declaration contains the keyword PROTECTED. The object is marked as (protected) in the POU or device view. ● *“INTERNAL”*: Access to the method is restricted to the namespace (library). The object is marked as (internal) in the POU or device view. The declaration contains the keyword INTERNAL.
“Abstract”	☑: Identifies that the method does not have an implementation and the implementation is provided by the derived FB
“Add”	Adds a new method below the selected object.

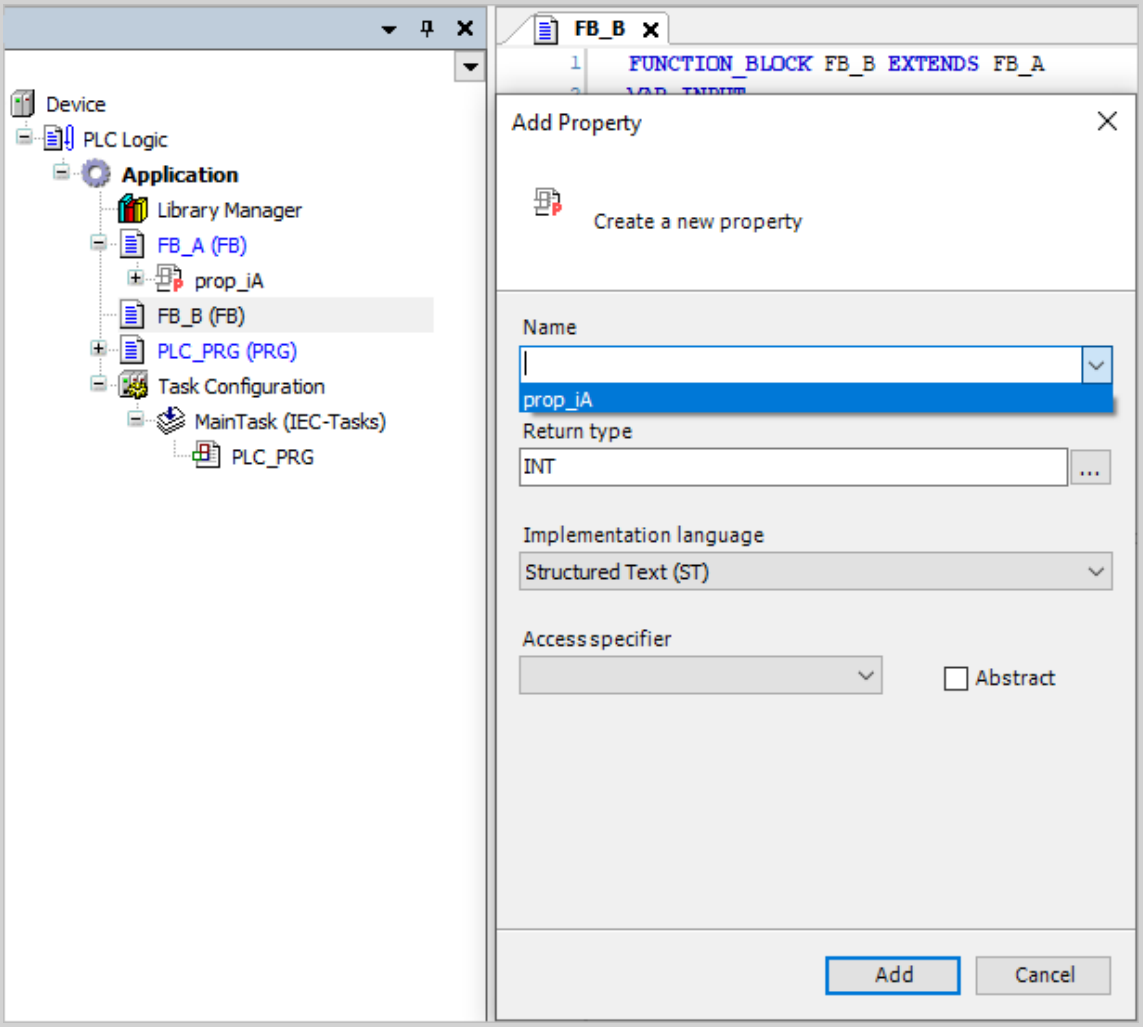
Input support when generating inheriting POUs

When you doing object-oriented programming and using the inheritance (keyword `EXTENDS`) of POUs, you can get support as follows:

When you insert an action, a property, a method, or a transition below a POU derived from a base POU, the *"Add …"* dialog opens. Then the input field for the name extends to a list box. The list box contains a valid selection from the actions, properties, methods, or transitions available in the base POU. Now you can, for example, easily accept a method of the base POU and then adapt it to the derived function of the POU.

Methods and properties with the access modifier `PRIVATE` are not listed here because they are also not inherited. Methods and properties with the access modifier `PUBLIC` automatically get a blank access modifier field when accepting into the derived POU, which means the same thing functionally.

Example



See also

- *Chapter 1.4.1.8.22.1 "Extension of function blocks" on page 310*
- *Chapter 1.4.1.8.22 "Object-Oriented Programming" on page 310*
- *Chapter 1.4.1.20.2.18.9 "Object 'Action'" on page 901*
- *Chapter 1.4.1.20.2.18.8 "Object 'Property'" on page 897*
- *Chapter 1.4.1.20.2.18.5 "Object 'Method'" on page 889*
- *Chapter 1.4.1.20.2.18.10 "Object 'Transition'" on page 903*

See also

- *Chapter 1.4.1.8.22.2 "Implementing interfaces" on page 312*
- *Chapter 1.4.1.19.1.3.2 "ST editor in online mode" on page 463*

- *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*
- *Chapter 1.4.1.19.2.9 “Instance variables - VAR_INST” on page 533*

Object 'Interface Method'

Symbol: 

This object is used for object-oriented programming.

The object *“Interface Method”* is added to an interface via the command *“Project ➔ Add Object”*.

If a method is inserted underneath an interface, you can add and instance only variable declarations (input, output and input/output variables) in this method.

You can only add program code to the method if a function block 'implements' the interface to which the method belongs. CODESYS then inserts the method underneath the function block.

See also

- *Chapter 1.4.1.20.2.18.4 “Object 'Interface'” on page 888*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.8.22.2 “Implementing interfaces” on page 312*

Object 'Interface Property'

Symbol: 

Interface properties are an extension of the IEC 61131-3 standard and a tool for object-oriented programming. An interface property declares the accessor methods `Get` and `Set` (no implementation code). Therefore, a function block that implements an interface also inherits their interface properties.

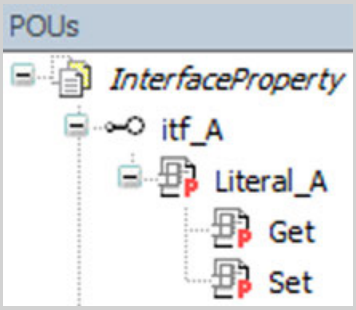
You can add an interface property to the device tree for an interface. Then an interface is extended with the accessor methods `Get` and `Set`. The `Get` accessor is for read access. The `Set` accessor is for write access. You can delete an unneeded accessor. Click *“Project ➔ Add Object ➔ Interface Property”* to add an accessor. The *“Add Interface Property”* dialog opens.

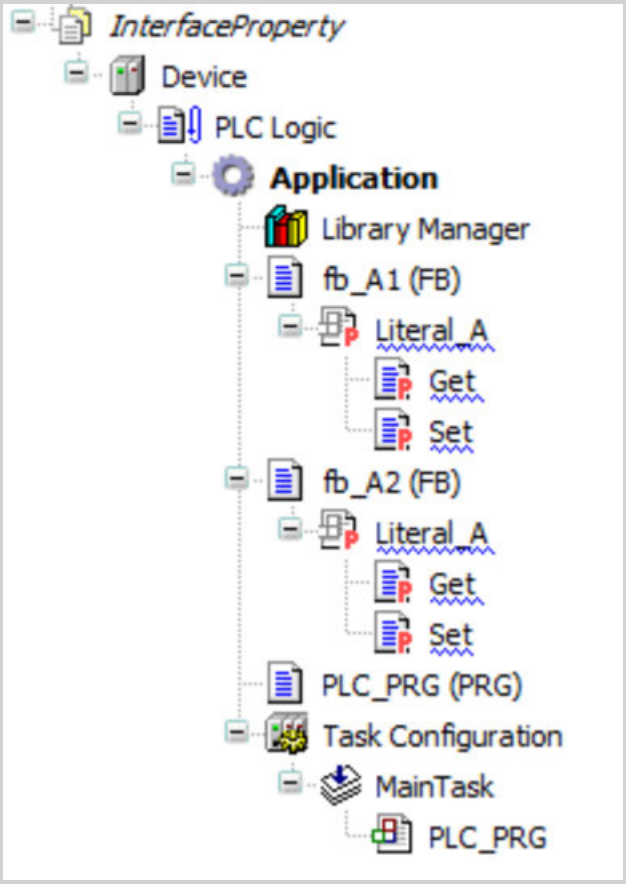
See also

- *Chapter 1.4.1.20.2.18.4 “Object 'Interface'” on page 888*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*

Example

Declaration and implementation of the interface property `Literal_A`





This interface `itf_A` has the property `Literal_A` with the accessor methods `Get` and `Set`.

The function blocks `fb_A1` and `fb_A2` implement the interface `itf_A` and therefore inherit its interface property. Each FB has its own implementation.

Interface `itf_A`

```
INTERFACE itf_A
VAR
END_VAR
PROPERTY Literal_A : STRING
```

FB `fb_A1`

```
FUNCTION_BLOCK fb_A1 IMPLEMENTS itf_A
VAR
    str_1 : STRING;
    str_2 : STRING;
    iCnt : INT;
END_VAR
iCnt := iCnt + 1;

str_1 := 'Function block A1';
```

Accessor fb_A1.Literal_A.Get

```
VAR
END_VAR
Literal_A := CONCAT (str_1,' and property.');
```

Accessor fb_A1.Literal_A.Set

```
VAR
END_VAR
str_2 := Literal_A;
```

FB fb_A2

```
FUNCTION_BLOCK fb_A2 IMPLEMENTS itf_A
VAR
    str_1 : STRING;
    str_2 : STRING;
    iCnt : INT;
END_VAR

iCnt := iCnt + 1;
str_1 := 'Function block A2';
```

Accessor fb_A2.Literal_A.Get

```
VAR
END_VAR
Literal_A := str_1;
```

Accessor fb_A2.Literal_A.Set

```
VAR
END_VAR
str_2 := Literal_A;
```

Program PLC_PRG

```
PROGRAM PLC_PRG
VAR
    iCnt : INT;
    my_1 : fb_A1;
    my_2 : fb_A2;
    strName_1 : STRING;
    strName_2: STRING;
END_VAR

iCnt := iCnt + 1;
my_1();
my_2();
strName_1:= my_1.Literal_A;
strName_2:= my_2.Literal_A;
my_1.Literal_A := 'Hello 1';
my_2.Literal_A := 'World 2';
```

This leads to the following monitoring of `PLC_PRG` when the application is in runtime mode:

Expression	Type	Value
iCnt	INT	-25848
my_1	fb_A1	
str_1	STRING	'Function block A1'
str_2	STRING	'Hello 1'
my_2	fb_A2	
str_1	STRING	'Function block A2'
str_2	STRING	'World 2'
iCnt	INT	-31303
strName_1	STRING	'Function block A1 and property.'
strName_2	STRING	'Function block A2'

Object 'Property'

Symbol:

Keyword: `PROPERTY`

Properties are an extension of the IEC 61131-3 standard and a tool for object-oriented programming.

Properties are used for data encapsulation because they allow for external access to data and act as filters at the same time. For this purpose, a property provides the accessor methods `Get` and `Set` which allows for read and write access to the data of the instance below the property.

You can add a property with accessor methods below a program, a function block, or a global variable list. Click *"Project ➔ Add Object ➔ Property"* to open the *"Add Property"* dialog.

> *You can add an interface property below an interface.*
>
> *When you copy a property that is inserted below a POU and add it below an interface, or if you move the property there, the included implementations are removed automatically.*

See also

- *Chapter 1.4.1.20.2.18.7 "Object 'Interface Property'" on page 894*

Dialog 'Add Property'

Function: Creates a new property below the selected POU when the dialog is closed.

Call: Menu bar: *"Project ➔ Add Object ➔ Property"*; context menu

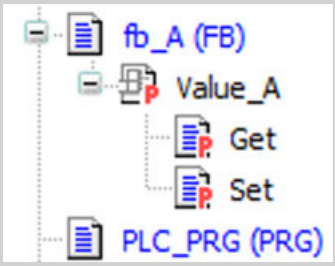
Requirement: A program (`PRG`), a function block (`FUNCTION_BLOCK`), or a global variable list (`GVL`) is selected in the *"POUs"* view or the *"Devices"* view.

"Name"	Name (identifier) of the property Example: `prop_iA`
"Return type"	Default type or structured type of return value Example: `INT`
"Implementation language"	Example: *"Structured Text (ST)"*

“Access specifier”	Controls access to data
“PUBLIC” or unspecified	Access is not restricted.
“PRIVATE”	Access is restricted to the program, function block, or GVL. The object is marked as `(private)` in the POU or device view. The declaration contains the keyword `PRIVATE`.
“PROTECTED”	Access is restricted to the program, function block, or GVL with its derivations. The object is marked as `(protected)` in the POU or device view. The declaration contains the keyword `PROTECTED`.
“INTERNAL”	Access is restricted to the namespace (library). The object is marked as `(internal)` in the POU or device view. The declaration contains the keyword `INTERNAL`.
“Abstract”	☑: Identifies that the property does not have an implementation and the implementation is provided by the derived FB
“Add”	Adds a new property be low the selected object and below that the accessor methods `Get` and `Set` Note: When you select a property, you can also add a previously removed accessor explicitly by clicking *“Add Object”*.

Editor 'Property' You can program the data access in the editor. The code can contain additional local variables. However, it must not contain any additional input variables or (as opposed to a function or method) output variables.

Example



Function block FB_A

```
FUNCTION_BLOCK FB_A
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iA : INT;
END_VAR

iA := iA + 1;
```

Property `prop_iA`

```
PROPERTY PUBLIC prop_iA : INT
```

Accessor method FB_A.prop_iA. Get

```
prop_iA := iA;
```

Accessor method FB_A.prop_iA. Set

```
iA := prop_iA;
```

```
PROGRAM PLC_PRG
VAR
    fbA : FB_A;
    iVar: INT;
END_VAR

fbA();
IF fbA.prop_iA > 500 THEN
    fbA.prop_iA := 0;
END_IF
iVar := fbA.prop_iA;
```

`Get` and `Set` accessors

The call of the `Set` accessor is written to the property. Then it is used in the same way as an input parameter. When the Get accessor is called, the property is read. It is used in the same way as an output parameter. Access is restricted in each case by means of access modifiers (qualifiers). As a result, the objects are identified accordingly.

When a property is accessed as read only or write only, you can delete the unneeded accessors.

You can add accessors explicitly by selecting a property and clicking *“Add Object”*. A dialog opens, either *“Add Get accessor”* or *“Add Set accessor”*. There you can set the implementation language and the access.

Table 81: Dialog “Add Get (Set) Accessor”

“Implementation language”	Example: *“Structured Text (ST)”*

"Access specifier"	Qualifier for the declaration part
`PUBLIC` or unspecified	Access is not restricted.
`PRIVATE`	Access is restricted to the program, function block, or GVL. The object is marked as `(private)` in the POU or device view. The declaration contains the keyword.
`PROTECTED`	Access to the property is restricted to the program, function block, or GVL and its derivations. The declaration contains the keyword. The object is marked as `(protected)` in the POU or device view.
`INTERNAL`	Access to the method is restricted to the namespace (the library). The object is marked as `(internal)` in the POU or device view. The declaration contains the keyword.
"Add"	Adds the accessor methods `Get` or `Set` below the selected property.

Monitoring of properties in online mode

The following pragmas are provided for the monitoring of properties in online mode. You insert them at the top position of the property definition:

- `{attribute 'monitoring' := 'variable'}`
  Each time the property is accessed, CODESYS saves the actual value to a variable and displays the value of this variable. This value can become outdated if no more access to the property takes place in the code.
- `{attribute 'monitoring' := 'call'}`
  Each time the value is displayed, CODESYS calls the code of the `Get` accessor. If this code contains a side effect, then the monitoring executes the side effect.

You can monitor a property with the help of the following functions.

- Inline monitoring
  Requirement: The *"Enable inline monitoring"* option is selected in the *"Text Editor"* category of the *"Options"* dialog.
- Watch List

See also

- *Chapter 1.4.1.8.22.4 "Calling methods" on page 314*
- *Chapter 1.4.1.19.6.2.25 "Attribute 'monitoring'" on page 709*

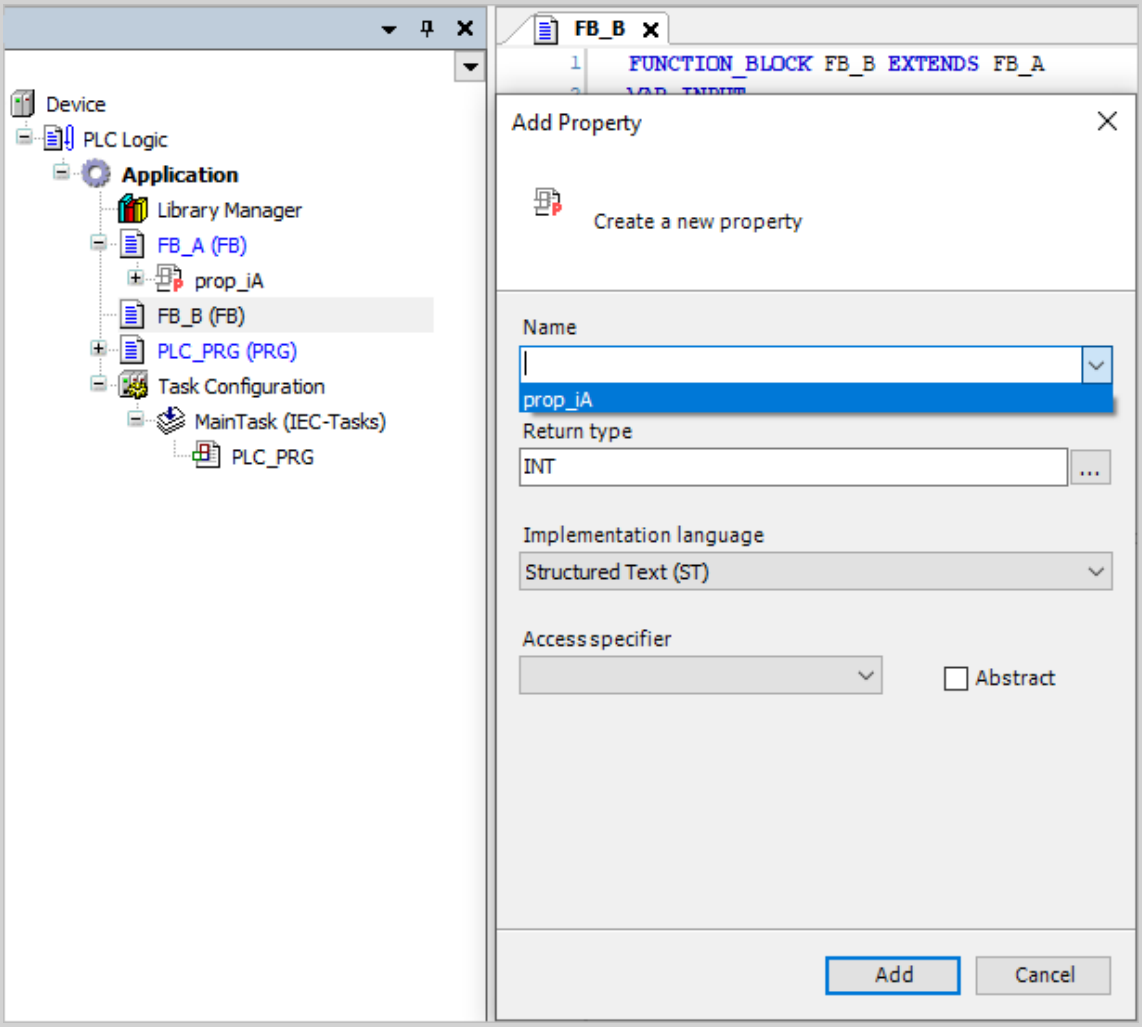
Input support when generating inheriting POUs

When you doing object-oriented programming and using the inheritance (keyword `EXTENDS`) of POUs, you can get support as follows:

When you insert an action, a property, a method, or a transition below a POU derived from a base POU, the *"Add …"* dialog opens. Then the input field for the name extends to a list box. The list box contains a valid selection from the actions, properties, methods, or transitions available in the base POU. Now you can, for example, easily accept a method of the base POU and then adapt it to the derived function of the POU.

Methods and properties with the access modifier `PRIVATE` are not listed here because they are also not inherited. Methods and properties with the access modifier `PUBLIC` automatically get a blank access modifier field when accepting into the derived POU, which means the same thing functionally.

Example



See also

- *Chapter 1.4.1.8.22.1 “Extension of function blocks” on page 310*
- *Chapter 1.4.1.8.22 “Object-Oriented Programming” on page 310*
- *Chapter 1.4.1.20.2.18.9 “Object 'Action'” on page 901*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.20.2.18.10 “Object 'Transition'” on page 903*

Object 'Action'

Symbol:

Implement more program code in an action. You can implement this program code as the base implementation in another language. The base implementation is a function block or a program where you inserted the action.

An action does not have its own declaration and it works with the data from the base implementation. This means that the action uses the input and output variables and the local variables from its base implementation.

Add an *“Action”* to a function block or program by clicking *“Project ➔ Add Object ➔ Action”*.

Table 82: “Add Action”

“Name”	Name of the action
“Implementation language”	List box of implementation language

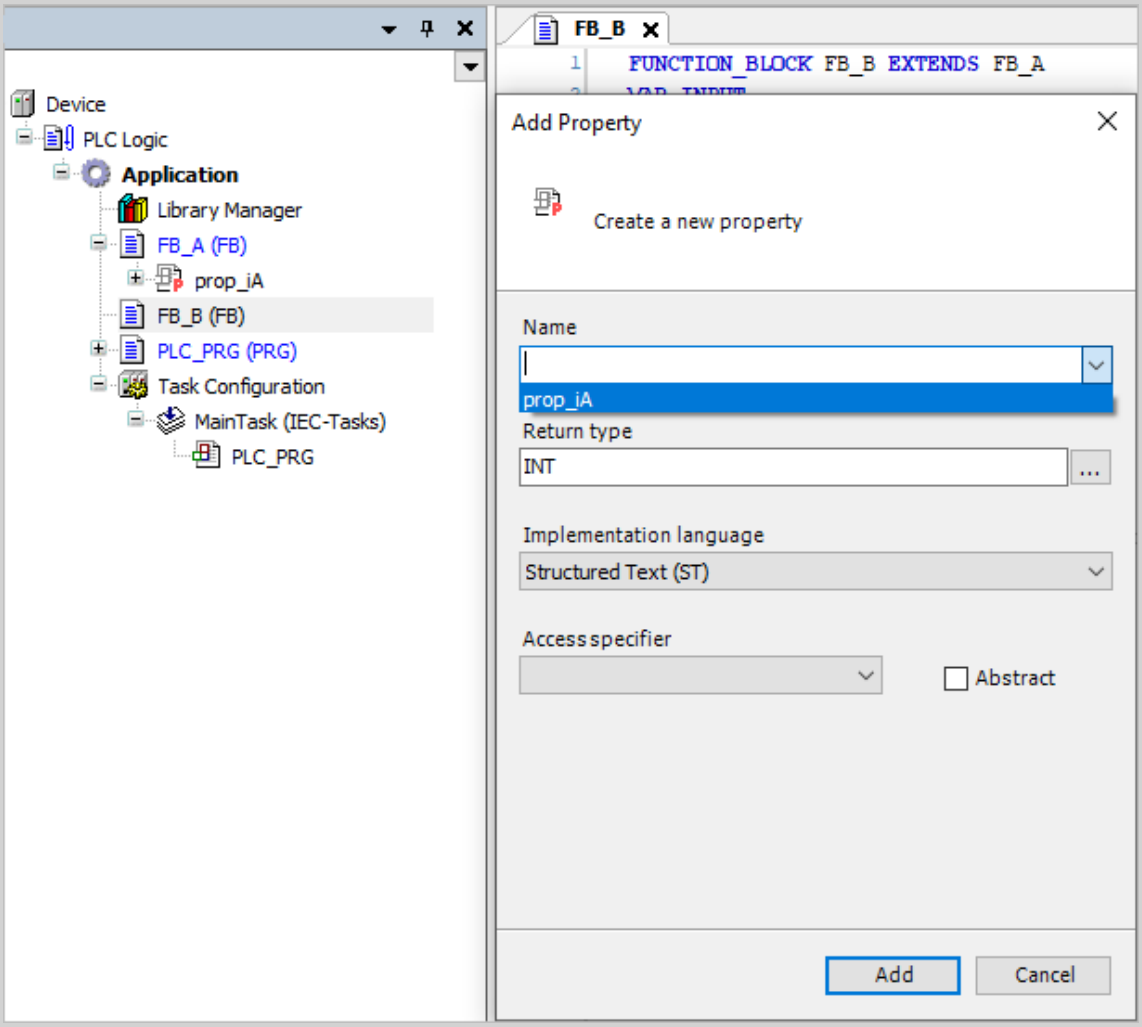
Input support when generating inheriting POUs

When you doing object-oriented programming and using the inheritance (keyword `EXTENDS`) of POUs, you can get support as follows:

When you insert an action, a property, a method, or a transition below a POU derived from a base POU, the *“Add …”* dialog opens. Then the input field for the name extends to a list box. The list box contains a valid selection from the actions, properties, methods, or transitions available in the base POU. Now you can, for example, easily accept a method of the base POU and then adapt it to the derived function of the POU.

Methods and properties with the access modifier `PRIVATE` are not listed here because they are also not inherited. Methods and properties with the access modifier `PUBLIC` automatically get a blank access modifier field when accepting into the derived POU, which means the same thing functionally.

Example



See also

- *Chapter 1.4.1.8.22.1 “Extension of function blocks” on page 310*
- *Chapter 1.4.1.8.22 “Object-Oriented Programming” on page 310*
- *Chapter 1.4.1.20.2.18.9 “Object 'Action'” on page 901*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.20.2.18.10 “Object 'Transition'” on page 903*

Calling an action

Syntax:

`<program>.<action>` or `<FB instance>.<action>`

To call an action from only within the base implementation, you only have to provide the action name.

Examples

Calling a *"Reset"* action from another POU The call is not executed from the base implementation.

Declaration:

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

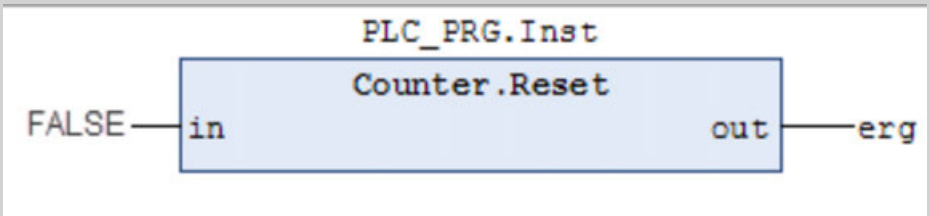
Calling a *"Reset"* action from an IL POU

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

Calling a *"Reset"* action from an ST POU

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

Calling a *"Reset"* action from an FBD POU



Actions are used frequently in the SFC implementation language.

See also

- *Chapter 1.4.1.19.1.4.8.2 "SFC Element 'Action'" on page 488*

Object 'Transition'

Symbol:

The object can be used as a transition element in a program block implemented in SFC.

See also

- *Chapter 1.4.1.19.1.4.8.1 "SFC elements 'Step' and 'Transition'" on page 486*

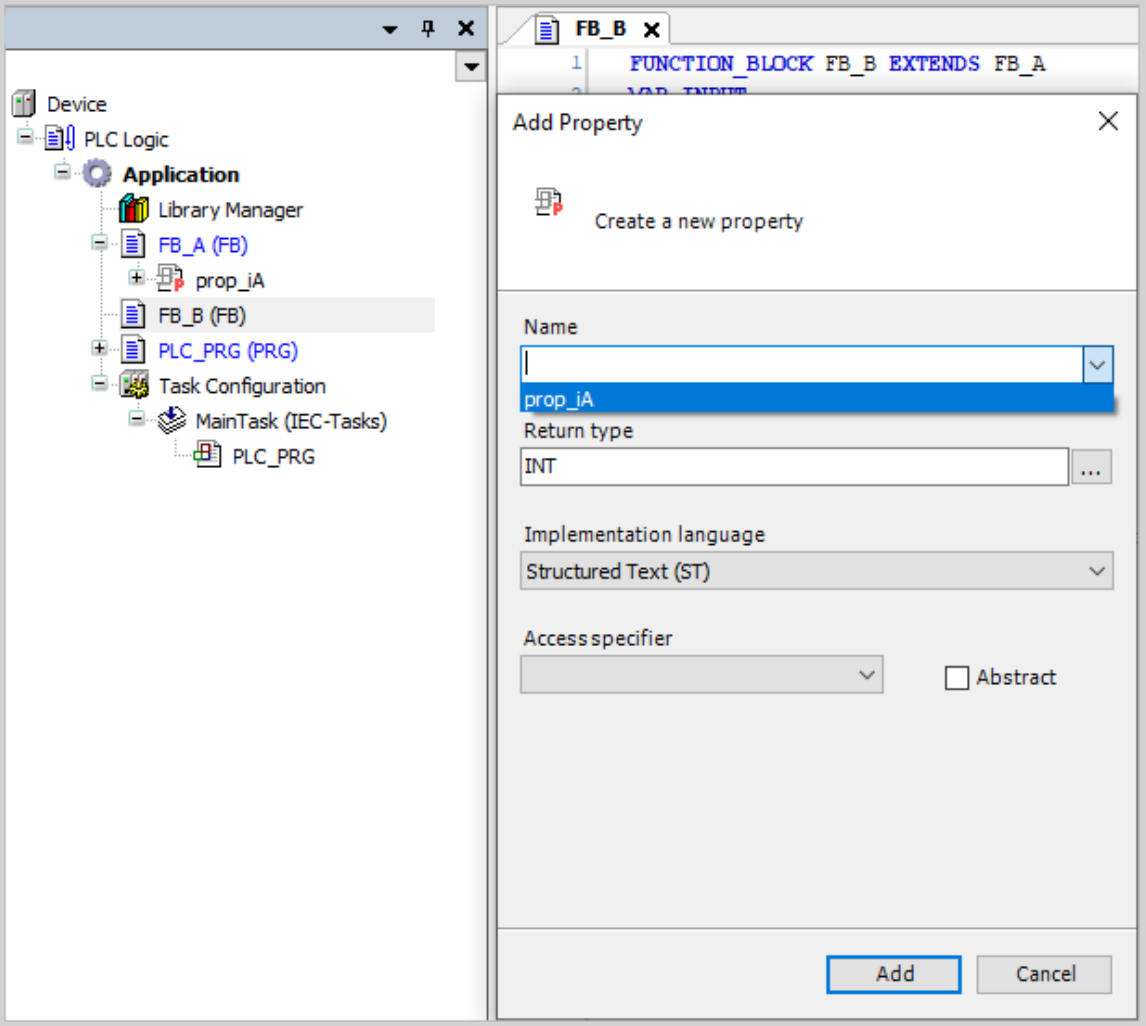
Input support when generating inheriting POUs

When you doing object-oriented programming and using the inheritance (keyword `EXTENDS`) of POUs, you can get support as follows:

When you insert an action, a property, a method, or a transition below a POU derived from a base POU, the *"Add …"* dialog opens. Then the input field for the name extends to a list box. The list box contains a valid selection from the actions, properties, methods, or transitions available in the base POU. Now you can, for example, easily accept a method of the base POU and then adapt it to the derived function of the POU.

Methods and properties with the access modifier `PRIVATE` are not listed here because they are also not inherited. Methods and properties with the access modifier `PUBLIC` automatically get a blank access modifier field when accepting into the derived POU, which means the same thing functionally.

Example



See also

- *Chapter 1.4.1.8.22.1 “Extension of function blocks” on page 310*
- *Chapter 1.4.1.8.22 “Object-Oriented Programming” on page 310*
- *Chapter 1.4.1.20.2.18.9 “Object 'Action'” on page 901*
- *Chapter 1.4.1.20.2.18.8 “Object 'Property'” on page 897*
- *Chapter 1.4.1.20.2.18.5 “Object 'Method'” on page 889*
- *Chapter 1.4.1.20.2.18.10 “Object 'Transition'” on page 903*

Object 'POUs for Implicit Checks'

You can add these special POUs to an application to equip them with implicit monitoring functions. At runtime, these functions check the limits of arrays or subrange types, the validity of pointer addresses, and division by zero. Please note: This option can be disabled for devices that are already equipped with these kinds of monitoring blocks by a special implicit library.

The command *“Add Object ➔ POU for Implicit Checks”* is used for adding to the application. The command opens the *“Add POU for Implicit Checks”* dialog where you can select a monitoring function type (see table below). Depending on the monitoring function, you have to adapt the implementation code or create it yourself from scratch.

To prevent multiple inclusions, monitoring functions that have already been inserted are disabled in the *“Add POU for Implicit Checks”* dialog.

NOTICE!

To get the feature for monitoring functions, do not edit their declaration part. However, you are permitted to add local variables.

After removing an implicit monitoring function (example: `Check Bounds`) from the project, only a download is possible, not an online change. A corresponding message is issued.

By default, CODESYS does not run implicit checks for function blocks from libraries used in the application. However, you can extend the check to the libraries by opening the “Properties” dialog of the application and specifying the compiler definition `checks_in_libs` in the “Compiler-Defines” field in the “Build” tab. This definition affects implementation libraries (`*.library`) only, not protected libraries (`*.compiled-library`).

You can use the "no_check" attribute to deactivate the check for special POUs in the project.

Table 83: “Available Functions”

Monitoring function	Type
“Check Bounds”	*“Bound Checks”* Appropriate handling of bound violations; such handling includes setting flags or changing field indices.
“CheckDivDInt”	*“Division checks”*: Monitors the divisor value to avoid division by zero.
“CheckDivLInt”	
“CheckDivReal”	
“CheckDivLReal”	
“CheckRangeSigned”	*“Range checks”*: Monitors the range limit of a subrange type in runtime mode. Valid for data types `DINT`/`UDINT`.
“CheckRangeUnsigned”	
“CheckLRangeSigned”	*“L-range checks”*: Monitors the range limit of a subrange type in runtime mode. Valid for data types `LINT`/`ULINT`.
“CheckLRangeUnsigned”	
“CheckPointer”	*“Pointer checks”* You are responsible for filling in this function completely with implementation code. Refer to the help page for "POU 'CheckPointer'". The function should monitor whether the passed pointer reference a valid memory address, and whether the orientation of the referenced memory area matches the variable type to which the pointer refers. If both conditions are fulfilled, then the pointer is returned. If not, then `CheckPointer` should complete an appropriate error handling. `CheckPointer` monitors the same way as variables of type `REFERENCE TO`.

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*
- *Chapter 1.4.1.20.2.19.1 “POU 'CheckBounds'” on page 906*
- *Chapter 1.4.1.20.2.19.2 “POU 'CheckDivInt'” on page 909*
- *Chapter 1.4.1.20.2.19.3 “POU 'CheckDivLInt'” on page 909*

- *Chapter 1.4.1.20.2.19.5 “POU 'CheckDivLReal'” on page 911*
- *Chapter 1.4.1.20.2.19.4 “POU 'CheckDivReal'” on page 910*
- *Chapter 1.4.1.20.2.19.10 “POU 'CheckPointer'” on page 917*
- *Chapter 1.4.1.20.2.19.6 “POU 'CheckRangeSigned'” on page 912*
- *Chapter 1.4.1.20.2.19.8 “POU 'CheckRangeUnsigned'” on page 915*
- *Chapter 1.4.1.20.2.19.7 “POU 'CheckLRangeSigned'” on page 914*
- *Chapter 1.4.1.20.2.19.9 “POU 'CheckLRangeUnsigned'” on page 916*
- *Chapter 1.4.1.20.4.10.4 “Dialog 'Properties' - 'Build'” on page 1159*
- *Chapter 1.4.1.19.6.2.27 “Attribute 'no_check'” on page 712*

POU 'CheckBounds'

Functions for Bound Checks: CheckBounds

The task of this monitoring function is to handle bound violations appropriately. Examples of reactions to violations include setting error flags and changing the value of the array index. The check is performed only for one variable array index. An incorrect constant array index causes a compiler error. CODESYS calls the function implicitly when values are assigned to an `ARRAY` variable.

After inserting the function, you receive automatically generated code in the declaration and implementation parts. See below.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration part. However, you are permitted to add local variables.

Declaration part

```
// Automatically generated code: DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
      index, lower, upper: DINT;
END_VAR
```

Implementation

```
// This automatically generated code is a suggested implementation.
IF  index < lower THEN
      CheckBounds := lower;
ELSIF  index > upper THEN
      CheckBounds := upper;
ELSE
      CheckBounds := index;
END_IF

(* It is also possible to set a breakpoint, log messages or e.g. to
halt on an exception:
Add CmpApp.library, SysExcept.library and SysTypes2_Itf as newest.
Declaration:
VAR
    _pApp : POINTER TO CmpApp.APPLICATION;
    _result   : SysTypes.RTS_IEC_RESULT;
END_VAR

Implementation:
_pApp := AppGetCurrent(pResult:=_result);
IF  index < lower THEN
        CheckBounds := lower;
        IF _pApp <> 0 THEN
                AppGenerateException(pApp:=_pApp,
ulException:=RtsExceptions.RTSEXCPT_ARRAYBOUNDS);
        END_IF
ELSIF  index > upper THEN
        CheckBounds := upper;
        IF _pApp <> 0 THEN
                AppGenerateException(pApp:=_pApp,
```

```
ulException:=RtsExceptions.RTSEXCPT_ARRAYBOUNDS);
        END_IF
ELSE
        CheckBounds := index;
END_IF
*)
```

When the *"CheckBounds"* function is called, it receives the following input parameters:

- `index`: Index of the array element
- `lower`: Lower limit of the array range
- `upper`: Upper limit of the array range

The return value is the index of the array element, as long as it is within a valid range. If not, then the CODESYS returns either the upper or lower limit, depending on which threshold was violated.

Example: Correction of the access to an array outside the defined array bounds

In the sample program below, the index falls short of the defined lower limit of the `a` array.

```
PROGRAM PLC_PRG
VAR
 a: ARRAY[0..7] OF BOOL;
    b: INT:=10;
END_VAR

a[b]:=TRUE;
```

In this example, the `CheckBounds` function causes `a` to change the upper limit of the array range index to `10`. The value `TRUE` is assigned then to the element `a[7]`. In this way, the function corrects array access outside of the valid array range.

Example: Output of an exception when array limits are violated.

Add the following libraries in the library manager of the application:

- `CmpApp.library` and `SysExcept.library` as placeholder libraries
- `SysTypes2_Itfs.library` with *"Newest version always"*

Add a *"CheckBounds"* object below the application and modify the specified code as shown below.

Declaration part

```
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper: DINT;
END_VAR
VAR
    _pApp : POINTER TO CmpApp.APPLICATION;
    _Result   : ISystypes2.RTS_IEC_RESULT;
END_VAR
```

Implementation part

```
// This automatically generated code is a suggested implementation.
_pApp := AppGetCurrent(pResult := _Result);
IF  index < lower THEN
    CheckBounds := lower;
    IF _pApp <> 0 THEN
        AppGenerateException(pApp := _pApp, ulException :=
RtsExceptions.RTSEXCPT_ARRAYBOUNDS);
    END_IF
ELSIF  index > upper THEN
    CheckBounds := upper;
    IF _pApp <> 0 THEN
       AppGenerateException(pApp:=_pApp,
ulException:=RtsExceptions.RTSEXCPT_ARRAYBOUNDS);
    END_IF
ELSE
    CheckBounds := index;
END_IF
```

Program a *"MAIN_PRG"* object below the application with the contents shown below.

```
PROGRAM MAIN_PRG
VAR
    xInit    : BOOL;
    arData   : ARRAY[0..7] OF BYTE;
    i        : INT;
    dwAdr       : DWORD;
END_VAR

IF NOT xInit THEN
    // Required for CheckBounds
    xInit  := TRUE;
END_IF

// Set i to a value > 7 or < 0
// Generates an exception in CheckBounds, user-defined
arData[i] := 11;
```

When you load and start this application, an exception will be thrown when array bounds are violated. Processing stops in *"CheckBounds"* so that the type of error can be detected.

See also

- *Chapter 1.4.1.8.21 "Using POUs for implicit checks" on page 309*
- *Chapter 1.4.1.20.2.8.8 "Tab 'Log'" on page 848*

POU 'CheckDivInt'

Functions for preventing division by zero: CheckDivInt, CheckDivLint, CheckDivReal, and CheckDivLReal

To prevent division by zero, you can use the functions `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`. If you include these functions in the application, then they are called before each division operation in the code.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

The default implementation of CheckDivReal:

Declaration section:

```
// This is automatically generated code: DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
    divisor:REAL;
END_VAR
```

Implementation section:

```
// This automatically generated code is a suggested implementation.
IF divisor = 0 THEN
    CheckDivReal:=1;
ELSE
    CheckDivReal:=divisor;
END_IF;
```

The `DIV` operator uses the output of the `CheckDivReal` function as a divisor. In the sample program below, `CheckDivReal` prevents division by `0` by changing the implicit value of the divisor `d` from "0" to `1` before the division operation is executed. Therefore, the division result is `799`.

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL:=0;
END_VAR
erg:= v1 / d;
```

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*

POU 'CheckDivLInt'

Functions for preventing division by zero: CheckDivInt, CheckDivLint, CheckDivReal, and CheckDivLReal

To prevent division by zero, you can use the functions `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`. If you include these functions in the application, then they are called before each division operation in the code.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

The default implementation of CheckDivReal:

Declaration section:

```
// This is automatically generated code: DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
    divisor:REAL;
END_VAR
```

Implementation section:

```
// This automatically generated code is a suggested implementation.
IF divisor = 0 THEN
    CheckDivReal:=1;
ELSE
    CheckDivReal:=divisor;
END_IF;
```

The `DIV` operator uses the output of the `CheckDivReal` function as a divisor. In the sample program below, `CheckDivReal` prevents division by `0` by changing the implicit value of the divisor `d` from "0" to `1` before the division operation is executed. Therefore, the division result is `799`.

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL:=0;
END_VAR
erg:= v1 / d;
```

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*

POU 'CheckDivReal'

Functions for preventing division by zero: `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`

To prevent division by zero, you can use the functions `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`. If you include these functions in the application, then they are called before each division operation in the code.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

The default implementation of CheckDiv-Real:

Declaration section:

```
// This is automatically generated code: DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
    divisor:REAL;
END_VAR
```

Implementation section:

```
// This automatically generated code is a suggested implementation.
IF divisor = 0 THEN
    CheckDivReal:=1;
ELSE
    CheckDivReal:=divisor;
END_IF;
```

The `DIV` operator uses the output of the `CheckDivReal` function as a divisor. In the sample program below, `CheckDivReal` prevents division by `0` by changing the implicit value of the divisor `d` from "0" to `1` before the division operation is executed. Therefore, the division result is `799`.

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL:=0;
END_VAR
erg:= v1 / d;
```

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*

POU 'CheckDivLReal'

Functions for preventing division by zero: `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`

To prevent division by zero, you can use the functions `CheckDivInt`, `CheckDivLint`, `CheckDivReal`, and `CheckDivLReal`. If you include these functions in the application, then they are called before each division operation in the code.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

The default implementation of CheckDiv-Real:

Declaration section:

```
// This is automatically generated code: DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
    divisor:REAL;
END_VAR
```

Implementation section:

```
// This automatically generated code is a suggested implementation.
IF divisor = 0 THEN
    CheckDivReal:=1;
ELSE
    CheckDivReal:=divisor;
END_IF;
```

The `DIV` operator uses the output of the `CheckDivReal` function as a divisor. In the sample program below, `CheckDivReal` prevents division by `0` by changing the implicit value of the divisor `d` from "0" to `1` before the division operation is executed. Therefore, the division result is `799`.

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL:=0;
END_VAR
erg:= v1 / d;
```

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*

POU 'CheckRangeSigned'

Function for monitoring the range limits of a subrange type of type DINT.

This monitoring function is responsible for the appropriate handling violations to range limits. Examples of reactions to violations include setting error flags and changing values. The functions are called implicitly when a value is assigned to a subrange type variable.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

When the function is called, it receives the following input parameters:

- `value`: Value that should be assigned to the subrange type variables
- `lower`: Lower range limit
- `upper`: Upper range limit

The return value is the assignment value as long as it is within the valid range. If not, then either the upper or lower limit is returned, depending on which threshold was violated.

For example, the assignment `i := 10*y` is replaced implicitly by `i := CheckRangeSigned(10*y, -4095, 4095);`

If `y` is "1000", then "10*1000=10000" is not assigned to `i` like in the original code. Instead, the upper range limit of "4095" is assigned.

The same is true for `CheckRangeUnsigned` function.

NOTICE!
If functions are not available, then the subrange is not checked for the respective variables at runtime. In this case, you can assign any value between -2147483648 and +2147483648 (or between 0 and 4294967295) to a variable of subrange type DINT/UDINT. You can assign any value between -9223372036854775808 and +9223372036854775807 (or between 0 and 18446744073709551615) to a variable of a subrange type LINT/ULINT.

CAUTION!
Linking area monitoring functions can lead to endless loops. For example, an endless loop can occur if the counter variable of a FOR loop is a subrange type and the counting range for the loop exits the defined subrange.

Example of an endless loop:

```
VAR
    ui : UINT (0..10000);
 ...
END_VAR

FOR ui:=0 TO 10000 DO
    ...
END_FOR
```

The program never exits the FOR loop because the `CheckRangeSigned` monitoring function prevents `ui` from being set to a value greater than 10000.

Example for `CheckRangeSigned`

The assignment of a value to a DINT variable of a signed subrange type is a condition for automatically calling the `CheckRangeSigned`. This function restricts the assignment value to the subrange as defined in the variables declaration. The default implementation of the function in ST is as follows:

Declaration section:

```
// This is automatically generated code: DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
    value, lower, upper: DINT;
END_VAR
```

Implementation:

```
// This automatically generated code is a suggested implementation.
IF (value < lower) THEN
    CheckRangeSigned := lower;
    ELSEIF(value > upper) THEN
    CheckRangeSigned := upper;
ELSE
    CheckRangeSigned := value;
END_VAR
```

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*

POU 'CheckLRangeSigned'

Function for monitoring the range limits of a subrange type of type LINT.

For an implementation example of range monitoring, refer to the help page for the `CheckRangeSigned` function.

This monitoring function is responsible for the appropriate handling violations to range limits. Examples of reactions to violations include setting error flags and changing values. The functions are called implicitly when a value is assigned to a subrange type variable.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

When the function is called, it receives the following input parameters:

- `value`: Value that should be assigned to the subrange type variables
- `lower`: Lower range limit
- `upper`: Upper range limit

The return value is the assignment value as long as it is within the valid range. If not, then either the upper or lower limit is returned, depending on which threshold was violated.

For example, the assignment `i := 10*y` is replaced implicitly by `i := CheckRangeSigned(10*y, -4095, 4095);`

If `y` is "1000", then "10*1000=10000" is not assigned to `i` like in the original code. Instead, the upper range limit of "4095" is assigned.

The same is true for `CheckRangeUnsigned` function.

NOTICE!
If functions are not available, then the subrange is not checked for the respective variables at runtime. In this case, you can assign any value between -2147483648 and +2147483648 (or between 0 and 4294967295) to a variable of subrange type DINT/UDINT. You can assign any value between -9223372036854775808 and +9223372036854775807 (or between 0 and 18446744073709551615) to a variable of a subrange type LINT/ULINT.

CAUTION!
Linking area monitoring functions can lead to endless loops. For example, an endless loop can occur if the counter variable of a FOR loop is a subrange type and the counting range for the loop exits the defined subrange.

Example of an endless loop:

```
VAR
    ui : UINT (0..10000);
 ...
END_VAR

FOR ui:=0 TO 10000 DO
    ...
END_FOR
```

The program never exits the FOR loop because the `CheckRangeSigned` monitoring function prevents `ui` from being set to a value greater than 10000.

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*
- *Chapter 1.4.1.20.2.19.6 “POU 'CheckRangeSigned'” on page 912*

POU 'CheckRangeUnsigned'

Function for monitoring the range limits of a subrange type of type UDINT.

For an implementation example of range monitoring, refer to the help page for the `CheckRangeSigned` function.

This monitoring function is responsible for the appropriate handling violations to range limits. Examples of reactions to violations include setting error flags and changing values. The functions are called implicitly when a value is assigned to a subrange type variable.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

When the function is called, it receives the following input parameters:

- `value`: Value that should be assigned to the subrange type variables
- `lower`: Lower range limit
- `upper`: Upper range limit

The return value is the assignment value as long as it is within the valid range. If not, then either the upper or lower limit is returned, depending on which threshold was violated.

For example, the assignment `i := 10*y` is replaced implicitly by `i := CheckRangeSigned(10*y, -4095, 4095);`

If `y` is "1000", then "10*1000=10000" is not assigned to `i` like in the original code. Instead, the upper range limit of "4095" is assigned.

The same is true for `CheckRangeUnsigned` function.

NOTICE!
If functions are not available, then the subrange is not checked for the respective variables at runtime. In this case, you can assign any value between -2147483648 and +2147483648 (or between 0 and 4294967295) to a variable of subrange type DINT/UDINT. You can assign any value between -9223372036854775808 and +9223372036854775807 (or between 0 and 18446744073709551615) to a variable of a subrange type LINT/ULINT.

CAUTION!
Linking area monitoring functions can lead to endless loops. For example, an endless loop can occur if the counter variable of a FOR loop is a subrange type and the counting range for the loop exits the defined subrange.

Example of an endless loop:

```
VAR
    ui : UINT (0..10000);
 ...
END_VAR

FOR ui:=0 TO 10000 DO
    ...
END_FOR
```

The program never exits the FOR loop because the `CheckRangeSigned` monitoring function prevents `ui` from being set to a value greater than 10000.

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*
- *Chapter 1.4.1.20.2.19.6 “POU 'CheckRangeSigned'” on page 912*

POU 'CheckLRangeUnsigned'

Function for monitoring the range limits of a subrange type of type ULINT.

For an implementation example of range monitoring, refer to the help page for the `CheckRangeSigned` function.

This monitoring function is responsible for the appropriate handling violations to range limits. Examples of reactions to violations include setting error flags and changing values. The functions are called implicitly when a value is assigned to a subrange type variable.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

When the function is called, it receives the following input parameters:

- `value`: Value that should be assigned to the subrange type variables
- `lower`: Lower range limit
- `upper`: Upper range limit

The return value is the assignment value as long as it is within the valid range. If not, then either the upper or lower limit is returned, depending on which threshold was violated.

For example, the assignment `i := 10*y` is replaced implicitly by `i := CheckRangeSigned(10*y, -4095, 4095);`

If `y` is "1000", then "10*1000=10000" is not assigned to `i` like in the original code. Instead, the upper range limit of "4095" is assigned.

The same is true for `CheckRangeUnsigned` function.

NOTICE!
If functions are not available, then the subrange is not checked for the respective variables at runtime. In this case, you can assign any value between -2147483648 and +2147483648 (or between 0 and 4294967295) to a variable of subrange type DINT/UDINT. You can assign any value between -9223372036854775808 and +9223372036854775807 (or between 0 and 18446744073709551615) to a variable of a subrange type LINT/ULINT.

CAUTION!
Linking area monitoring functions can lead to endless loops. For example, an endless loop can occur if the counter variable of a FOR loop is a subrange type and the counting range for the loop exits the defined subrange.

Example of an endless loop:

```
VAR
    ui : UINT (0..10000);
 ...
END_VAR

FOR ui:=0 TO 10000 DO
    ...
END_FOR
```

The program never exits the FOR loop because the `CheckRangeSigned` monitoring function prevents `ui` from being set to a value greater than 10000.

See also

- *Chapter 1.4.1.8.21 “Using POUs for implicit checks” on page 309*
- *Chapter 1.4.1.20.2.19.6 “POU 'CheckRangeSigned'” on page 912*

POU 'CheckPointer'

Monitoring function for pointers (Checkpointer)

Use this function to monitor the memory access of pointers in runtime mode. As opposed to other monitoring functions, a standard suggestion does not exist for the implementation of `CheckPointer`. You must define an implementation according to your own requirements.

The `CheckPointer` function should check whether the returned pointer references a valid memory address; monitors whether the orientation of the referenced memory range matches the variable type that the pointer refers to. If both conditions are fulfilled, then the pointer is returned. If not, then the function should complete an appropriate error handling.

CAUTION!
To obtain the feature for monitoring functions, do not edit the declaration section. However, you are permitted to add local variables.

NOTICE!
An implicit monitoring function call does not occur for THIS pointer and SUPER pointer.

NOTICE!
For complier version 3.5.7.40 and later, the implicit check function *"Checkpointer"* also acts on `REFERENCE` variables in the same way as on pointer variables.

Model

Declaration:

```
// Automatically generated code: DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
    ptToTest : POINTER TO BYTE;
    iSize : DINT;
    iGran : DINT;
    bWrite: BOOL;
END_VAR
```

Implementation: (incomplete)

```
// Not a standard implementation. Please add your own code here.
CheckPointer := ptToTest;
```

When the function is called, CODESYS provides the following input parameters:

- `ptToTest`: Target address of the pointer
- `iSize`: Size of the referenced variable; the data type of `iSize` must be compatible with `INT` and cover the dimensional scope of the variables.
- `iGran`: Granularity of the referenced size; this is the largest non-structured data type contained in the referenced variables; the data type of `iGran` must be compatible with `INT`
- `bWrite`: Access type (`TRUE` = write access, `FALSE` = read access); the data type of `bWrite` must be `BOOL`

When the result of the check is positive, the unchanged pointer is returned (`ptToTest`).

See also

- *Chapter 1.4.1.8.21 "Using POUs for implicit checks" on page 309*

Object 'Project Settings'

Symbol: 

Function: This object contains the configuration of the project.

Call:

- *"Project ➔ Project Settings"*
- Double-click on the object in the device tree

CODESYS saves the project settings directly in the project. If, for example, you transfer a project to another system, the *"Project Settings"* object is also transferred with it without a project archive being required.

The project settings are valid project-wide and offer setting possibilities for various categories such as *"AS"* or *"Users and Groups"*. The available categories vary, depending on which software packages you have installed via the package manager.

See also

- *Chapter 1.4.1.2.3.2 "Making project settings" on page 193*
- *Chapter 1.4.1.20.3.8.4 "Command 'Package Manager'" on page 1059*
- *Chapter 1.4.1.20.4.11.1 "Dialog 'Project Settings' - 'SFC'" on page 1171*
- *Chapter 1.4.1.20.4.11.2 "Dialog 'Project Settings' - 'Users and Groups'" on page 1172*
- *Chapter 1.4.1.20.4.11.3 "Dialog Box 'Project Settings' - 'Compileoptions'" on page 1173*

- *Chapter 1.4.1.20.4.11.4 “Dialog Box 'Project Settings' - 'Compiler Warnings'” on page 1173*
- *Chapter 1.4.1.20.4.11.5 “Dialog 'Project Settings' – 'Source Download'” on page 1174*
- *Chapter 1.4.1.20.4.11.6 “Dialog 'Project Settings' - 'Page Setup'” on page 1175*
- *Chapter 1.4.1.20.4.11.7 “Dialog 'Project Settings' - 'Security'” on page 1176*
- *Chapter 1.4.1.20.4.11.8 “Dialog 'Project Settings' - 'Static Analysis Light'” on page 1177*
- *Chapter 1.4.1.20.4.11.9 “Dialog 'Project Settings' - 'Visualization'” on page 1180*
- *Chapter 1.4.1.20.4.11.10 “Dialog 'Project Settings' - 'Visualization Profile'” on page 1181*

Object 'Project Information'

Symbol: 

Function: The object contains the properties, meta-information, and project information. With this, you can check the authorship and integrity of the project.

Call

- Double-click the object in the device tree
- Menu bar: *“Project ➔ Project Information”*

Requirement: CODESYS creates the object when you click *“Project ➔ Project Information”*, and the dialog opens.

CODESYS saves the project information directly in the project. For example, if you transfer a project to another system, then the *“Project Information”* object is also transferred. You do not need a project archive.

Tab 'File'

The tab displays the properties of the project file and their attributes. You cannot edit these attributes. They correspond to the file properties of Windows Explorer.

Tab 'Summary'

The tab contains general information and meta-information of the project file. CODESYS uses this information to create keys on the *“Properties”* tab. For example, if the name `Company_A` is specified in *“Company”*, then the `Company` key with the value `Company_A` is provided on the *“Properties”* tab.

NOTICE!
If you save your project as a library project, then you should pay attention to the guidelines for library developers (Library Development Summary).

For a library project, a *“Company”*, a *“Title”*, and a *“Version”* must be specified to install the library.	
“Company”	Name of the company (example: `Company_A`).
“Title”	Title of the project (example `Automation_A`).
“Version”	Version of the project (example: `0.0.0.1`).
“Released”	 : Activates protection from modification. Result: If you edit the project now, then a dialog prompt opens to confirm whether you really want to change the project. If you reply to this prompt one time by clicking *“Yes”*, then no additional prompts appear for more editing actions.

"Categories"	Categories of the library project, according to which you can sort in the *"Library Repository"* dialog. If no category is specified, then the category *"Other"* is assigned to the library. The categories originate from one or more external description files in XML format. However, they can also originate from a library project that has already been created. Requirement: The project is a library project. ...: The *"Library Categories"* dialog opens where you can add library categories.
"Default namespace"	If you do not define a standard namespace here, then the name of the library file is applied automatically as the namespace.
"Author"	Author of the project (example: `Arnold Best`).
"Description"	Example: `For internal use only`

See also

- *Chapter 1.4.1.20.3.1.7 "Command 'Save Project as Compiled Library'" on page 960*
- *Chapter 1.4.1.16.1 "Information for Library Developers" on page 449*

Table 84: Dialog "Library Categories"

List of categories	List of the categories that are assigned to the library project. They can originate from several sources. After you specify all desired categories, click *"OK"* to confirm.
Button *"Add"*	The *"From Description File"* and *"From Other Library"* commands appear.
Button *"Remove"*	CODESYS removes the selected category.
Command *"From Description File"*	The *"Select Description File"* dialog opens for you select a description file (`*.libcat.xml`). The file contains command categories. When you click *"Open"*, CODESYS accepts the categories.
Command *"From Other Library"*	The *"Select Library"* dialog opens, where you select a library with command categories to be accepted. When you click *"Open"*, CODESYS accepts the categories.
Button *"OK"*	CODESYS provides the categories as project information and displays it in the *"Library Categories"* field.

See also

- *Chapter 1.4.4.1 "Guidelines for creating libraries" on page 1249*
- *Chapter 1.4.1.20.3.8.5 "Command 'Library Repository'" on page 1061*

Tab 'Properties' On this tab, you can define keys that you can control externally from user-specific programs.

NOTICE!
If you have opened a library project, then note the description of the relevant keys in the guidelines for library developers (Library Development Summary).

If you have opened a symbol library as a project, then the key `VisuSymbolLibrary = TRUE` must be defined. It identifies the library as a symbol library.

"Key"	Name of the key. Specify any string of text for the new key, or select an existing key from the *"Properties"* table.
"Type"	Data type of the key. Possible types: *"Text"*, *"Date"*, *"Number"*, *"Boolean"*, *"Version"*.

"Value"	Value of the key in permitted format ● *"Text"*: Any character string ● *"Date"*: Example: `Friday, January 1, 2016 00:00:00`. Minimum entry for the date: `1.1` ● *"Number"*: Integer in Integer32 format with or without a sign (example: `-32500`). ● *"Boolean"*: `True` or `False`, capitalization irrelevant. ● *"Version"*: Examples: `1.1`, `1.0.1.0`, maximum four figures.
"Add"	Adds the new defined key to the *"Properties"* table.
"Modify"	Saves the change made for the key selected in the *"Properties"* table.
"Remove"	Removes the key selected in the *"Properties"* table.

"Properties"	List of the properties that are defined as keys. CODESYS creates keys automatically for the information in the *"Summary"* tab. Click a key to edit it in the input fields above the list.

See also

- *Chapter 1.4.4.1 "Guidelines for creating libraries" on page 1249*
- *Using the Symbol Library in the Visualization*
- *Chapter 1.4.1.20.4.10.17 "Dialog 'Properties' - 'Image Pool'" on page 1168*

Tab 'Statistics' The dialog provides statistical information about the number of objects of the individual type or use in the project.

Tab 'Licensing' The dialog is for the license protection of libraries.

CAUTION!
You can protect only compiled libraries in this way.

Table 85: "Variables"

"Activate dongle licensing"	☑: The library requires a dongle with a license to use it.
"Firm code"	License information that must be supplied from the dongle for using the library later.
"Product code"	
"Activation URL"	
"Activation mail"	

See also

- *Chapter 1.4.4.1 "Guidelines for creating libraries" on page 1249*

Tab 'Signing' This tab is displayed only for existing libraries whose signing has been created with this tab. This tab is no longer visible for newly generated libraries.

When a certificate-signed library is created (possible as of CODESYS V3 SP15) and library compatibility with CODESYS < V3 SP15 is not set, the settings on this tab are disabled. In this case, the signing is done by means of a certificate that has to be assigned to the user profile in the security screen.

One method, which is not recommended but may be necessary in some case for compatibility with versions < V3 SP15, is the less secure signing of a library by means of a vendor-specific, one-time key in this dialog. Requirement: This key is available as a private key file (`*.libpk`) with an associated token. The user of the library also has to obtain this key in order to be able to check whether the last signing was actually performed by the library vendor.

"Activate signing"	☑: CODESYS signs the library project with a single-use, manufacturer-specific key.
"Private key file"	Location of the private key file `*.libpk` (example: `D:\for lib developers only\mycomp_libkey.libpk`).
"Public key token"	Example: `427A5701DA3CF3CF` Requirement: A private key file is specified, and CODESYS has read and entered the token.
"Create Private Key File"	CODESYS creates a new private key file.

See also

- *Chapter 1.4.1.20.3.3.18 "Command 'Security Screen'" on page 995*

Options for creating blocks for accessing project information

"Automatically generate 'Project Information' POUs"	Note: The functions that are created with this option can be used only if the runtime supports the `WSTRING` data type. If this is not the case, then you can use the functions that were created automatically for the with the individual items of the project information, at least in the application for accessing properties. These functions are not registered in the runtime. ☑: CODESYS creates POUs of the `FUNCTION` type in the *"POUs"* view, allowing programmatic access to the project properties in the application. The function blocks `GetCompany`, `GetTitle` and `GetVersion` are created for the properties *"Company"*, *"Title"* and *"Version"*. The following function blocks are available for user-defined properties: • `GetBooleanProperty`: BOOL (TRUE/FALSE) • `GetNumberProperty`: DINT (numeric value) • `GetTextProperty`: WSTRING (character string) • `GetTextProperty2`: POINTER TO WSTRING (unlimited length) • `GetVersionProperty`: VERSION (version number as character string) Note: Do not activate this option for standard libraries, because this can cause problems on smaller systems due to the additional memory requirements. Note: If a library also contains this project information POU, then you should use the operator `__POOL` to make sure that this POU is accessed.
"Automatically generate 'Library Information' POUs"	☑: CODESYS creates POUs of the `FUNCTION` type in the *"POUs"* view, allowing programmatic access to the project properties in the application. For the *"Version"* and *"Released"* properties, the following functions are created: `GetLibVersion` (version number as character string), `GetLibVersionNumber` (version number as numeric value), and `IsLibReleased` (TRUE/FALSE). Note: These functions are not registered in the runtime. The option is available as an alternative solution is the runtime does not support the `WSTRING` data type, therefore not permitting you to use the functions created with the *"Automatically generate 'Project Information' POUs"* option.

See also

- *Chapter 1.4.1.2.3.1 “Retrieving and Editing Project Information” on page 191*
- *Chapter 1.4.1.2.3.2 “Making project settings” on page 193*
- *Chapter 1.4.1.19.3.73 “Operator '__POOL'” on page 630*

Object 'Recipe Manager'

Tab 'Storage' The recipe manager provides functions for maintaining user-defined variable lists, known as recipe definitions. The recipe definitions can be stored in recipe files on the PLC.

“Storage Type”	*“Textual”*: CODESYS saves the recipe in a readable Format with the configured columns and delimiters. *“Binary”*: CODESYS saves the recipe in a non-readable binary format. This format requires less storage space. Note: You can read binary recipes again only if you have not changed the variable lists.
“File path”	`<directory name>\` Example: `AllRecipes\` Path on the runtime system Notes: ● The path has to end with a backslash ("\"). ● The path is usually a relative path on the target system in the directory of the runtime files (`PlcLogic`). ● Access to paths outside of the directory `PlcLogic` is not permitted on every controller. An absolute path for Windows systems can be selected by pressing the ... button. Example of the file path in the runtime system: `PlcLogic/AllRecipes` CODESYS saves a file in this directory for each recipe when downloading to the PLC. The requirement is that you select the *“Recipe management in the PLC”* option. The files are loaded to the recipe manager each time the application is restarted.
“File extension”	File extension for the recipe file in the format `.<file extension>` The resulting default name for recipe files is in the form `<recipe>.<recipe definition>.<file extension>`.
“Separator”	Delimiters between the individual values in the saved file
“Available Columns” *“ Selected Columns”*	Defines the information that is saved and in which order in the recipe file
“Save as Default”	CODESYS uses the settings on the tab throughout the entire project for all other recipe managers.

Tab 'General'

“Recipe management in the PLC”	☑: Has to be selected for the user program or visualization elements to load recipes at runtime. If you transfer recipes to the PLC exclusively via the CODESYS program interface, then you can clear this option.

Table 86: “Save Recipe”

“Save recipe changes to recipe files automatically”	When *“Recipe management in the PLC”* is selected, there is the following option for saving the recipe: ☑: We recommend this option because it helps the recipe manager operate "normally". The recipe files on the PLC are updated automatically at runtime whenever a recipe is changed.

Table 87: “Load Recipe”

When *“Recipe management in the PLC”* is selected, there are the following two options for downloading from the PLC:	
“Download only for exact match of the variable list”	◉: The recipe is only downloaded if the file on the PLC contains all variables from the variable list of the recipe definition of the application and these are sorted in the same order. Additional entries at the end are ignored. If the required match does not exist, then the error status `ERR_RECIPE_MISMATCH` is set (`RecipeManCommands.GetLastError`).
“Download variables with matching names”	◉: The recipe values are downloaded only for those variables that have the same name in the recipe definition of the application as in the recipe file on the PLC. If the variable lists differ in composition and sorting, then no error status is set. In this way, recipe files can also be downloaded if variables in the file or in the recipe definition have been deleted.

“Overwrite existing recipes on download”	☑: If recipe files with the same name exist on the controller, then they are overwritten with the configured values from the project when the application is started. If the values from the existing recipe files should be loaded instead, then this option has to be disabled. Requirement: The *“Storage Type”* is *“Textual”* and the *“Save recipe changes to recipe files automatically”* option is selected.

Table 88: “Write Recipe”

The following options are available for writing recipe values to the variables on the PLC:	
“Limit the variable to min/max when recipe value is out of the range”	◉: If the recipe contains a value that is outside of the range of values specified in the definition, then the defined minimum or maximum value is written to the PLC variable instead of this value.
“Do not write to a variable when the recipe value is out of the min/max range”	◉: If the recipe contains a value that is outside of the range of values specified in the definition, then no value is written to the PLC variable. It retains its current value.

Table 89: “Read Recipe”

The following option is available for reading recipe values from the PLC into the recipe in the project:	
“Check recipe for changes”	Always use the function block `RecipeManCommands` from `RecipeManagement.library` to read recipes. Never call the method cyclically. This is because each call can be written to the file system, which is time-intensive and burdens the controller. For example, a Raspberry protocol interpreter has a limited number of write cycles. ☑: With each method call, the current PLC variable values are first read into the recipe. Then the system checks whether the values have changed. Only if the values have changed is the recipe saved. This means that the recipe file is overwritten with the current recipes. The option can be used in order to update the recipe file in the local file system only if recipe values have changed on the PLC. However, it affects performance because it generates additional code for checking. ☐: With each method call, the current PLC variable values are first read into the recipe. Then the recipe is written to the recipe file in the local file system. Note: As the file system is written to each call, the controller can be very burdened.

Recipes during online mode

Table 90: Option “Save recipe changes to recipe files automatically” is selected.

Menu commands	**Behavior of the recipes defined in the project**	**Behavior of the defined recipes at runtime**
“Online ➔ Reset Warm” *“Online ➔ Reset Cold”* *“Online ➔ Download”*	The recipes of all recipe definitions are downloaded with the values from the current projects.	Dynamically generated recipes remain unchanged.
“Online ➔ Reset Origin”	The application is removed from the PLC. If a download is done again afterwards, then the recipes are restored as for an online reset warm.	
Shutdown and restart of the PLC	After a restart, the recipes are downloaded again from the automatically created files. This will restore the same state as before shutdown.	
“Online ➔ Online Change”	The recipe values remain unchanged. In runtime mode, a recipe can be changed only via the function block command `RecipeManCommands`.	
“Debug ➔ Stop” *“Debug ➔ Start”*	The recipes remain unchanged when the PLC is stopped or started.	

Table 91: Option “Save recipe changes to recipe files automatically” is not selected.

Actions	**Recipes defined in the project**	**Recipes defined at runtime**
“Online ➔ Reset Warm” *“Online ➔ Reset Cold”* *“Online ➔ Download”*	The recipes of all recipe definitions are downloaded with the values from the current projects. However, these are set in the memory only. To save recipes to a file, you have to run the *“Save Recipe”* command explicitly.	Dynamically generated recipes are lost.
“Online ➔ Reset Origin”	The application is removed from the PLC. When a download is performed afterwards, the recipes are restored.	Dynamically generated recipes are lost.

Actions	Recipes defined in the project	Recipes defined at runtime
Shutdown and restart of the PLC	After the restart, the recipes are downloaded again from the automatically created files. This will restore the same state as before shutdown.	
“Online ➔ Online Change”	The recipe values remain unchanged. In runtime mode, a recipe can be changed only via the function block command `RecipeManCommands`.	
“Debug ➔ Stop” *“Debug ➔ Start”*	The recipes remain unchanged when the PLC is stopped or started.	

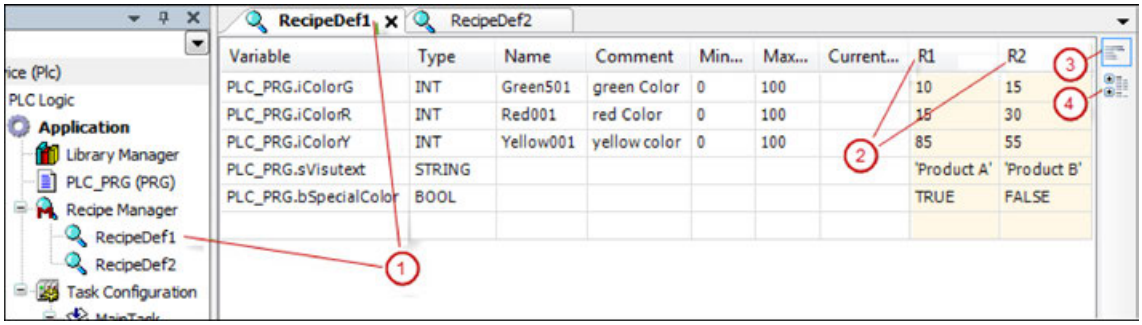
See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19.9 “Command 'Read and Save Recipe'” on page 1130*
- *Chapter 1.4.1.20.2.23 “Object 'Recipe Definition'” on page 926*
- *Method Calls of the 'Recipe Management' Library*

Object 'Recipe Definition'

In the recipe definition (1), you define different data sets for the variables, which are termed recipes (2).

You can toggle the display of the recipe definition between the flat list view (3) and the structured view (4). In the structured view, CODESYS groups variables according to structure.



“Type”	Entered automatically
“Name”	Optional
“Minimal Value” *“Maximal Value”*	If the variable value is less than the *“Minimal Value”* or greater than the *“Maximal Value”*, then CODESYS sets the value to the *“Minimal Value”* or *“Maximal Value”*.
“Comment”	Additional information, for example the unit of the value.
“Current Value”	Current variable value; not shown in online mode

Table 92: “Additional commands in the context menu in the structured view”

“Add Sibling”	Adds a sibling variable to the recipe definition.
“Add Child”	Adds a child variable to the recipe definition.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*

Object 'Text List'

Symbol: 

This object is used to create, manage, and translate texts. It contains a table with texts where you can add new texts. You can select a text which you have composed here can be selected in a visualization in the *"Dynamic texts"* property of an element. In runtime mode, the visualization displays this text dynamically in the selected language.

When the object is assigned to an alarm group and is located below the *"Alarm Configuration"* object, CODESYS adds the texts of the alarm group to the table. You can also add texts.

"ID"	Unique identifier of the text
"Standard":	Source text as character string (example: `Information A: %i options`). Use the keyboard shortcut *[Ctrl]+[Enter]* to add a line break. Double-click in the field to edit the text.
The table contains as many language columns as you want to add. A language column is named with a language code that you specified when you created the column by means of the *"Insert Language"* command.	
"<language code>"	Name of the language as a language code. Example: `en-US`. This column contains the translation of the text which is composed under *"Standard"*. Under the condition that the language code is selected as the language in the visualization manager, a visualization displays the translated text in runtime mode. If a translation has not been composed, then CODESYS uses the text under *"Standard"*. A visualization in runtime mode can also change the language if requested by a user.
Blank line	Edit the line to add your own text.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*
- *Chapter 1.4.1.20.2.9 "Object 'GlobalTextList'" on page 871*
- For descriptions about alarm management and alarm visualization, see the help for CODESYS Visualization.

Object 'Symbol Configuration'

You can use the symbol configuration for creating symbol descriptions for project variables. Click *"Project ➔ Add Object"* to add a symbol configuration object to the device tree. Then define specific presets. See dialog below: *"Add Symbol Configuration"*.

Double-click the *"Symbol Configuration"* object to open the symbol configuration editor.

Dialog 'Add Symbol Configuration'

Function: This dialog is used to define the defaults for a *"Symbol Configuration"* object.

Call: *"Project ➔ Add Object ➔ Symbol Configuration"* menu; context menu of the application object.

"Include comments in XML"	Exports the symbol file with the comments assigned to the variables.
"Support OPC UA features"	Note: Availability and editability of this option depend on the device. ☑: When downloading the symbol configuration, additional information is also downloaded to the controller. The information below is necessary for operating the OPC UA server. ● Base types of inherited function blocks ● Contents of attributes that were assigned via compiler pragmas ● Scopes (example: `VAR_INPUT`, `VAR_OUTPUT`, `VAR_IN_OUT`)

Table 93: “Client-Side Data Layout”

For detailed information and examples of layout options, see the next section "Symbol Configuration Editor".	
“Compatibility layout”	This setting is used for the compatibility of old projects. The data layout created for the client is matched as much as possible to the layout created internally by the compiler.
“Optimized layout”	Recommended for new projects. Calculates the output layout in optimized form detached from the internal compiler layout. Does not generate any gaps for unpublished elements and strictly fulfills the requirements for memory alignment of the data types. Requires compiler version 3.5.7.0 or higher.

Symbol configuration editor The editor includes a table with selected variables and a menu bar for editing.

Table 94: Menu bar

“View”	You can use this button for activating and deactivating the following categories of variables used in the configuration editor: ● *“Unconfigured from Project”*: Variables that have not been added to the symbol configuration, but are provided in the project. ● *“Unconfigured from Libraries”*: Variables that have not been added to the symbol configuration, but are provided in the project. ● *“Symbols Exported via Attribute”*: This filter also lists the variables that have already been marked for export in the symbol file by means of the `{attribute 'symbol' := 'read'}` pragma. These symbols are displayed in gray. The *“Attribute”* column shows which access rights are set by the pragma.
“Build”	Compiles the project. Requirement for current preparation of variables in the configuration editor.

"Settings"	• *"Support OPC UA features"*: Note: Availability and editability of this option depend on the device. ☑: When downloading the symbol configuration, additional information is also downloaded to the controller. The information below is necessary for operating the OPC UA server. This currently includes the following information: – Base types of inherited function blocks – Contents of attributes that were assigned via compiler pragmas – Scopes (example: `VAR_INPUT`, `VAR_OUTPUT`, `VAR_IN_OUT`) • <!> *"Include comments in XML"* ☑: Exports the symbol file with the comments assigned to the variables. • *"Include Node Flags in XML"* ☑: The namespace node flags provide additional information about the origin of a node in the namespace. The node flags always in the symbol table when OPC UA is activated. However, its inclusion in the XML file can be deactivated as some defective parsers have problems with it. • *"Configure Comments and Attributes"* Opens the *"Comments and Attributes"* dialog. Here you configure the details of what should be included in the symbol configuration and XML file with respect to comments and attributes. • *"Configure synchronization with IEC tasks"*: Opens the *"Properties - <device name>"* dialog, *"Options"* tab. This setting allows for the symbolic clients (e.g. visualizations or database links based on the PLCHandler) to have consistent read/write access synchronized with the IEC tasks. For a detailed description of this setting, see the "Setting: Configure synchronization with IEC tasks" section below. Note: Variable access which is synchronous with the IEC tasks can increase the jitter for all IEC applications on this device. Synchronized consistent access can interrupt the real-time capability. • List box for defining the data layout type for the client of the symbol configuration: Note: See the "Example of data layout types" section at the end of this help page. – *"Optimized layout"*: Recommend for new projects. Calculates the output layout in optimized form detached from the internal compiler layout. Does not generate any gaps for unpublished elements and strictly fulfills the requirements for memory alignment of the data types. Requires compiler version 3.5.7.0 or higher. – *"Compatibility layout"*: This setting is used for the compatibility of old projects. The data layout created for the client is matched as much as possible to the layout created internally by the compiler. Due to the configuration possibilities of the symbol configuration which have grown over time, problematic offsets can still result. Causes of offsets Memory gaps due to internal pointers or references in function blocks and structure components that are not released for symbol configuration. Memory gaps that occur differently in 32-bit and 64-bit systems depending on the data type, such as `__XINT` / `__XWORD` . Fields that are at uneven addresses. Some clients are not set up for this. Unintentional memory misalignment, which occurs when using the attributes `'pack_mode'` or `'relative_offset'`. • *"Use Empty Namespaces by Default (V2 Compatibility)"*: Required when using a CODESYS V2-compatible OPC server configuration. ☑: Behavior same as in CODESYS V2.3. – Program variables are exported without an application name (`Application.PLC_PRG.MyVar` --> `PLC_PRG.MyVar` – Global variables are exported additionally without the GVL name (`Application.GVL.MyGlobVar` --> `.MyGlobVar`

	• *“Enable Direct I/O Access”*: This feature is potentially dangerous and **not intended for operation in production**. Activate only for error checking and tests, or when commissioning the machinery (for example, for checking cables connections). ☑: In the symbol configuration, you can also use access to direct I/O addresses that correspond to IEC syntax (for example, "%IX0.0"). Access to input addresses (I) is read-only*. Access to output addresses (Q) and memory addresses (M) can be read-write. *Information: In simulation mode, write access to symbols is also possible for input addresses. Because external clients for protocols such as OPC or OPC UA do not always support IEC syntax for direct addresses, access is also provided using an array syntax in the namespace __MIO of the implicit code. For example, you can also access __MIO.MIO_IX[2].x3 instead of %IX2.3. However, the symbols for array access are hidden in browsers because some clients cannot handle the large number of nodes (several thousand depending on the size of the I/O ranges). • *“Support Calls of Functions, FBs, Methods, and Programs”*: Note: Availability and editability of this option depend on the device. ☑: The access rights *“execute”* can be set in the symbol table for symbols of POUs of type function, function block, method, or program. The *“Support OPC UA features”* option also has to be selected in the *“Settings”*. • *“Include Call information in XML”*: ☑: The information about called functions, function blocks, methods, or programs is also listed in the XML file of the symbol configuration. The option is enabled only if the *“Support calls of functions, FBs, methods, and programs”* option is supported by the device. • *“Enable Symbol Sets”*: ☑: A toolbar with buttons and a list box is displayed above the symbol table. You can use this to configure symbol sets for client-specific assignment of access rights to the controller. See "Toolbar for symbol set configuration" below.
“Download”	If you use a device that supports its own application file for the symbol configuration, then this button is also available in the toolbar. If you change the symbol configuration in online mode, then you can load the new `<application name>._symbols` file immediately to the PLC.
“Tools”	*“Save XSD Scheme File”*: This command opens the standard dialog for saving a file in the file system. With this command, you can prepare the XSD format of the symbol file, for example for use in external programs.

Table 95: Symbol table

“Access Rights”	You can change the access rights for a symbol by clicking the symbol in the *“Access Rights”* column. Icons for access rights (in ascending order) • : Read only • : Write only • : Read and write • : Execute This permission allow for execute access to functions, function blocks, methods, and programs. Requirements for the assignment: The device provides the *“Support calls of functions, FBs, methods, and programs”* and *“Support OPC UA features”* options. Both options are activated in the *“Settings”*. Note: In case the controller has a user management, you can use symbol sets to define client-specific access rights to the same symbols.
“Maximal”	Maximum access rights for this symbol

"Attribute"	If the access right was assigned by attribute, then a corresponding icon is displayed here.
"Type"	Alias data types are also displayed In CODESYS V3.5 SP6 and higher. Example: `MY_INT : INT` for a variable declared with the data type `MY_INT` (type `INT`).
"Members"	You can add variables of a structured data type also by selecting a check box for symbol configuration in the *"Symbols"* column. This causes CODESYS to export all member variable symbols. However, in the *"Members"* column, you can click the ellipsis button (...) to select only specific structural components. Note: This selection applies to all instances of this data type for which symbols are exported. If a member of a structured type cannot be selected, then an asterisk (*) is displayed in the check boxes of the members to indicate that all exportable members of that type are exported.

Table 96: Toolbar for symbol set configuration

"List box"	Already defined symbol sets
"Add New Symbol Set"	Opens the *"Add New Symbol Set"* dialog for specifying a name for this set
"Add Duplicate from Selected Symbol Set"	Opens the *"Add Duplicate from Selected Symbol Set"* dialog. A copy is created for the set selected in the list box. You can change the default name (`<group name>_duplicate`).
"Rename Selected Symbol Set"	Opens the *"Rename Selected Symbol Set"* dialog for specifying another name for the set selected in list box.
"Delete selected Symbol Set"	Opens a dialog prompting whether or not the symbol set selected in the list box should be deleted.
"Configure Symbol Rights"	Opens the *"Symbol Rights"* tab of the device editor. When logged in there, you can assign different access rights for each user group (client) to the symbol set selected in the list box.

See also

- *Chapter 1.4.1.20.2.8.15 "Tab 'Symbol Rights'" on page 868*

Dialog 'Comments and Attributes'

Table 97: "Symbol Table Contents"

"Enable extended OPC UA information"	Note: Availability and editability of this option depend on the device. ☑: Additional information that can be evaluated by OPC UA servers is included in the symbol table. This includes inheritance information of user-defined data types and the namespace node flags. Additional information, such as comments and attributes, can also be included if the OPC UA setting is active. When the OPC UA setting is enabled, attributes are included in the symbol table according to the following rule: ● In compiler versions V3.5.5.0 to V3.5.7.X, all attributes are included according to the *"Match simple identifiers"* setting. ● In compiler version V3.5.8.X, all attributes are included according to the setting *"Include all attributes"*. ● In compiler version V3.5.9.0 and higher, you can customize the attributes that are included.
"Include comments"	Requirement: *"Enable extended OPC UA information"* is activated. ☑: Comments and attributes are also saved in the symbol table.

“Include attributes”	
“Also include comments and attributes for type nodes”	Requirement: *“Include comments”* is activated. ☑: The information for type nodes is also included (user-defined types, such as `STRUCT` and `ENUM` elements). ☐: Only directly exported variables have comments and attributes.

Table 98: “XML symbol file contents”

“Include namespace node flags”	☑: The namespace node flags provide additional information about the origin of a node in the namespace. The node flags always in the symbol table when OPC UA is activated. However, its inclusion in the XML file can be deactivated as some defective parsers have problems with it.
“Include comments”	☑: Comments can also be saved in the XML file. In compiler versions V3.5.5.x to V3.5.8.0, this includes the setting *“Prefer docu-comments”*.
“Include attributes”	☑: Attributes can also be saved in the symbol file.
“Also include comments and attributes for type nodes”	Requirement: *“Include comments”* is activated. ☑: The information for type nodes is also included (user-defined types, such as `STRUCT` and `ENUM` elements). ☐: Only directly exported variables have comments and attributes.

Table 99: “Select Comments”

Requirement: *“Include comments”* is activated.	
“Include docu comments” *“Include normal comments ”* *“Always include both types of comments”* *“Prefer docu comments, fallback to normal ones”* *“Prefer normal comments, fallback to docu comments”*	The options determines the comments that are saved in the symbol configuration.

Table 100: “Filter Attributes (Case-Insensitive)”

Requirement: *“Include attributes”* is activated.	
“Include all attributes” *“Include attributes starting with”* *“Filter attributes with regular expression”*	Defines the attributes that are saved in the symbol configuration.
“Match simple identifiers”	Exists primarily due to the backward compatibility to older versions in order to emulate the old behavior.

Setting: Configure synchronization with IEC tasks

For synchronously consistent access, the symbolic client waits in the runtime when processing a read or write request until a time is found when no IEC task is executed. When this gap is detected, restarting the IEC tasks is prevented until all values of the variable list have been copied. Then the IEC tasks are planned again as usual. Synchronized access can cause a delayed starting of IEC tasks, which is shown as increased jitter. As all applications in the runtime are managed by a common scheduler, this potential impairment of the real-time behavior

affects all applications on the device. All applications of the device are affected, regardless of whether or not they include a symbol configuration or they have been downloaded to the controller from one or more CODESYS projects. Therefore, the runtime permits synchronized consist access only if this it allows all applications that are downloaded to the controller at the time of access.

The setting is located in the editor of the symbol configuration of the “Settings” menu. In addition, the setting is also located in the context menu of the controller when you click the “Properties” command and then select the “Options” tab in the opened dialog.

For applications without symbol configuration, the setting can only be found in the properties dialog.

NOTICE!

After changing the setting, all applications downloaded to the device by means of a download or online change have to be reloaded and all boot applications updated.

In which cases is synchronized consistent access necessary?

As a rule, there is no need for consistent values for displayed values because it is mostly irrelevant from which IEC task cycle the changed values originate. It is completely irrelevant for seldom changed values. Even when writing there are almost no hard consistency demands because typically the machine must be in a kind of standby mode (for example when writing recipes) in which there is no direct access to the values written as recipes.

In contrast, consistent values are particularly necessary for database links to save production data. For clocked machines, however, these values must be synchronous with the production timing (one value set per produced product) and not consistent with reference to one or more IEC tasks. With reference to the machine clocking, the consistency must be already ensured by the IEC application. For this purpose, the values that arise during a production cycle are typically collected in a global variable list. At the end of the cycle, the symbolic client is notified by means of an additional variable (`BOOL` or counter) that the machine cycle has ended and the values are valid. Now the client has the chance to archive the values from the production cycle. Depending on necessity, the successful reading can also be displayed in the opposite direction by means of a released variable, so that the production can also be halted in case the production data cannot be archived. Synchronized consistent access is not necessary and helpful for this use case because the synchronization takes place at the application level.

In contrast, synchronized consistent access by symbolic clients is typically applied in the process industry with continuously running systems without production clocking when, for example when process values are written consistently and cyclically in a fixed time frame of 60s. This can take place either by synchronization on the application level similar to clocked machines (see above) or by synchronization of the synchronized consistent symbolic access. The advantage of the latter is that no logic has to be implemented in the IEC program and access is controlled entirely by the client.

CAUTION!

Due to the increased jitter, the synchronized consistent monitoring is not suitable for motion or real-time critical applications. For these reasons, synchronized consistent access should be released and used only if it is absolutely necessary.

If a client uses synchronous consistent access released by this setting, then it has and effect on the client. Depending on the scheduler of the runtime, the response time can jitter more here for read/write access because the system might still have to wait for an execution gap of the IEC tasks. Read and/or write access can still fail when IEC tasks run for a long time (in the range of several 100 ms) or the CPU load is close to 100% for an extended period of time with one or more IEC tasks (in the range of several 100 ms). Therefore, the availability of the values also depends on the load of the controller by the IEC application.

Moreover, the client can minimize the effects on itself and on the runtime if it observes the following in the definition of the variable lists to be read or written:

- Synchronized consistent access only to those variables that are absolutely and consistently required.
- Separate variable lists for variables that have to be consistent and for variables that could be inconsistent.
- Divide variable lists with several consistent variables into several smaller lists.
- Select read intervals for cyclic reading of values as large as possible.

Support for the current configuration and possible corrective actions

Entries marked in red in the symbol table show variables that they are configured for export to the symbol file but are currently invalid in the application. The cause for this can be that the declaration has been removed from the block.

In version 3.5.8.0 and higher, a warning appears in the editor if variables that have configured symbols are not used in the IEC code or are not mapped in the case of I/O variables. In addition, the compiler indicates variables that are referenced from outdated library versions n the symbol configuration.

NOTICE!

Object variables that are not used in the program code remain uncompiled by default and are therefore not available in the symbol configuration.

However, CODESYS provides variables from uncompiled objects in the symbol configuration when one of the following conditions is met:

– The *“Link always”* POU property is selected.
– The `{attribute 'linkalways'}` pragma is used.

See also

- *Chapter 1.4.1.9 “Working with Controller Networks” on page 352*
- *Chapter 1.4.1.20.4.10.19 “Dialog 'Properties' - 'Options'” on page 1169*
- *Chapter 1.4.1.20.4.10.4 “Dialog 'Properties' - 'Build'” on page 1159*
- *Chapter 1.4.1.19.6.2.24 “Attribute 'linkalways'” on page 708*
- *Chapter 1.4.1.19.6.2.44 “Effects of Pragmas on Symbols ” on page 729*

Example for the data layout types

Examples for the layout types

The following examples from an IEC application will show how gaps can result in the client-side memory layout caused by unpublished symbols, internal "invisible" pointers, or a "pack mode" definition in the device description. With the *"Optimized layout"* setting, the gaps are avoided. The symbol file contains different information about the size and offset of memory locations, depending on the selected layout setting.

Example: Large structure

```
// Example of a big structure, where not all members get published :
STRUCT
    {attribute 'symbol':='readwrite'}
    PublicNumber : INT;

    {attribute 'symbol':='none'}
    InternalData : ARRAY[0..100] OF BYTE;

    {attribute 'symbol':='readwrite'}
    SecondNumber : INT;

    {attribute 'symbol':='none'}
    MoreData : ARRAY[0..100] OF BYTE;
END_STRUCT
END_TYPE
```

Resulting entries in the symbol file; pay attention to "size" and "byteoffset":

Symbol file, large structure, compatibility layout option

```
<TypeUserDef name="T_LargeStructure" size="208" nativesize="208"
typeclass="Userdef" pouclass="STRUCTURE" iecname="LargeStructure">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0"
vartype="VAR" />

<UserDefElement iecname="SecondNumber" type="T_INT"
byteoffset="104" vartype="VAR" />

</TypeUserDef>
```

Symbol file, large structure, optimized layout option

```
<TypeUserDef name="T_LargeStructure" size="4" nativesize="208"
typeclass="Userdef" pouclass="STRUCTURE" iecname="LargeStructure">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0"
vartype="VAR" />

<UserDefElement iecname="SecondNumber" type="T_INT" byteoffset="2"
vartype="VAR" />

</TypeUserDef>
```

Example: Structure with uneven addresses

```
// The following mechanisms can cause memory misalignment:
// - {attribute 'relative_offset':='…'} at a member
// - {attribute 'pack_mode':='…'} at a structure declaration
// - target setting 'memory-layout\pack-mode' in the device
description

{attribute 'pack_mode':='1'}
TYPE UnevenAddresses:
STRUCT
    {attribute 'relative_offset':='3'}
    {attribute 'symbol':='readwrite'}
    PublicNumber : INT;
```

```
    {attribute 'symbol':='readwrite'}
    PublicValue : LREAL;
END_STRUCT
END_TYPE
```

Resulting entries in the symbol file; pay attention to "size" and "byteoffset":

Symbol file, structure with uneven addresses, compatibility layout option

```
<TypeUserDef name="T_UnevenAddresses" size="13" nativesize="13"
typeclass="Userdef" pouclass="STRUCTURE" iecname="UnevenAddresses">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="3"
vartype="VAR" />

<UserDefElement iecname="PublicValue" type="T_LREAL" byteoffset="5"
vartype="VAR" />

</TypeUserDef>
```

Symbol file, structure with uneven addresses, optimized layout option

```
<TypeUserDef name="T_UnevenAddresses" size="16" nativesize="13"
typeclass="Userdef" pouclass="STRUCTURE" iecname="UnevenAddresses">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0"
vartype="VAR" />

<UserDefElement iecname="PublicValue" type="T_LREAL" byteoffset="8"
vartype="VAR" />

</TypeUserDef>
```

Example: Function block

```
// Each POU contains some implicit variables, which do not get
published. Depending on the data type these might cause memory gaps
of different sizes.
FUNCTION_BLOCK POU IMPLEMENTS SomeInterface
VAR_INPUT
    in : INT;
END_VAR
VAR_OUTPUT
    out : INT;
END_VAR
VAR
END_VAR
```

Each POU contains some implicit variables, which do not get published. If it is a data type such as `__XWORD`, then different sizes of memory gaps result in the client-side data layout, depending on whether the system is 64-bit or 32-bit.

Resulting entries in the symbol file for 64-bit and 32-bit; pay attention to "size" and "byteoffset":

Symbol file, function block, compatibility layout option, 64-bit

```
<TypeUserDef name="T_POU" size="24" nativesize="24"
typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="POU">

<UserDefElement iecname="in" type="T_INT" byteoffset="16"
vartype="VAR_INPUT" />

<UserDefElement iecname="out" type="T_INT" byteoffset="18"
vartype="VAR_OUTPUT" />

</TypeUserDef>
```

Symbol file, function block, optimized layout option, 64-bit

```
<TypeUserDef name="T_POU" size="4" nativesize="24"
typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="POU">

<UserDefElement iecname="in" type="T_INT" byteoffset="0"
vartype="VAR_INPUT" />
```

```
                <UserDefElement iecname="out" type="T_INT" byteoffset="2"
                vartype="VAR_OUTPUT" />

                </TypeUserDef>
```

Symbol file, function block, compatibility layout option, 32-bit

```
                <TypeUserDef name="T_POU" size="12" nativesize="12"
                typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="POU">

                <UserDefElement iecname="in" type="T_INT" byteoffset="8"
                vartype="VAR_INPUT" />

                <UserDefElement iecname="out" type="T_INT" byteoffset="10"
                vartype="VAR_OUTPUT" />

                </TypeUserDef>
```

Symbol file, function block, optimized layout option, 32-bit

```
                <TypeUserDef name="T_POU" size="4" nativesize="12"
                typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="POU">

                <UserDefElement iecname="in" type="T_INT" byteoffset="0"
                vartype="VAR_INPUT" />

                <UserDefElement iecname="out" type="T_INT" byteoffset="2"
                vartype="VAR_OUTPUT" />

                </TypeUserDef>
```

See also

- *Chapter 1.4.1.9.2 “Symbol Configuration” on page 357*

Object 'Task Configuration'

Symbol:

The object is used to define and display the basic settings for the task configuration.

The *“Task Configuration”* object must be included exactly one time in each application.

“Task Configuration” tabs and functions

- *“Properties”*: Display of the basic settings
- *“System Events”*: Linking of POU calls with system events
- *“Monitor”*: Display of the status and current statistics for the cycles times in online mode
- *“Variable Usage”*: Overview of the tasks that access the variables and how they do it
- *“Task Groups”*: Definitions of the tasks groups and their assignment to CPUs
- *“CPU Load”*: Graphical representation of the CPU load in online mode

See also

- *Chapter 1.4.1.8.16.1 “Creating a task configuration” on page 293*
- *Chapter 1.4.1.20.2.26.1 “Tab 'Properties'” on page 938*
- *Chapter 1.4.1.20.2.26.2 “Tab 'System Events'” on page 938*
- *Chapter 1.4.1.20.2.26.3 “Tab 'Monitor'” on page 940*
- *Chapter 1.4.1.20.2.26.4 “Tab 'Variable Usage'” on page 941*
- *Chapter 1.4.1.20.2.26.5 “Tab 'Task Groups'” on page 941*
- *Chapter 1.4.1.20.2.26.6 “Tab 'CPU Load'” on page 942*
- *Chapter 1.4.1.20.2.27 “Object 'Task'” on page 942*

Tab 'Properties'

Object: *“Task Configuration”*

In this tab, you define the basic settings of the task configuration as predefined by the target system, such as the maximum values for tasks and watchdog parameters.

Tab 'System Events'

Object: *“Task Configuration”*

On the *“System Events”* tab, you define which event calls which function and whether or not the configuration is currently activated. You use this tab when a system event (instead of a task) should call a project function.

“Add Event Handler”	Opens the *“Add Event Handler”* dialog
“Remove Event Handler”	Deletes the selected list assignment
“Event Info”	Shows information from the corresponding event library
“Open Event Function”	Opens the editor of the new function for the selected assignment You have selected the implementation language of the new function in the *“Add Event Handler”* dialog.
Assignment of functions to call for events with: *“Name”*,*“Description”*, *“Function to call”*, and *“Active”* (activate/deactivate configuration).	

Table 101: “Add Event Handler”

Adds a new assignment "Event – Function to call" to the list	
“Event”	The possible selection depends on the target device. CODESYS marks unavailable events with a red symbol in front of the name. A list of all possible system events is located at the end of this section.
“Function to call”	Function name (*“POU”*, type *“FUNCTION”*) You have to specify the name of the **new** function. CODESYS inserts the function to the device tree after you confirm the dialog.
“Scope”	● *“Application”*: The function is available to the application. ● *“POUs”*: The function is available to the entire project.
“Implementation language”	Implementation language for the new function
“Description”	Short description of the selected event

Features in *“Online Mode”*

The list of assignments from called functions to events also includes the following information: *“Event Status”*, *“Call Count”*, and the *“Online Reset”* button.

“Event Status”	`0`: No error has occurred. Does not equal `0`: Error. You need to consult the respective runtime system documentation.
“Call Count”	Displays how often the event has occurred or the associated function has been called.
“Online Reset”	CODESYS reinitializes the event lists and resets the counter for the events/function calls. Incorrectly initialized events are displayed with a red status cell.

Possible system events

Event	Description	Task	Debugging
PrepareStart	Call before starting the application	Communication task	No
StartDone	Call after starting the application	Communication task	No
PrepareStop	Call before stopping the application	Communication task	No
StopDone	Call after stopping the application	Communication task	No
PrepareReset	Call before resetting the application	Communication task	No
ResetDone	Call after resetting the application	Communication task	No
PrepareOnline-Change	Call before online change of the application	Communication task	No
OnlineChangeDone	Call after online change of the application	Communication task	No
PrepareDownload	Call before downloading the application	Communication task	No
DownloadDone	Call after downloading the application	Communication task	No
PrepareDelete	Call before deleting the application	Communication task	No
DeleteDone	Call after deleting the application	Communication task	No
PrepareExit	Call before exiting the application	Communication task	No
ExitDone	Call after exiting the application	Communication task	No
CodeInitDone	Event is sent after Code Init. Called within the task safe section and only for an online change change (for example, the copy code for online change is executed here).	Communication task	No
Exception	The event is sent if an exception has occurred in the context of an application.	Exception handling task of the runtime, or the task itself if the runtime does not support exception handling	Depends on the task
Login	Login of a client to this application	Communication task	No
Logout	Logout of a client from this application	Communication task	No
BeforeReadingInputs	Call before reading the inputs	IEC task	Yes
AfterReadingInputs	Call after reading the inputs	IEC task	Yes

Event	Description	Task	Debugging
BeforeWritingOutputs	Call before writing the outputs	IEC task	Yes
AfterWritingOutputs	Call after writing the outputs	IEC task	Yes
DebugLoop	Event is sent in cycles to the debug loop if the IEC task stops at a breakpoint.	Communication task	No
PrepareShutdown	Event is sent immediately before the runtime system is downloaded.	Runtime main loop	No
PrepareExitComm	Event is sent during download before exiting the communication server.	Runtime main loop	No
PrepareExitTasks	Event is sent during download before exiting all tasks.	Runtime main loop	No

Tab 'Monitor'

Object: *"Task Configuration"*

In online mode, the tab shows the status of the tasks of the task configuration, as well as some current measurements of the cycles and cycle times. CODESYS updates the values in the same time interval as for the monitoring of values from the PLC.

The displayed values can be reset to `0` by means of the *"Reset"* context menu command.

"Task"	Task name (as defined in the task configuration)
"Status"	● *"Not created"*: The task has not been started since the last update (especially for event tasks). ● *"Generated"*: The task is recognized in the runtime system, but not yet in operation. ● *"Valid"*: The task is operating normally. ● *"Exception"*: The task has produced an exception status.
"IEC-Cycle Count"	Number of cycles executed since starting the application where the IEC code was executed (`0` if the target system does not support the counter function)
"Cycle Count"	Number of executed cycles since logging in to the PLC It depends on the target system whether cycles are also counted where the application is not running. In these cases, the *"Cycle Count"* may be greater than the *"IEC-Cycle Count"*.
"Last Cycle Time (µs)"	Last measured cycle time [µs]
"Average Cycle Time (µs)"	Average cycle time over all cycles [µs]
"Max. Cycle Time (µs)"	Maximum measured cycle time over all cycles [µs]
"Min. Cycle Time (µs)"	Minimum measured cycle time over all cycles [µs]
"Jitter (µs)"	Current value of the periodic jitter [µs] Note: From CODESYS 3.5 SP11 to SP15, the peak-peak value of the periodic jitter is displayed. In earlier versions and in SP16 and later, the current value of the periodic jitter is displayed.

“Min. Jitter (µs)”	Minimum measured periodic jitter [µs]
“Max. Jitter (µs)”	Maximum measured periodic jitter [µs]
“Core”	Number of the processor core where the task is currently running Example: `2` Requirement: The controller is equipped with a multicore processor. If the CPU is not a multicore CPU, then the value `-1` is displayed here.

See also

- *Chapter 1.4.1.8.16.2 “Definitions of Jitter and Latency” on page 294*
- *Chapter 1.4.1.8.16 “Task Configuration” on page 292*

Tab 'Variable Usage'

Object: *“Task Configuration”*

The *“Variable Usage”* tab provides an overview of all variables and their usage. There you can see the tasks where variables are accessed.

When using multicore, write access (w) to a variable should take place only in a task because otherwise it can cause inconsistencies.

In the context menu, you can hide individual tasks and show the cross-reference list to variables.

“Variables”	Name of the variable
“Type”	Data type
“Number”	Number of tasks that access these variables.
“<task name>”	Access to the variable (r: read, w: write, rw: read/write)

See also

- *Chapter 1.4.1.8.16 “Task Configuration” on page 292*

Tab 'Task Groups'

Object: *“Task Configuration”*

You define task groups on the *“Task Groups”* tab. Task groups can be distributed over the individual processor cores in multicore systems. The tasks of a task group are bound to the processor cores according to the strategy defined in the *“Core”* field.

“Add Group”	The button adds a new task group named `NewGroup_<no>`.
“Remove Group”	Deletes the selected task group.

“Group Name”	The name can be changed by double-clicking in the field.
“Core”	Determines the processor core for process the tasks of this group. ● *“Free floating”*: All tasks are bound dynamically to different processor cores. The user does not have any influence over this. The operating system is responsible for the distribution. ● *“Sequentially pinned”*: All tasks are bound and fixed to different processor cores. The user does not have any influence over this. ● *“Fixed pinned”*: All tasks are bound to one processor core. By default, the runtime system determines the processor core. ● *“<core number>”*: Fixed defined processor core. If the processor core is not available, then an error message is issued.

See also

- *Chapter 1.4.1.8.16 “Task Configuration” on page 292*

Tab 'CPU Load'

Object: *“Task Configuration”*

The *“CPU Load”* tab is available in online mode for multicore devices only. The load of the individual CPUs is presented in the trace editor.

You open the trace configuration by double-clicking the legend in the window on the right side. Adding more variables is not possible here.

See also

- *“Displaying the CPU load with DeviceTrace objects in the CODESYS project (example)” on page 429*

Object 'Task'

Symbol: 

In this object, you define the conditions for starting and calling the task.

You insert the object below *“Task Configuration”* in the device tree.

Tab 'Configuration'

Object: *“Task”*

“Priority”	Possible values: 0..31, where 0 is the highest priority
“Task group”	Assigned task group. This assignment is shown in parentheses in the device tree. Task groups can be assigned to specific processor cores in multicore. The task group is shown in parentheses after the task in the device tree.

Table 102: “Type”

“Cyclic”	CODESYS processes the task in cycles. The cycle time of the task is defined in the input field *“Interval”*.
“Event”	CODESYS starts processing the task as soon as the global variable defined in the input field *“Event”* contains a rising edge.
“Freewheeling”	CODESYS starts processing the task again automatically in a continuous loop at program start and at the end of a complete pass. The cycle time is not defined.

“Status”	CODESYS starts task processing as soon as the variable defined in the *“Event”* input field yields the Boolean value `TRUE`.
“External”	CODESYS starts processing the task as soon as the event defined in the *“Event”* input field occurs. The target system determines which events are supported and offered in the list box. (Not to be confused with system events).
“Interval”	Task-cycle time Required for the types *“Cyclic”* or *“External Event”* when the event requires a given time. Time span after which the task should be restarted. If you enter a number here, then you can select the desired unit in the list box after the input field. When you select *“ms”*, an entry is automatically displayed in TIME format, for example `t#200ms`, as soon as the window is in focus again. You can also enter the task cycle time directly in TIME format. Entries in [µs] format are always displayed as a pure number. Deviations of the task from this desired task cycle time are displayed at runtime as periodic jitter on the *“Watchdog”* tab.

NOTICE!
For fieldbuses, a fixed cycle matrix is necessary to assure a determined behavior. Therefore, you should not use *“Type” “Freewheeling”* for a bus cycle task.

NOTICE!
Note the following difference between the processing types *“Status ”* and *“Event”*. If the given event yields `TRUE`, then the start condition of a task of type *“Status”* is fulfilled. In contrast, the start of a task of type *“Event”* requires a switch of the event from `FALSE` to `TRUE`. If the sampling rate of the task scheduler is too low, then the rising edge of the event can remain unnoticed.

NOTICE!
When setting the task cycle time, you have to identify which bus system is currently being used. For example, the task cycle time in a CAN bus system must match the currently set baud rate and the number of frames used in the bus. In addition, the times set for heartbeat, node guarding, and sync should always be a multiple of the task cycle time. If not, then CAN frames can be lost.

Table 103: “Watchdog”

Defines the time monitoring for a task. If the target system supports an advanced watchdog configuration, then the following settings may be predefined in the device description. ● Upper and lower limit ● Default watchdog time ● Time specified as percentage The default watchdog settings depend on the device.	
“Enable”	The watchdog is active. If the task exceeds the currently set *“Time”* of the watchdog, then the task is halted with an error status (exception). The application in whose task the error occurred and its child applications are also halted. In this way, all tasks of the affected applications are also halted. Then the currently defined *“Sensitivity”* is also taken into account. If you activate the option *“Update I/Os”* in the *“PLC Settings”* of the PLC, then CODESYS resets the outputs to the defined default values. Possible cases: ● Multiple consecutive timeouts: Sensitivity: 0, 1 - exception in cycle 1 Sensitivity: 2 - exception in cycle 2 Sensitivity: n - exception in cycle n ● Single timeout: Exception if the cycle time of the current cycle is longer than (time * sensitivity). Example: Time=t#10ms, Sensitivity=5 (i.e., exception as soon as the one-time task runs longer than 50 ms)
“Time (e.g. t#200ms)”	Watchdog time Defines (together with *“Sensitivity”*) the watchdog for a task; description as for *“Enable”*. Depending on the target system, the monitoring time span is given as a percentage of the task interval if possible. In this case, the list box for the unit is disabled and displays *“%”*.
“Sensitivity”	Number Defines (together with the watchdog) the watchdog for a task; description as for *“Enable”*.

Using the functions from the library `CmpIecTask.library`, you can deactivate a watchdog for specific PLC cycles. This is useful for cycles that demand more time due to initialization.

Example Deactivating/reactivating the watchdog:

```
hIecTask := RTS_IEC_HANDLE //Declaration of the variable hIecTask
hIecTask := IecTaskGetCurrent(0);
IecTaskDisableWatchdog(hIecTask); // Watchdog disabled
...
IecTaskEnableWatchdog(hIecTask); Watchdog enabled
```

List of *“POU”*s that the task calls The calling order corresponds to the POU order in the list (from top to bottom).	
“Add Call”	Defines a new program call

“ Open POU”	Opens the selected POU
“Move Up” *“Move Down”*	Changes the calling order

See also

- *Chapter 1.4.1.20.2.26 “Object 'Task Configuration'” on page 937*
- *Chapter 1.4.1.20.2.26.3 “Tab 'Monitor'” on page 940*
- *Chapter 1.4.1.20.2.26.5 “Tab 'Task Groups'” on page 941*

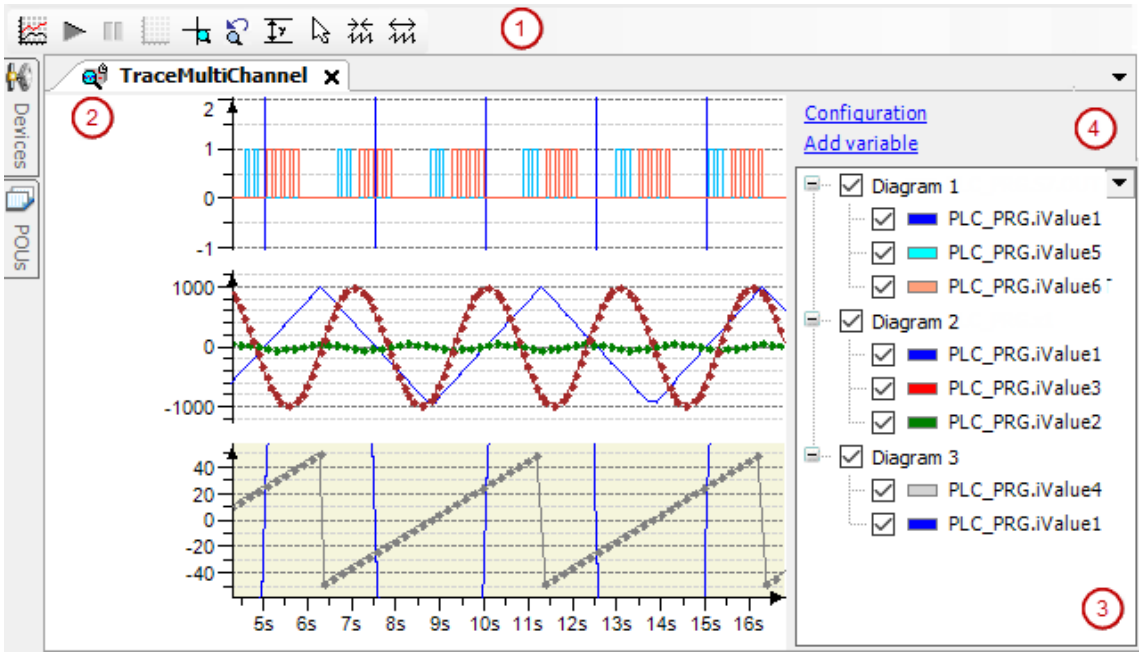
Object 'Trace'

Symbol:

An object of type *“Trace”* is used for configuring and displaying application-specific trace data in one or more charts. At application runtime, value curves of trace variables, which you can monitor in the trace editor in CODESYS, are recorded on the controller. Requirements are that a trace configuration has been configured transferred to the controller, and the trace recording has been started. The recorded data is transferred to the development system and displayed in diagrams according to the configuration. You can navigate through the data when tracing.

> *If the controller supports a Trace Manager, then you can use the 'DeviceTrace' object type in the Trace Manager to access all traces that are running on the controller.*

Double clicking the trace object opens the trace editor. The corresponding toolbar contains the most important trace commands. The trace variable list shows the variable whose value curve is recorded.



- (1): Toolbar of the trace editor
- (2): Trace editor
- (3): Trace variable list
- (4): Links for trace configuration
  “Configuration”
  “Add Variable”

See also

- *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*
- *Chapter 1.4.1.20.2.29 “Object 'DeviceTrace'” on page 948*

Toolbar of the trace editor

See also

-  : *Chapter 1.4.1.20.3.21.6 “Command 'Download Trace'” on page 1138*
-  : *Chapter 1.4.1.20.3.21.16 “Command 'Start Trace'” on page 1145*
-  : *Chapter 1.4.1.20.3.21.17 “Command 'Stop Trace'” on page 1145*
-  : *Chapter 1.4.1.20.3.21.13 “Command 'Reset Trigger'” on page 1144*
-  : *Chapter 1.4.1.20.3.21.9 “Command 'Mouse Zooming'” on page 1141*
-  : *Chapter 1.4.1.20.3.21.2 “Command 'AutoFit'” on page 1137*
-  : *Chapter 1.4.1.20.3.21.5 “Command 'Cursor'” on page 1137*
-  : *Chapter 1.4.1.20.3.21.3 “Command 'Compress'” on page 1137*
-  : *Chapter 1.4.1.20.3.21.3 “Command 'Compress'” on page 1137*
-  : *Chapter 1.4.1.20.3.21.18 “Command 'Stretch'” on page 1146*

Trace editor

At application runtime, the runtime system buffer of the trace component is filled with the recorded samples. The data is transferred to the development system and stored in its trace editor buffer. The trace editor accesses this data and displays it in diagrams as a graph over time. When you close the trace editor, the trace editor buffer will be freed.

Use menu commands for controlling the trace. In addition, you can use menu commands, keyboard shortcuts, and mouse input for navigating through the data.

See also

- *Chapter 1.4.1.12.3.3 “Operating the data recording” on page 427*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Trace variable list

The trace variable list provides an overview of the current trace configuration. In the list, all charts with the respective trace variables are displayed in a table. When you double-click a trace variable, the *“Trace Configuration”* dialog also opens with the variable settings.

	A list box opens by means of the *“Hide Instance Paths”* command.
“Hide Instance Paths”	Display of the variable name in the list ●  : Variable name with full instance path Example: `PLC_PRG.iCounter`

Table 104: Charts

Tabular display of the charts:	
“Name”	List of charts with the respective variables ● *“Chart <n>”*☑: The chart is displayed. The chart name can be changed by clicking the selected name. ● *“<variable>”*☑: The variable is displayed. The variable name can be changed by clicking the selected name in the line editor. When you select a *“Chart <n>”* in the table, the corresponding chart is also selected in the editor. This also works the other way around.
“Cursor <n>”	Y-value at the cursor position
“Delta”	Delta of the Y-value from *“Cursor 2”* to *“Cursor 1”*

> *You can drag the charts and variables to sort them or move them to other diagrams. When the [Ctrl] key is pressed, the variable is copied. This is also possible in online mode.*

Table 105: Context menu in the trace variable list

"Add Variable"	Adds a new trace variable and opens the *"Trace Configuration"* dialog with its variable settings. Select a variable in the input field of the *"Variable"* setting to trace its value curve.
"Visible"	Toggles the visibility of the graph (value curve or trace variable) in the corresponding diagrams: ● : Visible. ● : Invisible.
"Display Mode"	Opens the *"Trace Configuration"* dialog. Select a configuration item in the *"Trace Record"* tree view or *"Presentation (Diagrams)"*.
"Configuration"	Opens the *"Trace Configuration"* dialog. The *"Variable Settings"* are displayed on the right.

See also

- *Chapter 1.4.1.20.4.15.2 "Dialog 'Trace Configuration'" on page 1209*

Navigating in the diagram

Table 106: With mouse input

User input with the mouse	**Mouse cursor symbol during user input**	**Effect**
Drag the graph along the X-axis.		Scrolls trace graphs of all diagrams at the same time along the time axis (X-axis).
Hold down the *[Ctrl]* key and drag the graphs along the Y-axis.		Scrolls the trace graphs of the selected diagrams along the Y-axis.
Roll the mouse wheel backwards.		Compressed time axis (like the symbol).
Roll the mouse wheel forwards.		Stretches time axis (like the symbol).
Press and hold down the *[Ctrl]* key and roll the mouse wheel backwards.		Compresses the Y-axis.
Press and hold down the *[Ctrl]* key and roll the mouse wheel forwards.		Stretches the Y-axis.
Requirement: One or two trace cursors are activated. Drag the triangle of a trace cursor to another position along the time axis.		Refreshes the Y-values in the trace variable list at the same time ● First value: Y-value at the position of the left cursor. ● Second value: Y-value at the position of the right cursor. ● Third value: Difference between both values.
Requirement: *"Mouse zooming"* is activated ( ). Stretch a rectangle.		Zooms the trace graphs of all diagrams to the rectangle.

Table 107: With keyboard shortcuts

Shortcut	Effect
Requirement: No trace cursor is activated. *[Arrow Left]* *[Arrow Right]*	Scrolls trace graphs of all diagrams at the same time along the time axis.
[Arrow Up] *[Arrow Down]*	Scrolls the trace graphs of the selected diagrams along the Y-axis.
Requirement: One or two trace cursors are activated. *[Alt]+[Arrow Left]* *[Alt]+[Arrow Right]*	Scrolls trace graphs of all diagrams at the same time along the time axis.
[-]	Compressed time axis (like the symbol).
[+]	Stretches the X-axis (like the symbol).
[Ctrl]+[-]	Compresses the Y-axis of the selected diagram.
[Ctrl]+[+]	Stretches the Y-axis of the selected diagram.
[Tab]	Selects the next lower diagram.
Requirement: One or two trace cursors are activated. *[Arrow Left]* *[Arrow Right]*	Moves the black trace cursor.
Requirement: Two trace cursors are activated. *[Shift]+[Arrow Left]* *[Shift]+[Arrow Right]*	Moves the gray trace cursor.

See also

- *Chapter 1.4.1.20.3.21.5 “Command 'Cursor'” on page 1137*
- *Chapter 1.4.1.20.3.21.9 “Command 'Mouse Zooming'” on page 1141*
- *Chapter 1.4.1.12 “Application at Runtime” on page 409*

Object 'DeviceTrace'

Symbol:

A *“DeviceTrace”* object shows trace data in one or more diagrams, as does a *“Trace”* object.

The difference is that a *“DeviceTrace”* directly accesses traces that are running on the controller. The object is inserted below the device in the device tree. Therefore there is no immediate dependency on the applications in the CODESYS project.

You can use the DeviceTrace for visualizing the processor load of a multicore controller.

For more information about the editor and its operation, refer to the help page for the *“Trace”* object.

See also

- *Chapter 1.4.1.20.2.28 “Object 'Trace'” on page 945*
- *“Runtime system component `CmpTraceMgr`, "Trace manager"” on page 421*
- *Chapter 1.4.1.12.3.4 “Accessing All Traces on the Controller” on page 428*

Object 'Trend Recording Manager'

Symbol 

A *“Trend Recording Manager”* object makes it possible to save data at runtime in a database for a long period of time. This data is recorded with the *“CmpTraceMgr”* runtime system component. In the device tree, this object is used as a node for trend recordings that are created below an application. It is available below an application only one time.

See also

- *Chapter 1.4.1.12.4 “Data Recording with Trend” on page 430*
- *Chapter 1.4.1.20.3.4.1 “Command ‘Add Object’” on page 1001*
- *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*

Object 'Trend Recording'

Symbol: 

A *“Trend Recording”* object is always located below a *“Trend Recording Manager”* and enables editing of the trace configuration. At runtime, CODESYS transfers the configuration that is available to the runtime system component `CmpTraceMgr`. You can configure an application with any number of trend recordings.

NOTICE!

Timeout for trend recording

During a trend recording, it can happen that the application task triggers a timeout that is caught with an exception when transitioning from *“Running”* to *“Stop”*. Causes can be that file operations with the SQLite database are taking too long or that too many variables are being recorded. This usually happens on a target device with weak performance.

You can avoid the occurrence of an exception:

– Configure the trend recording with less memory demand so that the amount of data that is stored is adapted to the target system.
– Reduce the number of variables.

The editor includes the configuration for trend recording. The tree view shows the trend configuration and enables navigation there.

The top entry contains the trend name. When this entry is selected, the *“Record Settings”* are displayed next to it. An entry is located here for each variable whose data was recorded continuously. When a variable is selected, the *“Variable Settings”* are displayed next to it.

“Add Variable”	When you click the link, a new entry is displayed in the trend configuration with its blank configuration below the *“Variable Settings”* group.
“Delete Variable”	The selected variable is removed. Requirement: A variable is selected.

See also

- *Command 'Edit Trend Recording'*

'Recording Settings'

The data is recorded in the runtime system component by means of the functionality which is also used for the trace. The settings that appear here are the same. The options that are not required here are deactivated.

The settings that affect the trigger are deactivated. Only a trace configuration for a trace editor can configure triggering.	
“Task”	Task where data was recorded. Click ▼ to open a list box with all tasks available in the project. In general, the trend recording runs in the same task as the main program. For example: `MainTask`
“Recording condition”	Condition under which the application records data: ● IEC variable of type `INT`. The condition is fulfilled for `TRUE`. ● Bit access to an integer variable. The condition is fulfilled for `1`. As read access to a property. The contents of a pointer are not permitted. Note: If no condition is defined, then the recording starts automatically.
“Comment”	Comment (example: `Data recording of sensor A`)
“Resolution”	Resolution that the application saves the time stamp Note: If the task where the trend object is executed has a cycle time of 1 ms or less, then you should set the resolution of the time stamp to *“1 µs”*.
“Trend Storage”	The *“Trend Storage”* dialog opens.
“Advanced”	The *“Advanced Trend Settings”* dialog opens.

See also

- *Chapter 1.4.1.20.4.16 “Dialog Box 'Trend storage'” on page 1214*
- *Chapter 1.4.1.20.4.17 “Dialog Box 'Advanced Trend Settings'” on page 1214*

'Variable Settings'

“Variable”	Variable for recorded value. ● IEC variable with valid data type ● Property ● Reference ● Contents of the pointer ● Array element of a valid data type ● Enumeration of a valid data type Valid data types are all standard types, **except** `STRING`, `WSTRING`, and `ARRAY`.
“Parameter”	Parameter for the recorded value The *“Input Assistant”* dialog lists all valid system parameters in the *“Parameters”* category of the *“Categories”* tab.
	Click the symbol to toggle between *“Variable”* and *“Parameter”*.
“Recording condition”	Condition under which the application records the data of these *“Variables”*: ● IEC variable of type `INT`. The condition is fulfilled for `TRUE`. ● Bit access to an integer variable. The condition is fulfilled for `1`. As read access to a property The contents of a pointer are not permitted. Note: If no condition is defined, then the recording starts automatically.
“Attached y axis”	Y-axis of the trend diagram that displays the *“Variable”*. The list box provides the standard Y-axis and the configured Y-axes. Requirement: This option is visible only when the *“Trend”* visualization element has configured additional Y-axes in the *“Edit Display Settings”* dialog.

"Display variable name"	☑: The visualization shows the name of the IEC variable in the trend diagram at runtime. Either alone or in parentheses after the *"Description"* ☐: The name of the IEC variable is shown and does not appear in parentheses after the *"Description"*. Requirement: If any text is typed in *"Description"*, then you can disable the option.
"Description"	Text for the tooltip (example: `Sensor A`): When a visualization user focuses on the variable in the trend diagram, the visualization shows the text as a tooltip. The text is typed into the *"GlobalTextList"* object and can be localized there. When the *"Display Variable Name"* property is activated, then the text is supplemented with the variable name in parentheses. Example: `Sensor A (PLC_PRG.iSensor_A)` If *"Description"* does not contain any text, then *"Display Variable Name"* is enabled. The name is then alone without parentheses (for example, `PLC_PRG.iSensor_A`). If a legend is assigned to the trend, then the trend variable is labeled in the legend and shown like the trend is configured here.
"Curve type"	● *"Line"* ● *"Area"*:
"Graph color"	Color of the curve in the trend diagram
"Line type"	● *"Line"*: Values are linked to form a line. ● *"Step"*: Values are linked in the form of steps. ● *"None"*: Values are not linked. Requirement: The *"Curve type"* is *"Line"*.
"Filling type"	● *"No filling"* ● *"Plain color"*: ● *"Gradient"* Requirement: The *"Curve type"* is *"Area"*.
"Filling color"	☑: The area is filled with the selected color. Requirement: The *"Curve type"* is *"Area"*.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color Example `255`: The color is opaque. `0`: The color is completely transparent Requirement: The *"Curve type"* is *"Area"*.
"Line width"	Value (in pixels) Example: `1`
"Line style"	The display of the line is solid, dash, dot, dash-dot, or dash-dot-dot.
"Point type"	Display as scatter chart ● *"Dot"*: Value as a dot. ● *"Cross"*: Value as a cross. ● *"None"*: No dot display Hint: Select *"None"* for larger size data.
"Activate minimum warning"	☑: Warning when below the lower limit.
"Critical lower limit"	If the variable value is below the limit, then the variables are displayed with the alert color in the trend diagram.
"Color"	Warning color on falling below the limit
"Activate maximum warning"	☑: A warning is issued if the upper limit is exceeded.

“Critical upper limit”	If the variable value exceeds the limit, then the variables are displayed with the alert color in the trend diagram.
“Color”	Warning color on exceeding the limit

See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Dialog 'Display Settings'*
- *Visualization Element 'Legend'*

See also

- *Chapter 1.4.1.20.3.4.1 “Command ‘Add Object’” on page 1001*
- *Chapter 1.4.1.12.4 “Data Recording with Trend” on page 430*
- *Visualization Element 'Trend'*

Object 'Trend Recording Task'

Symbol 

If you design a visualization with a *“Trend”* element, then CODESYS automatically extends the *“Task Configuration”* with a *“Trend Recording Task”*. The task is below an application one time at most and calls the `VisuTrendStorageAccess.GlobalInstances.g_TrendRecordingManager.CyclicCall` program to run the trend recording manager.

See also

- *Chapter 1.4.1.20.3.4.1 “Command ‘Add Object’” on page 1001*
- *Chapter 1.4.1.12.4 “Data Recording with Trend” on page 430*

Object 'Unit Conversion'

Symbol 

A *“Unit Conversion”* object is used to define a conversion rule. The following table lists all defined conversion rules. You can edit a conversion rule in the input fields listed below the table.

Table of conversion rules

“Name”	`<name> : <name>_Impl` is the name of the conversion rule. CODESYS automatically implements the entry as a function block `<name>_Impl` and instances it as `<name>`.
“Type”	Type of conversion rule ● *“Single scaling (offset)”*: adds an offset to the input variable. `Result := Input + Offset` ● *“Single scaling (factor)”*: multiplies the input variable by a factor. `Result := Input * Factor` ● *“Linear scaling 1 (factor and offset)”*: converts the input variable with a factor and offset. `Result := Input * Factor + Offset` ● *“Linear scaling 2 (Base and target range)”*: converts the input variable for the output value to be within a target range. CODESYS calculates the functional linear equation internally. ● *“User defined conversion”*: configures a user-defined conversion rule with IEC operators. The input variable is `rValue`. ● *“Switchable conversion”*: defines a conversion rule that CODESYS executes independent of any specified language or variable.
“Setting”	Displays the configured conversion rule.
“Condition”	● *“TRUE”*: CODESYS always executes the conversion. ● *“Language”* If the language in the visualization is the language defined here, then CODESYS executes the conversion. The current visualization language is located in the `VisuElems.CurrentLanguage` variable. ● *“Variable”*: If the comparison is TRUE, then CODESYS executes the conversion rule. CODESYS can pass the comparison for a constant, variable, or IEC expression. You can edit the comparison below the table in the *“Condition Setting”*.
“Condition Setting”	If you select *“TRUE”* as the *“Condition”*, then the field is hidden. If you configure *“Language”* as the *“Condition”*, then the field shows the current configuration, for example `en,de`. If you select *“Variable”* as the *“Condition”*, then the field shows the current configuration, for example `PLC_PRG.bActual=PLC_PRG.bSet`. You can edit the current condition setting below the table in the input fields for *“Condition Setting”*.

Input field 'Single scaling (offset)'

The input variable is added with an offset.

“Offset”	● as a number, including `REAL` ● as an IEC variable

Input field 'Single scaling (factor)'

The input variable is multiplied by the factor.

“Factor”	● as a number, including `REAL` ● as an IEC variable

Input field 'Linear scaling 1 (factor and offset)'

The input variable is converted with the linear equation defined below.

“Factor”	● as a number, including `REAL` ● as an IEC variable
“Offset”	● as a number, including `REAL` ● as an IEC variable

Input field 'Linear scaling 2 (Base and target range)'

The input variable is converted to be within a target range. CODESYS internally creates a linear equation from the following input values.

“Base start value”	Lowest possible value for the input variable. ● as a number, including `REAL` ● as an IEC variable
“Base end value”	Highest possible value for the input variable. ● as a number, including `REAL` ● as an IEC variable
“Target start value”	Lowest possible value for the output variable. ● as a number, including `REAL` ● as an IEC variable
“Target end value”	Highest possible value for the output variable. ● as a number, including `REAL` ● as an IEC variable

Example Conversion of electric current from a 10-bit input signal to an amperage range of 4-20 mA

“Base start value”	`0`
“Base end value”	`1024`
“Target start value”	`4.0`
“Target end value”	`20.0`

Input field 'User defined conversion'

“Convert :=”	Conversion rule as mathematical function of `rValue` The input variable is `rValue`.
“Reverse :=”	Reverse function of the function defined in *“Convert”*

Input field 'Switchable conversion'

Use this conversion rule when you want to apply a conversion that is language-specific or variable-dependent.

“Switchable conversion name”	Selected from a list of predefined conversion rules. Double-click directly into the field for editing.
“Condition setting”“”	Configured condition. Click into the input fields in *“Condition setting”* to edit the condition.

Example

The `Conv_A_LanguageDependent` conversion rule that defines which conversion rule is executed for the English or German language.

"Name"	*"Type"*	*"Setting"*	*"Condition"*	*"Condition setting"*
`Conv_A_LanguageDependent`	*"Switchable conversion"*	`Conv_AInInch, Conv_AInMM`	*"Language"*	

"Switchable conversion name"	*"Condition setting"*
`Conv_AInInch`	`en`
`Conv_AInMM`	`de`

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

1.4.1.20.3 Menu Commands

By default the most important commands are already provided in the CODESYS user interface. If you want to customize the menu configuration individually, choose command *"Tools ➔ Customize ➔ Menu"*.

When you have installed any packages or add-ons, additional menus and commands might be available.

Menu 'File'

Command 'New Project'

Symbol:  , Shortcut: *[Ctrl]* + *[N]*

Function: This command opens the *"New Project"* dialog box for the creation of a new project file.

Call: *"File"* menu

'New Project' dialog box

Function: Selection of a project category and a project template.

Call: *"File ➔ New Project"*

Depending on the template, you obtain a project that is automatically equipped with a certain range of objects.

Table 108: "Categories"

"Libraries"	
"Projects"	

Table 109: "Templates"

"Projects" category::	
"Empty project"	Contains only the *"Project Settings"* object
"Standard project"	Contains a basic range of objects and libraries. A wizard assists with the creation – see below.
"Standard project with Application Composer"	Contains a basic range of objects and libraries for working with the Application Composer. A wizard assists with the creation.
"Libraries" category:	
"CODESYS container library "	Library that contains only further libraries, but no function blocks of its own.
"CODESYS interface library "	Library only for the definition of the interface of a software component. Thus contains only objects that do not generate any code (constants, structures, interfaces, etc.).
"Empty library"	Contains only the *"Project Settings"* object
" External CODESYS library "	Target-system-specific library which is implemented as part of the runtime system (in ANSI C or C++).
"Name"	Name of the project to be created. Depending on the template, a standard name appears. The numerical suffix ensures the uniqueness of the name in the file system. You can change the file name, taking into consideration the file path conventions of the operating system. Periods are not permitted in names. CODESYS automatically adds the appropriate file extension to the selected template.
"Location"	Location for the new project file. ... opens a dialog box for browsing the file system. ▼ displays the history of previously entered paths
"OK"	CODESYS opens a new project. An error symbol ⓘ next to the input field draws attention to missing specifications. If you place the mouse pointer on it, a tooltip appears, informing you what to do.

***"Standard Project"* dialog box**

Function: Wizard for the creation of a standard project.

Call: Command *"File ➔ New Project"*; in the *"New Project"* dialog box, select the *"Projects"* category and the *"Standard project"* template and click on *"OK"*.

“Device”	Selection list with PLC devices. The selected device is inserted as an object in the Devices view below the root node
“PLC_PRG in”	Selection list with the programming languages. The automatically inserted program PLC_PRG is created in the selected language.

Command 'Open Project'

Symbol:  ; shortcut: *[Ctrl]+[O]*

Function: The command opens the default dialog for loading a project. You can search for a CODESYS project in the file system and open it in the development system.

Call: Menu bar: *“File”*

Dialog 'Open Project'

“File type”	Type of the CODESYS project to be loaded to the development system
“All supported files”	Filters by all projects which CODESYS can load Hint: For example, you can select `PRO` projects which have been created with CoDeSys V2.3. These kinds of projects are also converted.
File extension `project`	Filters by projects which have been created with CODESYS V3
File extension `projectarchive`	Filters by project archives which have been created with CODESYS V3
File extension `library`	Filters by library projects which have been created with CODESYS V3
“Open”	Loads the project you selected to CODESYS Note: Depending on the state of your CODESYS installation, it may be necessary to update or supplement the installation. If this is the case, then first open another *“Open Project”* dialog with options for installation management.

See also

- *Chapter 1.4.1.2.1 “Opening a V3 Project” on page 186*
- *Chapter 1.4.1.2.2 “Opening a V2.3 project” on page 187*

Command ‘Close Project’

Function: This command closes the currently opened project. CODESYS remains opened.

Call: *“File”* menu. In addition implicitly when opening a new/other project, while another project is still open.

If the project contains unsaved changes, a query appears, asking whether the project should be saved.

If you have not yet explicitly saved the project, a query appears asking whether you wish to delete the project files.

Command 'Save project'

Symbol:  , shortcut *[Ctrl]* + *[S]*

Function: this command saves the project file.

Call: *"File"* menu

This command saves the project file with the current project name, which appears in the title bar of the main window. If the project has been changed since it was last saved, the project name is given an asterisk.

The command is not available if the project is read-only.

Write protection exists if

- the project is identified in the project information (summary) as 'Released'
- the option *"Open read-only"* was selected in the dialog box *"Open Project"* when opening the project

Write protection is indicated by a line in the top right corner of the main window. A mouse-click on this line brings up a menu with commands for the possible actions:

- *"Save project under a different file name on the disk"*: a mouse-click on this option leads to 'Save file as…'
- *"Exit read-only mode"*: appears only if the option *"Open read-only"* was selected when opening the project.
- *"Remove read-only attribute from the project on the disk"*: appears only if the project file had been provided with the property 'Read-only' on the disk at the time of opening.
- *"Remove identification 'Released' in the project information"*: appears only if this attribute is currently set.

Backup copy

Optionally a backup copy of the project file can be created. If the option *"Create backup copy"* is activated in the option dialog box 'Load and Save', the project is additionally copied to a file `<projectname.backup>` each time the project is saved.

See also

- *Chapter 1.4.1.20.3.1.5 "Command 'Save Project as'" on page 958*
- *Chapter 1.4.1.5.8 "Saving the Project" on page 209*
- *Chapter 1.4.1.20.4.13.16 "Dialog 'Options' – 'Load and Save'" on page 1196*

Command 'Save Project as'

This command opens the standard Windows dialog box for saving a file. The project can be stored with the desired location and file type.

"File type"	For both normal projects and library projects, this drop-down list contains the respective versions of the development system for which the project can be saved. If the current project contains add-ons that are not available in the selected memory format (profile), then the *"Extend Profile"* dialog box opens. ● *"Project files (CODESYS v<version>) (*.project)"*: The project is saved as a CODESYS project file "<project name>.project " for the currently used or selected version of the development system. ● *"Library files (CODESYS v<version>) (*.library)"*: The project is saved as a CODESYS library file "<project name>.library" for the currently used or selected version of the development system. If the project should be opened later in an older version, then it makes sense to save for precisely this version, as you will then be informed immediately about possible data loss.

Before saving a project as a library:

- Make sure that the rules for creating libraries have been followed.
- If it is to be possible to configure global constants provided by the library at a later time in an application, then you must define them in a parameter list. A parameter list is a special type of global variable list.
- When saving the project, no automatic check for errors is performed.
- Unlike CoDeSys V2.3, there is no distinction between 'external' and 'internal' libraries. Now you can define in the properties of each individual project object whether or not it should be treated as 'external'.
- Consider whether the library created is to be installed in the system library repository immediately. If so, then use the command 'Save project and install in the library repository'.
- If you want to protect the library project from later changes, then set the *"Released"* attribute in the *"Project Information"* dialog box. At the next attempt to save the project, a corresponding message will be displayed and the user must respond to the write protection with deliberate actions.
- If you save the project as a version of the development system other than the one currently in use, then you will be informed first about possible data loss.

Dialog box 'Extend Profile'

In this dialog box, the selected profile (memory format) can be extended by the add-ons that are contained in the current project. The profile is saved temporarily and then deleted after being saved or exported.

"Add to profile"	☑: The current profile is extended by the add-on so that the add-on data of the current project is also saved.
"Add-on"	The add-on of the current project that is not contained in the selected memory format.
"Version"	Version of the *"Add-on"* included in the current profile. If several versions are installed, then the version can be selected.
"Save profile"	Opens the *"Enter Profile Name"* dialog box. In this dialog box, specify the name for the new profile. The new profile is saved permanently at `$ProgramData$/$PRODUCT$/CustomInformationalProfiles`.
"Use saved profile"	The profile which was permanently saved in *"Save profile"* is used for saving or exporting the current project.

See also

- *Chapter 1.4.1.20.3.1.4 "Command 'Save project'" on page 957*
- *Chapter 1.4.1.5.8 "Saving the Project" on page 209*
- *Chapter 1.4.1.20.4.13.16 "Dialog 'Options' – 'Load and Save'" on page 1196*
- *Chapter 1.4.1.16.1 "Information for Library Developers" on page 449*
- *Chapter 1.4.1.20.3.1.6 "Command 'Save Project and Install into Library Repository'" on page 959*

Command 'Save Project and Install into Library Repository'

Function: this command saves the project as a library in the 'system' library repository.

Call: Main menu *"File"*.

With this command CODESYS saves the project as a library in the 'system' library repository. This is an extension to the saving of a project as a library file using the *"Save Project as"* command. The library is installed on the local system and is immediately available for insertion into a project.

See also

- *Chapter 1.4.1.20.3.1.5 "Command 'Save Project as'" on page 958*

Command 'Save Project as Compiled Library'

Function: The command saves a library project in encrypted form.

Call: Menu bar: *"File"*

The command opens the default dialog for saving a file in the file system. The *"Compiled CODESYS Libraries"* file type is already preset. The file extension is `.compiled-library-v3` or `.compiled-library` (CODESYS < SP15). In this format, then source code of the library POU is not visible when the library is used in a project.

If the *"Enforce signing of compiled libraries"* option is selected in the *"Security Screen"* view on the *"User"* tab, then a library project has to be provided with a digital certificate-based signature when being saved. When a suitable certificate is available, it is provided in the *"Security Screen"* on the *"User"* tab in the *"Digital Signature"* section. In the *"Project Information"*, on the *"Summary"* tab, a *"Library compatibility"* with a CODESYS version >= V3 SP15 is set by default. In this case, the project file is stored with the file extension `.compiled-library-v3` when being saved as a compiled and signed library. If you still have not specified a suitable valid certificate for your user profile in the *"Security Screen"*, then a dialog prompt opens next for you to do this. Afterwards, you can execute the save command again.

In all other respects, compiled library files behave just like `*.library` files, and therefore they can be installed and referenced with the same steps.

We recommend the use of compiled libraries signed with certificates. Besides the protection of the source code and the unauthorized use of a library, less memory is also used which therefore results in shorter loading times.

If you have the corresponding help files with translations, then as of CODESYS V3 SP15 you can extend the library documentation with the translation into other languages. This is done as follows:

Place the files created for the new languages `__lmd__<language>.aux` in a directory `<library name>.lmd` parallel to the library project `<library name>.compiled-library-v3`. If the files are correct, then they are included in the compiled library file when saving the library project by means of the "Save Project as Compiled Library" command.

Example: The directory `standard.lmd` is exists parallel to the library file `standard.compiled-library-v3` and contains the file `__lmd__fr.aux` with the French translation of the library documentation. After the compiled library is saved, the French version of the documentation is also available in the Library Manager.

See also

- *"Tab 'Summary'" on page 919*
- *Chapter 1.4.1.20.3.3.18 "Command 'Security Screen'" on page 995*
- *Chapter 1.4.1.16.1 "Information for Library Developers" on page 449*
- *Chapter 1.4.1.20.3.1.6 "Command 'Save Project and Install into Library Repository'" on page 959*

Command 'Save/Send Archive'

Function: This command opens the dialog *"Project Archive"* for the configuration of project archives.

Call: Menu bar: *"File ➔ Project Archive"*

An archive file (*.projectarchive) contains all files contained and referenced in the currently opened project. It can either be saved or dispatched as an e-mail attachment. The dispatch by email is very helpful for providing an employee with all project-relevant files. The file can be simply unpacked again with the command *"Extract Archive"*.

NOTICE!
The archiving function is not intended for the storage of a project, but rather for the simple summarizing of all project-relevant files.

See also

- *Chapter 1.4.1.5.9 “Saving/Sending the project archive” on page 210*
- *Chapter 1.4.1.20.3.1.9 “Command 'Extract Archive'” on page 961*

Dialog 'Project Archive'

The dialog displays all the categories that can be added to the project archive. In this dialog, complete categories or individual objects from the categories can be added to the project archive by setting a check mark (☑).

Entries that are display as red in the list require your attention. Move the mouse pointer over this library for more information.

“Additional Files”	Opens the dialog *“Additional Files”*. Here, further files can be added to the archive with the *“Add”* button.
“Comment”	Opens the "Comment" dialog. Here, comments can be added to the archive.
“Save”	Creates the archive file and saves it. The storage location and the archive name are specified in the subsequent dialog
“Send”	Creates a temporary archive file that is attached to an empty e-mail. A correct installation of the MAPI (Messaging Application Programming Interface) is required for the successful execution of this operation. Failure is documented by the display of a corresponding error message. The temporary archive is automatically deleted after sending the e-mail.

Command 'Extract Archive'

Function: The command extracts a project archive, that was created with the command *“Save/Send Archive”*. You have to configure which objects of the archive CODESYS shall extract and in which directory of the file system they will be copied.

Call: Main menu *“File ➔ Project Archive”*

The file extension of an archive is `.projectarchive`.

After the archive is selected, the dialog *“Extract Project Archive”* opens to configure the extract parameters.

Dialog Box 'Extract Project Archive'

This dialog box shows the contents of the project archive. You can exclude complete categories or single objects from categories by clearing the check boxes (☑) from the extraction.

Table 110: “Locations”

“Extract into the same folder where the archive is located”	The archive is extracted to the same directory.
“Extract into the following folder”	The contents of the archive are extracted to the given path.
“Advanced”	Opens the *“Advanced”* dialog box for you to define where special and additional files from the archive are extracted.

Table 111: “Contents”

“Items”	Shows the contents of the archive structured in object categories. ☑: The object is extracted. ☐: The object is not extracted.
“Comment”	Comment that was entered when creating the project archive

“Extract”	If an extracted file has the same name as an existing file in the target directory, then a dialog box opens, prompting whether the local file should be replaced. The decision can be applied automatically to any additional conflicting names. In this case, you have to select the ☑ *“Apply to all objects and files”* check box.

Dialog 'Advanced'

Table 112: “Repositories”

“Install devices into”	Drop-down list with currently available repositories. Select the repositories, in which CODESYS shall install the devices and the libraries of the archive.
“Install libraries into”	

Table 113: “Additional Files”

By default the "additional files" are set to *“Do not extract”*. Select the entries in the table and chose one of the following options:	
“Extract into project folder”	Folder of the project file
“Extract into folder”	User defined folder
“Do not extract”	Default

See also

- *Chapter 1.4.1.20.3.1.8 “Command 'Save/Send Archive'” on page 960*

Command 'Source Upload'

Function: This command loads the project source code (as project archive) from the controller.

Call: Main menu *“File”*.

Requirement: The network path for the controller must be configured.

After you execute the command, an overview opens with all devices in the network. Select a controller from this overview. The dialog box *“Extract Project Archive”* then opens with export settings.

See also

- *Chapter 1.4.1.10.7 “Downloading source code to and from the PLC” on page 393*
- *Chapter 1.4.1.20.3.1.11 “Command 'Source Download'” on page 963*

Dialog Box 'Extract Project Archive'

This dialog box shows the contents of the project archive. You can exclude complete categories or single objects from categories by clearing the check boxes (☑) from the extraction.

Table 114: “Locations”

“Extract into the same folder where the archive is located”	The archive is extracted to the same directory.
“Extract into the following folder”	The contents of the archive are extracted to the given path.
“Advanced”	Opens the *“Advanced”* dialog box for you to define where special and additional files from the archive are extracted.

Table 115: “Contents”

“Items”	Shows the contents of the archive structured in object categories. ☑: The object is extracted. ☐: The object is not extracted.
“Comment”	Comment that was entered when creating the project archive

“Extract”	If an extracted file has the same name as an existing file in the target directory, then a dialog box opens, prompting whether the local file should be replaced. The decision can be applied automatically to any additional conflicting names. In this case, you have to select the ☑ *“Apply to all objects and files”* check box.

Command 'Source Download'

Function: This command loads the project source code (as project archive) to the controller.

Call: Main menu *“File”*.

Requirement: The network path for the controller must be configured.

After you execute the command, an overview opens with all devices in the network. Select a controller from this overview. Then the `Archiv.prj` project archive is downloaded to this controller. You can click *“Source Upload”* to upload the complete source code to the CODESYS development system at a later time.

If you are already connected to a controller (in online mode), then the *“Source Download to Connected Device”* command is also available for this process.

See also

- *Chapter 1.4.1.10.7 “Downloading source code to and from the PLC” on page 393*
- *Chapter 1.4.1.20.3.1.10 “Command 'Source Upload'” on page 962*
- *Chapter 1.4.1.20.3.6.7 “Command 'Source Download to Connected Device'” on page 1035*

Command 'Print'

Symbol: 

Function: This command opens the default Windows dialog box for printing documents.

Call: Main menu *“File”*

See also

- *Chapter 1.4.1.20.4.11.6 “Dialog 'Project Settings' - 'Page Setup'” on page 1175*

Command 'Print Preview'

Function: This command opens a print preview for the currently open element.

Call: Main menu *“File”*

Requirement: An object is open in the editor.

See also

- *Chapter 1.4.1.20.4.11.6 “Dialog 'Project Settings' - 'Page Setup'” on page 1175*
- *Chapter 1.4.1.20.3.1.12 “Command 'Print'” on page 963*

Command 'Page Setup'

Symbol: 

Function: This command opens the *“Page Setup”* dialog box for configuring the layout of the printed version of the project contents.

Call: Main menu *“File ➔ Page Setup”*

See also

- *Chapter 1.4.1.20.4.11.6 “Dialog 'Project Settings' - 'Page Setup'” on page 1175*
- *Chapter 1.4.1.20.3.1.12 “Command 'Print'” on page 963*

Command ‘Recent Projects’

Function: Opens the list of the projects used recently, from which you can select a project to open.

Call: *“File”* menu

Command 'Exit'

Shortcut: *[<Alt>]+[<F4>]*

Function: this command exits from the programming system. If a project is currently opened that has been changed since it was last saved, a dialog box opens asking whether the project should be saved.

Call: *“File”* menu

Menu 'Edit'

Standard Commands

CODESYS provides the following standard commands:

- Undo: , shortcut: *[Ctrl]* + *[Z]*
- Redo: , shortcut: *[Ctrl]* + *[Y]*
- Cut: , shortcut: *[Ctrl]* + *[X]*
- Copy: , shortcut: *[Ctrl]* + *[C]*
- Paste: , shortcut: *[Ctrl]* + *[V]*
- Delete: , shortcut: *[Ctrl]*
- Select all: shortcut: *[Ctrl]* + *[Ctrl]*

Not all editors support the *"Insert"* command. In some editors it can be used with limitations. Graphical editors only support the command if the pasted elements will create a valid construct.

In object trees like POUs or device view the command refers to the currently selected object. Multi selection is possible.

Command 'Find', 'Find in Project'

Symbol ; keyboard shortcut: *[Ctrl]+[F]*

Symbol , keyboard shortcut *[Ctrl]+[Shift]+[F]*

Function: These commands scan the project or parts of it for a specified character string.

Call: Menu bar: *"Edit* ➔ *Find Replace"*

This command opens the *"Find"* dialog where the searched character string is specified and the search options are defined.

Dialog 'Find'

"Search for"	Character string to be searched.
"Match case":	: The search considers uppercase and lowercase.
"Match whole word":	: Only character strings are found that exact matches.
"Search up":	: The specified search range runs upwards. : The specified search range runs downwards.
"Use regular expressions":	Use the > button to receive support when specifying regular expressions.
"Search in"	: Drop-down list with the areas of the project to be searched: ● *"Active editor"* ● *"All open editors"* ● *"Selected objects & Subobjects"* ● *"Entire project"* ● *"Entire project & Uncompiled libraries"* ● *"Selection only"* ...: Opens a dialog where you set the areas of the project to be searched (see below)
"Find next"	Start the search
"Find all"	All search results are listed in the message view with their object path, project name, object name, and object position. Possible additional information for position: *"(Decl)"* = Declaration part of the object; *"(Impl)"* = Implementation part of the object Double-clicking the entry in the list opens the match position in the respective object editor.
"Replace"	Switches to the *"Replace"* dialog

The color of the search result markings can be customized in the options of the text editor. This is done by means of the parameter "Selection color" - "Inactive" in the "Text Area" tab.

See also

- *"Tab 'Text Area'" on page 1204*

Dialog for setting the objects to be searched

"Entire project"	All editable positions in all objects of the project are searched.
"Entire project and all uncompiled libraries"	All editable positions in all objects of the project, including integrated uncompiled libraries, are searched.
"Within the following objects"	Only the editable positions within the objects defined here are searched:
	• *"Scheme"*: The *"Save"* command saves the current search configuration by the specified name. All saved schemes are available in the drop-down list ( ). • *"Object types"*: : The object is searched. • *"Name filter"*: Name filter for the searched objects. The placeholder "*" can be used. Example: Filter "*CAN*": All objects are searched that have "CAN" in the name.
"All open editors"	All editors are searched that are currently open in a window.
"Active editor"	Only the editor is searched where the cursor currently is.
"Selection only"	Only the text is searched that is currently selected in an object.

See also

- *Chapter 1.4.1.20.3.2.3 "Command 'Replace', 'Replace in Project'" on page 967*
- *Chapter 1.4.1.8.14 "Searching and replacing in the entire project" on page 289*

Command 'Replace', 'Replace in Project'

Symbol ; keyboard shortcut: *[Ctrl]+[H]*

Symbol , keyboard shortcut *[Ctrl]+[Shift]+[H]*

Function: These commands scan the project or parts of it for a specified character string and replaces it.

Call: Menu bar: *"Edit ➔ Find Replace"*

Requirement: The application is in online mode.

This command opens the *"Replace"* dialog where the search and replace character strings are specified and the search options are defined.

Table 116: In addition to the options of the "Search" dialog, the following settings are still possible:

"Replace with"	Input field for the new character string.
"Replace"	Each next found string is highlighted in the editor and replaced (step-by-step replace).

“Replace all”	All found strings are replaced at one time without them being displayed in the editors.
“Leave changed objects open after "Replace all"”	The editors of the found objects remain open.

Replacement in referenced libraries is not possible.

See also

- *Chapter 1.4.1.20.3.2.2 “Command 'Find', 'Find in Project'” on page 966*
- *Chapter 1.4.1.8.14 “Searching and replacing in the entire project” on page 289*

Command 'Find Next'

Symbol  ,keyboard shortcut *[F3]*

Function: During the search for a certain string within the project, this command selects the next match at its position in the respective editor.

Call: Menu *“Edit ➔ Search Replace”*

Requirement: You have already started searching the project for a certain string by using the commands *“Find”* or *“Replace”*.

See also

- *Chapter 1.4.1.20.3.2.2 “Command 'Find', 'Find in Project'” on page 966*
- *Chapter 1.4.1.20.3.2.3 “Command 'Replace', 'Replace in Project'” on page 967*
- *Chapter 1.4.1.8.14 “Searching and replacing in the entire project” on page 289*

Command 'Find Next (Selection)'

Keyboard shortcut *[Ctrl]* + *[F3]*

Function: The command searches the project for the next string matching the string which is currently selected or in which you have currently placed the cursor.

Call: Menu *“Edit ➔ Find Replace”*

Requirement: You have the cursor placed in an editable string in your project, or you have selected an editable string.

See also

- *Chapter 1.4.1.20.3.2.2 “Command 'Find', 'Find in Project'” on page 966*
- *Chapter 1.4.1.20.3.2.3 “Command 'Replace', 'Replace in Project'” on page 967*
- *Chapter 1.4.1.8.14 “Searching and replacing in the entire project” on page 289*

Command 'Find Previous'

Symbol  , keyboard shortcut *[Shift]* + *[F3]*

Function: During the search for a certain string within the project, this command selects the next match at its position in the respective editor.

Call: Menu *“Edit ➔ Search Replace”*

Requirement: You have already started searching the project for a certain string by using the commands *“Find”* or *“Replace”*.

See also

- *Chapter 1.4.1.20.3.2.2 “Command 'Find', 'Find in Project'” on page 966*
- *Chapter 1.4.1.20.3.2.3 “Command 'Replace', 'Replace in Project'” on page 967*
- *Chapter 1.4.1.8.14 “Searching and replacing in the entire project” on page 289*

Command 'Find Previous (Selection)'

Keyboard shortcut *[Ctrl]* + *[Shift]* + *[F3]*

Function:The command searches the project for the previous string matching the string which is currently selected or in which you have currently placed the cursor.

Call: Menu *“Edit ➔ Find Replace”*

Requirement: You have the cursor placed in an editable string in your project, or you have selected an editable string.

See also

- *Chapter 1.4.1.20.3.2.2 “Command 'Find', 'Find in Project'” on page 966*
- *Chapter 1.4.1.20.3.2.3 “Command 'Replace', 'Replace in Project'” on page 967*
- *Chapter 1.4.1.8.14 “Searching and replacing in the entire project” on page 289*

Command 'Insert File as Text'

Function: This command copies the contents of a text file to the active editor as the current cursor position.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog box from *“Tools ➔ Customize”* (command category *“Text Editor”*).

Requirement: The file must have the extension `.txt`. The command is available in a text editor only.

Many development environments and text processing applications provide the option of exporting code and text as a plain text file. This command can copy the contents of this file to the editor.

See also

- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Overwrite Mode'

Shortcut: *[Insert]*

Function: This command activates the overwrite mode.

Call: Menu *“Edit ➔ Advanced”*

Requirement: A text editor is opened.

If the overwrite mode is activated, characters in front of the cursor are overwritten when entering new characters. If the overwrite mode is deactivated, characters are inserted and existing characters in front of the cursor are retained.

See also

- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'View Whitespace'

Symbol: 

Function: This command causes control characters for spaces and tabs to be shown.

Call: Menu *“Edit ➔ Advanced”*

Requirement: A text editor is opened.

CODESYS visualizes spaces by a period and tabs by an arrow.

See also

- ↳ *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'View Indentation Guides'

Function: This command activates the indentation help lines.

Call: Menu *“Edit ➔ Extended”*

Requirement: A text editor is opened.

If the indentation help lines are activated, a broken line is inserted for each manual indentation in the code. This facilitates the overview of the different levels in the code. You can insert manual indentations with the *[Tab]* key.

See also

- ↳ *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Go to Line'

Function: With this command the cursor jumps to a defined line in the code.

Call: Menu *“Edit ➔ Extended”*

Requirement: A text editor is opened.

This command opens a dialog box with an input field *“Line number”*.

See also

- ↳ *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Make Uppercase'

Shortcut: *[Ctrl]+[Shift]+[U]*

Function: This command converts all lowercase letters in the selected code into uppercase letters.

Call: Menu *“Edit ➔ Advanced”*

Requirement: A text editor is opened and code is selected, or the declaration editor is opened and variable declarations are selected.

See also

- ↳ *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Make Lowercase'

Shortcut: *[Ctrl]+[U]*

Function: This command converts all uppercase letters in the selected code into lowercase letters.

Call: Menu *“Edit ➔ Advanced”*

Requirement: a text editor is opened and code is selected, or the declaration editor is opened and variable declarations are selected.

See also

- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Go to Matching Bracket'

Function: This command makes the cursor jump to the other part of the selected code parenthesis.

Call: Menu *“Edit ➔ Advanced”*

Requirement: A text editor is opened and the cursor is positioned at an opening or closing code parenthesis. If you position the cursor at a code parenthesis, CODESYS displays the corresponding parenthesis in color, provided you have activated the option *“Associated parentheses”* in the CODESYS options in the *“Text Editor”* category, *“Text Area”* tab.

See also

- *“Tab 'Text Area'” on page 1204*

Command 'Select to Matching Bracket'

Function: This command selects the entire code section within the currently selected code parentheses.

Call: Menu *“Edit ➔ Extended”*

Requirement: A text editor is opened and the cursor is positioned at an opening or closing code parenthesis. If you position the cursor at a code parenthesis, CODESYS displays the corresponding parenthesis in color, provided you have activated the option *“Associated parentheses”* in the project options in the *“Text Editor”* category, *“Text Area”* tab.

See also

- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Expand All Folds'

Function: This command expands all collapsed code segments in the textual editor or result locations in the cross-reference list so that the code and all search locations are displayed in full again.

Requirement: A textual editor is active and indentation is activated in the *“Options”* (*“Text Editor”* category); or the cross-reference list is active.

Call: Textual editors: main menu*“Edit ➔ Advanced”*, or right-click. In the cross-reference list: right-click.

See also

- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*
- *Chapter 1.4.1.20.3.2.18 “Command 'Collapse All Folds'” on page 971*
- *“Right-click commands in the cross-reference list” on page 992*

Command 'Collapse All Folds'

Function: This command collapses all expanded code segments in the textual editor or result locations in the cross-reference list. In this way, only the uppermost level of code and only the root node of the result locations displayed.

Requirement: A textual editor is active and indentation is activated in the *"Options"* (*"Text Editor"* category); or the cross-reference list is active.

Call: In textual editors: main menu *"Edit ➔ Advanced"*, or right-click. In the cross-reference list: right-click.

See also

- *Chapter 1.4.1.20.4.13.25 "Dialog 'Options' - 'Text Editor'" on page 1203*
- *Chapter 1.4.1.20.3.2.17 "Command 'Expand All Folds'" on page 971*
- *"Right-click commands in the cross-reference list" on page 992*

Command 'Comment Out Selected Lines'

Symbol ; keyboard shortcut: *[Ctrl]+[O]*

Function: The command inserts comment marks ('//') at the beginning of the selected lines.

Call: Menu bar: *"Edit ➔ Advanced"*; context menu

Requirement: In the ST editor, either the cursor is located in a line of the implementation or multiple lines are selected.

See also

- *Chapter 1.4.1.20.3.2.20 "Command 'Uncomment Selected Lines'" on page 972*

Command 'Uncomment Selected Lines'

Symbol ; keyboard shortcut: *[Ctrl]+[I]*

Function: The command removes any comment marks ('//') at the beginning of the selected lines.

Call: Menu bar: *"Edit ➔ Advanced"*; context menu

Requirement: In the ST editor, either the cursor is located in a line of the implementation or multiple lines are selected.

See also

- *Chapter 1.4.1.20.3.2.19 "Command 'Comment Out Selected Lines'" on page 972*

Command 'Enable Inline Monitoring'

Function: This command enables or disables the inline monitoring function. This works the same way as the check box with the same name in the CODESYS options (*"Text Editor"* category).

Requirement: A text editor is active.

Call: Context menu of the text editor in the *"Advanced"* submenu.

See also

- *"Tab 'Monitoring'" on page 1205*
- *Chapter 1.4.1.12.1 "Monitoring of Values" on page 409*

Command 'Toggle Bookmark'

Symbol  , keyboard shortcut *[Ctrl]+[F12]*

Function: The command sets or removes a bookmark at the current position.

Call: Menu bar: *"Edit ➔ Bookmarks"*

Requirement: A POU is open in the editor and the cursor is at a program line.

See also

- *Chapter 1.4.1.8.13.3 “Setting and using bookmarks” on page 287*

Command 'Next Bookmark (Active Editor)'

Symbol: ; keyboard shortcut: *[F12]*

Function: The command jumps to the next bookmark in the active editor.

Call: Menu bar: *“Edit ➔ Bookmarks”*

Requirement: A POU is open in the editor and the cursor is positioned in the POU.

See also

- *Chapter 1.4.1.20.3.2.24 “Command 'Next Bookmark'” on page 973*
- *Chapter 1.4.1.8.13.3 “Setting and using bookmarks” on page 287*

Command 'Next Bookmark'

Symbol:

Function: The command jumps to the next bookmark in the *“Bookmarks”* view and in the project, and opens the respective POU. The order of jumping to bookmarks corresponds to the order of bookmarks in the table of the *“Bookmarks”* view.

Call:

- *“Next Bookmark”* button in the *“Bookmarks”* view
- The command is not in any menu by default. You can add it to a menu by using the dialog from *“Tools ➔ Customize”* (command category *“Bookmarks”*).

Requirement:

- A project is open.
- The *“Bookmarks”* view is open.

See also

- *Chapter 1.4.1.20.3.2.23 “Command 'Next Bookmark (Active Editor)'” on page 973*

Command 'Previous Bookmark (Active Editor)'

Symbol: ; keyboard shortcut: *[Shift]+[F12]*

Function: The command jumps to the previous bookmark in the active editor.

Call: Menu bar: *“Edit ➔ Bookmarks”*

A POU is open in the editor and the cursor is positioned in the POU.

See also

- *Chapter 1.4.1.20.3.2.26 “Command 'Previous Bookmark'” on page 973*
- *Chapter 1.4.1.8.13.3 “Setting and using bookmarks” on page 287*

Command 'Previous Bookmark'

Symbol:

Function: The command jumps to the previous bookmark in the *“Bookmarks”* view and in the project, and opens the respective POU. The order of jumping to bookmarks corresponds to the order of bookmarks in the table of the *“Bookmarks”* view.

Call:

- *"Next Bookmark"* button in the *"Bookmarks"* view
- The command is not in any menu by default. You can add it to a menu by using the dialog from *"Tools ➔ Customize"* (command category *"Bookmarks"*).

Requirement:

- A project is open.
- The *"Bookmarks"* view is open.

See also

- *Chapter 1.4.1.20.3.2.25 "Command 'Previous Bookmark (Active Editor)'" on page 973*
- *Chapter 1.4.1.8.13.3 "Setting and using bookmarks" on page 287*

Command 'Clear All Bookmarks (Active Editor)'

Symbol:

Function: The command deletes all bookmarks in the active editor.

Call: Menu bar: *"Bookmarks"*

Requirement: A POU is open in the editor and the cursor is positioned in the POU.

See also

- *Chapter 1.4.1.20.3.2.28 "Command 'Clear All Bookmarks'" on page 974*
- *Chapter 1.4.1.8.13.3 "Setting and using bookmarks" on page 287*

Command 'Clear All Bookmarks'

Symbol:

Function: The command deletes all bookmarks in the open project.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog from *"Tools ➔ Customize"* (command category *"Bookmarks"*).

Requirement: A POU is open in the editor and the cursor is positioned in the POU.

See also

- *Chapter 1.4.1.20.3.2.27 "Command 'Clear All Bookmarks (Active Editor)'" on page 974*
- *Chapter 1.4.1.8.13.3 "Setting and using bookmarks" on page 287*

Command 'Browse Cross References'

Symbol:

Function: The command shows all occurrences of a variable in the *"Cross Reference List"* view.

Call: Menu bar: *"Edit ➔ Browse"*; cross reference view: toolbar

Requirement: A POU is open in the editor and the cursor is set at a variable. Or the *"Cross Reference List"* view is open and a variable is specified in the *"Name"* field.

See also

- *Chapter 1.4.1.8.13.1 "Using the cross-reference list to find occurrences" on page 285*
- *Chapter 1.4.1.20.3.2.30 "Command 'Browse Global Cross References'" on page 975*

Command 'Browse Global Cross References'

Symbol: 

Function: The command shows all occurrences of all variables with the same name in the *"Cross Reference List"* view. In contrast to the *"Browse Cross References"* command, these can be different variables.

Call: Menu bar: *"Edit ➔ Browse"*; cross reference view: toolbar

Requirement: A POU is open in the editor and the cursor is set at a variable. Or the *"Cross Reference List"* view is open and a variable is specified in the *"Name"* field.

See also

- *Chapter 1.4.1.8.13.1 "Using the cross-reference list to find occurrences" on page 285*
- *Chapter 1.4.1.20.3.2.29 "Command 'Browse Cross References'" on page 974*

Command 'Browse Call Tree'

Symbol: 

Function: The command opens the view *"Call Tree"*, which displays the calls of a module and also its callers.

Call:

- Menu *"Edit ➔ Browse"*
- Context menu, see below: Requirement

Requirement: A module is opened in the editor and the cursor is placed in a variable, or a module is selected in the *"Devices"* view or in the *"POUs"* view.

See also

- *Chapter 1.4.1.20.3.3.16 "Command 'Call tree'" on page 993*

Command 'Auto Declare'

Keyboard shortcut: *[Shift]+[F2]*

Function: The command opens the *"Auto Declare"* dialog, which supports the declaration of a variable.

Call: Menu bar: *"Edit"*

Requirement: An object or a device of the project is opened in the editor.

With the auto-declaration function, the *"Auto Declare"* dialog also appears when the cursor is located in the implementation part of a POU in a line containing the name of an undeclared variable. The requirement for this is that you must have clicked *"Tools ➔ Options"* and enabled the *"Declare unknown variables automatically (AutoDeclare)"* option in the *"SmartCoding"* category.

With the smart tag function, the *"Auto Declare"* command also appears when you place the cursor over an undeclared variable in the implementation part of the ST editor and then click  .

Dialog 'Auto Declare'

"Scope"	Scope of the variable that is not declared yet. Example: `VAR` (default setting for local variables)
"Name"	Variable name not declared yet Example: `bIsValid`

"Type"	Example: `BOOL` ● ▼: Lists the standard data types. ● >: – *"Input Assistant"*: Opens the *"Input Assistant"* dialog – *"Array Assistant"*: Opens the *"Array"* dialog
"Object"	Object where the new variable is declared. By default, the object that you are editing now. Example: `fbA` ▼: Lists that objects where the variable can be declared. If no objects are available for the selected *"Scope"*, the entry *"<create object>"* appears. When you select the *"<create object>"* entry, the *"Add Object"* dialog opens for generating a suitable object.
"Initialization"	Example: `FALSE` If you do not specify an initialization value, then the variable is initialized automatically. ...: Opens the *"Initialization Value"* dialog. This procedure is helpful for the initialization of structured variables.
"Address"	Memory address of the application for the variable that is not declared yet. Example: `%IX1.0` Note: Possible only for the following scopes: ● Local variable (`VAR`) ● Global variable (`VAR_GLOBAL`) ● Or for a persistent variable (`PERSISTENT`).
"Flags"	Attribute keywords ● `CONSTANT`: Keyword for a constant. ● `RETAIN`: Keyword for a remanent variable. ● `PERSISTENT`: Keyword for a persistent variable (stricter than `RETAIN`). The selected attribute keyword is added to the variable declaration.
"Comment"	Example: `New input In1` In the tabular declaration editor, the comment entered is displayed in the *"Comment"* column, while in the textual declaration editor it is displayed above the variable declaration.
"Apply changes using refactoring"	☑: When you exit the dialog, the variable is not declared yet, but then it opens the *"Refactoring"* dialog. You can continue editing your changes here. The option appears for the following scopes: ● Input variable (`VAR_INPUT`) ● Output variable (`VAR_OUTPUT`) ● `VAR_IN_OUT` variables (input variable and output variable)

"OK"	The variable is declared and appears in the declaration.
	Example:
	`VAR RETAIN`
	`    // New input In1`
	`    xIn1 AT %IX1.0: BOOL := FALSE;`
	`END_VAR`

Dialog 'Array'

"Ranks and base type specification"	Definition of the field sizes (*"Dimension"*) by entering the lower and upper limits and the *"Base type"* of the array. You can enter the basic type directly or with the help of the *"Input Assistant"* or *"Array"* dialogs when you click the > button.
"Result"	Display of the defined array

NOTICE!
CODESYS reinitializes variables only if you have modified the initialization values of the variables.

Dialog 'Initialization Value'

List of the variables with name (*"Expression"*), *"Initialization Value"* and *"Data Type"*. Modified initialization values are displayed in bold fonts.	
Input field below the list	Input of an initialization value for the selected variable(s)
"Apply value to selected lines"	Change of the initialization value of the selected line(s) according to the value of the input field
"Reset selected lines to default values"	Resets the default initialization values
"OK"	CODESYS applies the initialization values in the *"Auto Declare"* dialog.

In the case that the variable to be initialized by means of this dialog is a function block instance with an extended FB_Init method, an additional table is displayed above the *"Initialization Value"* table. The additional FB_Init parameters are listed in this table. The meaning and operation essentially correspond to the lower table with the following differences:

- All variables have to be assigned with initialization values. Otherwise *"OK"* remains disabled.
- For complex data types (structures, arrays), no components contained within are displayed (type cannot be expanded). In this case, the complex type has to be initialized with a corresponding variable.

For FB_Init parameters configure this way, a corresponding symbol is displayed after the initialization value in the *"Auto Declare"* dialog.

See also

- *Chapter 1.4.1.19.10 "Methods 'FB_Init', 'FB_Reinit', and 'FB_Exit'" on page 748*
- *Chapter 1.4.1.8.2.2 "Using the 'Declare variable' dialog box" on page 227*
- *"Smart tag functions" on page 263*
- *Chapter 1.4.1.8.11.2 "AT declaration" on page 281*
- *Chapter 1.4.1.8.15 "Refactoring" on page 289*
- *Chapter 1.4.1.8.19 "Data Persistence" on page 301*
- *"Dialog box 'Refactoring'" on page 982*
- *Chapter 1.4.1.19.4.10 "Addresses" on page 643*
- *Chapter 1.4.1.20.4.13.21 "Dialog 'Options' - 'Refactoring'" on page 1199*
- *Chapter 1.4.1.19.1.3.1 "ST Editor" on page 463*

Command 'Input Assistant'

Symbol: ; keyboard shortcut: *[F2]*

Function: This command opens the *“Input Assistant”* dialog which helps you to select one of the possible programming elements at the current cursor position.

Call: Menu bar: *“Edit”*; context menu.

Requirement: A POU is open in the editor and the cursor is at a program line.

Dialog 'Input Assistant' - Tab 'Categories'

The input assistant provides all program elements that you can insert at the current cursor position in the editor.

The elements are sorted by *“Categories”*. In the category *“Variables”*, you can also set a *“Filter”* for the scope, for example *“Local variables”*, *“Global variables”*, or *“Constants”*.

“Structured view”	☑: The elements are displayed in a structure tree. You can show/hide the columns *“Type”*, *“Address”*, and *“Origin”* by right-clicking the column title and selecting/clearing the column name in the dropdown list. ☐: The elements are displayed in a flat structure.
“Show documentation”	☑: The dialog is extended with the *“Documentation”* field.
“Insert with arguments”	☑: Elements that include arguments (for example, functions) also insert with these arguments at the cursor position. Example: If you insert the function block `fb1`, which contains an input variable `fb1_in` and an output variable `fb1_out`, "with arguments", then this appears in the editor as follows: `fb1(fb1_in:= , fb1_out=> ).`
“Insert with namespace prefix”	☑: Inserts the selected element with the appended namespace. In the case of library modules, the check box remains disabled if the requirement for a namespace has been defined in the library properties.

If you create objects with the same name in the same category, whether globally (“POUs” view) or assigned to an application (“Devices” view), then only one entry appears in the input assistant. The usage conforms to the usual call priority (application assigned before global).

Dialog 'Input Assistant' - Tab 'Text Search'

This tab allows you to search for specific objects. When you begin typing a search string into the search field, the names of all objects are listed whose names include the search string. Double-click an object to insert it at the current cursor position in the editor.

“Filters”	Limits the search to a specific variable category

See also

- *Chapter 1.4.1.8.5 “Using input assistance” on page 260*
- *“Dialog 'Properties'” on page 1118*
- *Chapter 1.4.1.20.3.2.33 “Command 'Input Assistant'” on page 977*

Command 'Go to Source Position'

Function: The command sets the cursor to the position in the source code that causes the message.

Call: Main menu *“Edit”*, context menu of the message in the message view.

Requirements: A message is selected in the message view.

Use the command *“Next Message”* or *“Previous Message”* to display the source code position of the next or previous message.

See also

- *Chapter 1.4.1.20.3.2.35 “Command 'Next Message'” on page 979*
- *Chapter 1.4.1.20.3.2.36 “Command 'Previous Message'” on page 979*
- *Chapter 1.4.1.20.3.3.5 “Command 'Messages'” on page 986*

Command 'Next Message'

Keyboard shortcut: *[F4]*

Function: This command selects the next message in the messages view.

Call: Main menu *“Edit”*.

If the last message in the list has been reached, then the marking jumps to the beginning.

See also

- *Chapter 1.4.1.20.3.2.36 “Command 'Previous Message'” on page 979*

Command 'Previous Message'

Keyboard shortcut: *[Shift]+[F4]*

Function: This command selects the previous message in the messages view.

Call: Main menu *“Edit”*

If the first message in the list has been reached, then the marking jumps to the end.

See also

- *Chapter 1.4.1.20.3.2.35 “Command 'Next Message'” on page 979*

Command 'Go to Definition'

Symbol: 

Function: This command shows the definition locations of a variable or function.

Call: Main menu *“Edit ➔ Browse”*

Requirement: A POU is open in the editor and the cursor is at a variable or function.

See also

- *Chapter 1.4.1.8.13.2 “Finding declarations” on page 287*

Command 'Go To Reference'

Symbol: 

Function: The command opens the declaration location of the variable that is referenced by the pointer currently in focus in online mode.

Call:

- Context menu in the declaration part or implementation code
- Menu bar: *“Edit ➔ Browse”*

Requirement: Online mode. A POU is open in the editor and the cursor is at a pointer. The referenced variable is stored in static memory.

If the pointer does not point exactly to the beginning of the variable, then a corresponding message is displayed when you switch to the variable declaration.

See also

- *Chapter 1.4.1.19.5.12 “Pointers” on page 656*

Command 'Go to Instance'

Symbol:

Function: This command opens the instance of a function block in a new window.

Call: Menu bar: *“Edit ➔ Browse for Symbol”*

Requirement: The application is in online mode. A POU is open in the editor and the cursor is at an instance of a function block.

The command is not available for temporary instances or instances from compiled libraries.

See also

- *Chapter 1.4.1.8.13.2 “Finding declarations” on page 287*

Command 'Refactoring' - 'Rename <...>'

Function: This command opens a dialog box for renaming an object or variable across the project.

Call: Main menu *“Edit ➔ Refactoring”* or right-click.

Requirement: An object is selected in the device tree or in the *“POUs”* view, or the cursor is placed before or on a variable identifier in the declaration section of a programming object.

You can rename the following:

- Variables
- POUs
- GVLs
- Methods
- Properties
- Devices
- Variables and unit conversions in the unit conversion edit

Dialog box 'Rename'

“Current name”	Name of the object or variable
“New name”	Input field for a new name. If the name already exists, then CODESYS reports this directly below this input field.
“OK”	Can be activated if you have typed a valid name in *“New name”*. Opens the *“Refactoring”* dialog box. The affected objects and occurrences are highlighted in both views. You can determine how to handle the occurrences in each view by right-clicking the occurrences and clicking the available commands.

Dialog box 'Re-factoring'

This dialog box displays all occurrences in the project. The affected objects and occurrences are highlighted in both views.	
Right view	Displays the occurrence within an object where *"Current name"* occurs.
Left view	Device tree of the project with the object.
You can determine how to handle the occurrences in each view by right-clicking the occurrences and clicking the available commands.	
"Reject this change"	Reject the single change in view on the right.
"Accept this object"	Accept all changes in the affected object.
"Reject this object"	Reject all changes in the affected object.
"Accept whole project"	Accept all changes in the project.
"Reject whole project"	Reject all changes in the project.
CODESYS highlights the accepted changes in yellow and the rejected changes in gray.	

See also

- *Chapter 1.4.1.8.15 "Refactoring" on page 289*

Command 'Refactoring' - 'Update Referenced Pins'

> **NOTICE!**
> Currently, this command applies only to the CFC, FBD, LD, and IL editors. It is a combination of the *"Reset Pins"* and *"Update Parameters"* commands.

Function: This command modifies the pins according to the latest block declaration in all affected occurrences of the block.

Call: Main menu *"Edit ➔ Refactoring"* or right-click.

Requirement: The cursor is placed in the name of the block in the first line of the block declaration or in the device tree.

See also

- *Chapter 1.4.1.8.15 "Refactoring" on page 289*
- *Chapter 1.4.1.20.3.12.24 "Command 'Reset Pins'" on page 1098*
- *Chapter 1.4.1.20.3.13.38 "Command 'Update Parameters'" on page 1114*

Command 'Refactoring' - 'Add Variable'

Symbol: 

Function: This command enables the declaration of variables in a POU, as well as the automatic update to the occurrence of the POU.

Call: Main menu *"Edit ➔ Refactoring"*, or right-click.

Requirements: The declaration part is in focus.

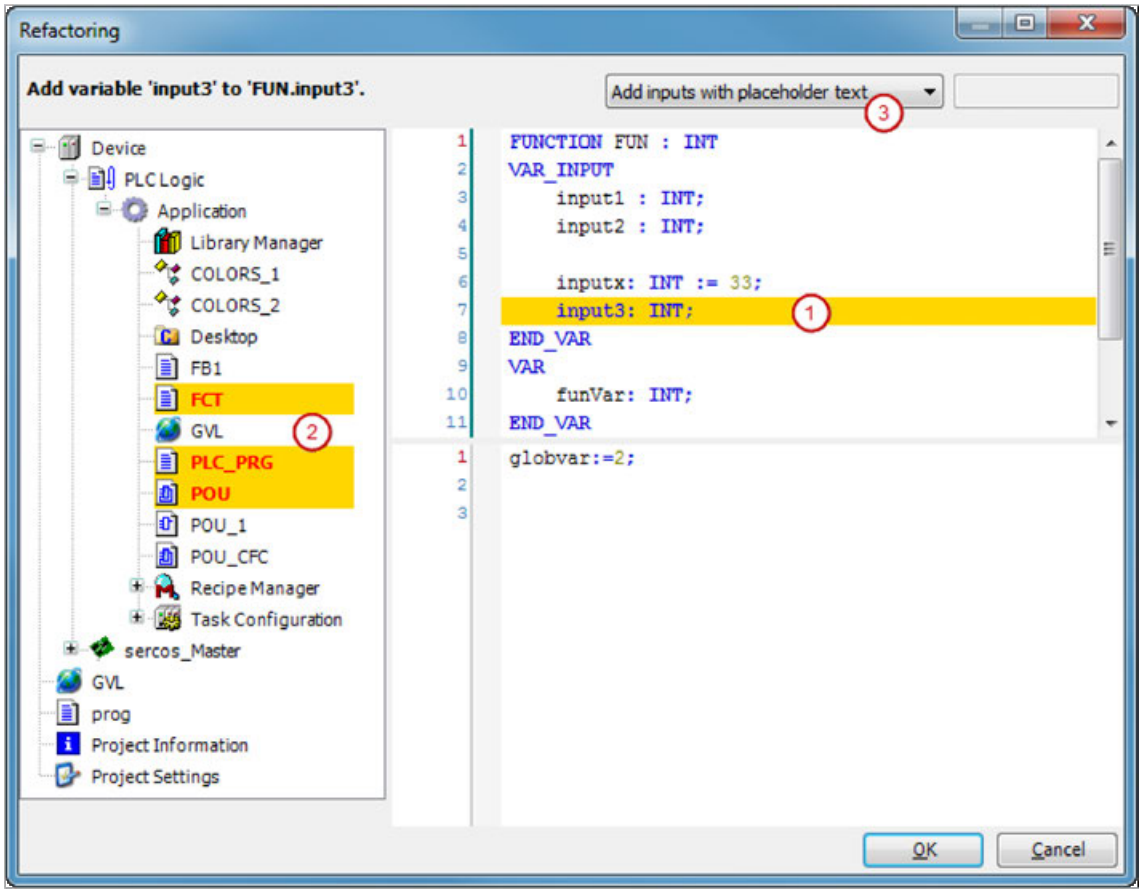
The command opens the default dialog box for declaring variables.

See also

- *"Dialog 'Auto Declare'" on page 975*

Dialog box 'Refactoring'

After clicking *"OK"* to close the declaration dialog, the *"Refactoring"* dialog box opens with two frames.



"Right dialog frame "	Declaration part and implementation of the POU where the variable is added. Colored highlighting of changed locations: New added declarations have a blue font and are highlighted in yellow (1).
"Left dialog frame "	Device tree or POUs tree of the project. Colored highlighting of blocks where the POU is used: red font and yellow highlight (2). After you double-click the POU object, the detail view opens.

Before you decide which changes to accept at which locations, select the required option from the drop-down list (3) at the upper right part of the window:

"Add inputs with placeholder text"	Default placeholder text: `_REFACTOR_`; editable The placeholder text defined here is used at the occurrence locations of the new added variables in the implementation code. This is used for searching for the affected locations.
"Add inputs with the following value"	Initialization value for the new variable.

You can accept or reject changes by right-clicking the changed locations or by executing commands in the left or right area of the dialog box. Refer to the description of the *"Refactoring ➔ Rename"* command.

Examples

1. By refactoring, the `fun` block receives a new input variable `input3` with the initialization value `1`. The change has the following effect:

Before:

```
fun(a + b, 3, TRUE);
fun(input1:= a + b , input2 :=3 , inputx := TRUE);
```

After:

```
fun(a + b, 3, 1, TRUE);
fun(input1:= a + b , input2 :=3 , _REFACTOR_, inputx := TRUE);
```

2. By refactoring, the "fun" block receives a new input variable `input3` with the placeholder text "_REFACTOR_":

Before:

```
inst(input1 := a + b, input2 := 3, inputx := TRUE);
fun(a + b, 3, TRUE);
```

After:

```
inst(input1 := a + b, input2 := 3, input3 := _REFACTOR_, inputx
:= TRUE);
fun(a + b, 3, _REFACTOR_, TRUE);
```

See also

- *Chapter 1.4.1.8.15 “Refactoring” on page 289*
- *Chapter 1.4.1.20.3.2.40 “Command 'Refactoring' - 'Rename <...>'” on page 980*

Command 'Refactoring' - 'Remove <variable>'

Symbol: 

Function: This command removes an input or output variable from the POU and all occurrences of the POU.

Call: Main menu *“Edit ➔ Refactoring”*, or right-click.

Requirements: In the declaration part of the POU, the cursor is located in the identifier of the variable to be removed.

Then, the command opens a dialog box with information about the removal. After you confirm this, the *“Refactoring”* dialog box opens. For a description of the *“Refactoring”* dialog box, refer to the *“Edit ➔ Refactoring ➔ Rename”* help page.

When you accept the changes in the *“Refactoring”* dialog box, the respective input and output parameters are deleted at the occurrence locations of the affected POU.

In CFC, only the connection is removed between the removed input or output to the block. The input or output itself remains in the chart.

Example in ST

In a POU, refactoring removes the `input4` input variable. The occurrences are updated automatically:

Before removal:

```
inst(input1 := a + b, input2 := 3, input4 := 1, input5 := TRUE);
fun(a + b, 3, 1, TRUE);
```

After removal:

```
inst(input1 := a + b, input2 := 3, input5 := TRUE);
fun(a + b, 3, TRUE);
```

See also

- *Chapter 1.4.1.8.15 “Refactoring” on page 289*
- *Chapter 1.4.1.20.3.2.40 “Command 'Refactoring' - 'Rename <...>'” on page 980*

Command 'Refactoring' - 'Reorder Variables'

Symbol: 

Function: This command allows changing the order of variables in the declaration editor for the selected scope: `VAR_INPUT`, `VAR_OUTPUT`, or `VAR_IN_OUT`.

Call: *“Edit ➔ Refactoring”*; context menu of the focused scope in the declaration editor.

Requirement: One of the above scopes is selected in the declaration, and more than one variable is declared in it.

The command opens the *“Reorder”* dialog box with a list of all declarations of the selected scope. You can drag a selected declaration up or down to another position.

See also

- *Chapter 1.4.1.8.15 “Refactoring” on page 289*

Command 'Advanced' - 'Format Document'

Symbol: 

Function: The command starts an automatic formatting of the code in the open ST editor.

Call: Menu bar: *“Edit ➔ Advanced”*; context menu of the window in focus in the ST editor

Requirement: The focus is in the ST editor. The syntax of the ST code does not contain any errors.

The following formatting is performed automatically:

- Keywords are converted to uppercase letters.
- Spacing is standardized.
- Indentations are changed according to syntax.
- Long lines are wrapped in sensible places.

See also

- *Chapter 1.4.1.19.1.3.1 “ST Editor” on page 463*

Menu 'View'

Standard Menu in View 'Devices', 'POUs', 'Modules'

The views *“Devices”*, *“POUs”* and *“Modules”* provide the button in the top right corner to open a menu with the following commands:

- *“Open in editor”*: Opens the selected object in the corresponding editor.
- *“Find object”*: Opens the dialog *“Find Object”* for the object tree. Starting to enter a search string all matching objects will be displayed with their path. Use the button *“Open”* to open the selected search result in the editor.
- *“Sort by type”*: Sorts the objects in the view alphabetic by type.
- *“Sort by name”*: Sorts the objects in the view alphabetic by name.
- *“Sort ascending”*: Displays the chosen sorting in ascending order.
- *“Sort descending”*: Displays the chosen sorting in descending order.
- *“Track active editor”*: CODESYS selects the object, that is opened in the active editor, in the device tree of the view.

See also

- *Chapter 1.4.1.20.3.3.2 “Command 'Devices'” on page 985*
- *Chapter 1.4.1.20.3.3.3 “Command ‘POUs’” on page 986*
- *Chapter 1.4.1.20.3.3.4 “Command 'Modules'” on page 986*

Command 'Devices'

Symbol: , view: *[Alt]* + *[0]*

Function: The command opens the view *“Devices”* in the CODESYS main window. The view contains the project's "device tree", where you configure your applications

Button opens the standard menu for navigating in the tree view.

Call: Menu *“View”*

See also

- *Chapter 1.4.1.20.3.3.3 “Command ‘POUs’” on page 986*
- *Chapter 1.4.1.20.3.3.1 “Standard Menu in View 'Devices', 'POUs', 'Modules'” on page 985*

Command ‘POUs’

Symbol: , Shortcut: *[Alt]* + *[1]*

Function: This command opens the *“POUs”* view in the CODESYS main window. POUs located here are available in the entire project.

Call: Menu *“View”*

See also

- *Chapter 1.4.1.20.3.3.1 “Standard Menu in View 'Devices', 'POUs', 'Modules'” on page 985*

Command 'Modules'

Symbol:

Function: This command opens the *“Modules”* view and shows the modules of the application composer in a tree structure.

Call: Main menu *“View”*

See also

- *Chapter 1.4.1.20.3.3.1 “Standard Menu in View 'Devices', 'POUs', 'Modules'” on page 985*

Command 'Messages'

Symbol:

Function: This command opens the *“Messages”* view.

Call: Menu bar: *“View”*.

View 'Messages'

Message category	The messages are categorized by component or functionality for selection from a drop-down list. Filter the message display by selecting a category.
Message type	Click the symbol of the message type to show or hide messages. CODESYS displays the number of messages next to each symbol. ● : Error ● : Warning ● : Message
	Deletes all messages in the selected message category.
	Deletes all messages in all message categories.
“Description”	Message text with the reported object and the location in the object. Double-click a message in the table to jump to the source text location.
“Project”	
“Object”	
“Position”	

Table 117: "Commands in the context menu"

"Next Message"	The source text position of the next message is displayed.
"Previous Message"	The source text position of the previous message is displayed.
"Go to Source Position"	The source position of the selected message is displayed.

Command 'Element properties'

Symbol: 

Function: This command opens the *"Element Properties"* view.

Call: Main menu *"View"*

This command opens the properties view for the open object. This view is available only for a few objects, for example visualization and POU (SFC).

The properties are displayed in a structured table. You change the property values by clicking into the value fields. You can filter or sort the properties view.

Command 'ToolBox'

Symbol: 

Function: This command opens the *"ToolBox"* view.

Call: Main menu *"View"*

This command opens the toolbox view for the open object. By default, this view is available for graphical editors and visualizations. It includes the graphical programming elements that you can drag into the editor.

Command 'Watch' - 'Watch <n>'

Symbol: 

Function: This command opens the "Watch <n>" view. You can populate a watchlist with variables from your project in order to monitor, force, or write these variable values in an individual view in online mode. The value "n" can be 1, 2, 3, or 4 for a total of up to four watchlists.

Call: Main menu *"View"*

See also

- *Chapter 1.4.1.12.1.2 "Using watch lists" on page 416*

Command 'Watch' - 'Watch All Forces'

Symbol: 

Function: The command opens the *"Watch All Forces"* view, which is a special kind of watch list.

Call: Menu bar: *"View* ➔ *Watch* ➔ *Watch All Forces"*

Requirement: A project is in offline mode or online mode.

The view contains all variables currently prepared for forcing, and all forced variables of the application in one list. Actions are possible in the list which are also possible in other watch lists. Moreover, the following commands are available in the *"Unforce"* list box of the view:

Table 118: “Watch All Forces”

Tabular view of all forced variables of the application, including variables prepared for forcing	
“Expression”	Variable name
“Data Type”	Data type of the variable
“Value”	Currently forced value of the variable
“Prepared Value”	Value prepared for forcing
“Overwritten value at start of cycle”	For inputs, the actual value is already overwritten by the force value before the application code is executed. As a result, this is the forced value. For outputs, this is the forced value.
“Overwritten value at end of cycle”	For outputs, this is the value which is calculated in the cycle. However, this value is overwritten by the force value at the end of the cycle. For inputs, this is the forced value.

- *“Unforce and Keep All Selected Values”*: For all selected entries in the list, the variables will be set to the forced value and the forcing will be lifted
- *“Unforce and Restore All Selected Values”*: For all selected entries in the list, the variables will be reset to the values they had before they were forced, and the forcing will be lifted.

See also

- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*

Command 'Add All Forces to Watchlist'

Function: The command adds all variables of the active application, which are currently prepared for forcing, or which are already forced, to the watchlist. Please regard, that this works only for docked watch list views.

Call: Context menu of view *“Watch”*

Requirement: Online mode, a watch list view is active.

There is a special watch list: “Watch All Forces”. This view shows automatically all variables currently prepared for forcing or already being forced. It provides additional commands for releasing any forces.

See also

- *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*
- *Chapter 1.4.1.20.3.3.8 “Command 'Watch' - 'Watch <n>'” on page 987*
- *Chapter 1.4.1.20.3.3.9 “Command 'Watch' - 'Watch All Forces'” on page 987*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*

Command 'Bookmarks'

Symbol:

Function: This command opens the *“Bookmarks”* view.

Call: Menu bar: *“View”*.

"Previous Bookmark"	Jumps to the bookmark that above the selected bookmark in the table and opens the respective POU in the editor.
"Next Bookmark"	Jumps to the bookmark that below the selected bookmark in the table and opens the respective POU in the editor.
	Deletes the selected bookmark from the table and in the respective POU.
List of bookmarks in the project with the following information: *"Bookmark"*, *"Object"*, and *"Position"*. You can edit the bookmark order per drag&drop. When you double-click a row, CODESYS opens the respective *"Object"* in the editor and jumps to this bookmark.	
"Bookmark"	Name of the bookmark as assigned by CODESYS in ascending numerical order: *"Bookmark_0"*, *"Bookmark_2"* etc. If the bookmark is selected and you click in the field, then it is editable and you can modify the bookmark name.
"Object"	Name and project path of the POU where the bookmark is set Example: `POU_Add [PLC_1: SPS-Logic: Application]`
"Position"	Position of the bookmark in the POU Example: `Row 3, Column 1 (Impl)` `(Impl)`: in the implementation part of the POU `(Decl)`: in the declaration part of the POU

See also

- *Chapter 1.4.1.8.13.3 "Setting and using bookmarks" on page 287*
- *Chapter 1.4.1.20.3.2.24 "Command 'Next Bookmark'" on page 973*
- *Chapter 1.4.1.20.3.2.26 "Command 'Previous Bookmark'" on page 973*

Command 'Breakpoints'

Symbol:

Function: This command opens the *"Breakpoints"* view.

Call: Menu bar: *"View"*.

This view shows an overview of all defined breakpoints for an application. You have access to all breakpoint commands within this view.

Table 119: Table of current breakpoints

"Application"	Select the required application from the list.
"POU"	Name of the function block that will receive the breakpoint
"Location"	Location of the breakpoint in the POU ● Text editor: Line number and column number ● Graphical editor: Network number or element number For function blocks, "(Impl)" indicates that the breakpoint is located in the implementation of the function block, not in an instance.
"Instance Path"	Complete object path of the breakpoint location.
"Tasks"	Names of tasks that will be effective when the breakpoint is executed. If there are no restrictions, then "(all)" is displayed here.

"Condition"	• *"Break always"*: No additional enable condition defined; the breakpoint is always enabled. • Boolean expression. The expression must yield `TRUE` for the breakpoint to be enabled.
"Hit Count Condition"	When the breakpoint should be in effect (depending on the hit count)
"Current Hit Count"	How often the breakpoint has already been reached up to now during the execution

Table 120: Toolbar

	*"New Breakpoint"*	This command opens the *"Breakpoint Properties"* dialog.
	"New Data Breakpoint"	This command opens the *"New Breakpoint"* dialog.
	*"Clear Breakpoint"*	Removes the breakpoint (not the same as disable)
	*"Enable/Disable Breakpoint"*	Toggles the status of the breakpoint or execution point between "enabled" and "disabled" • Breakpoint enabled • Breakpoint disabled • Execution point enabled • Execution point disabled • Data breakpoint enabled • Data breakpoint disabled • Data execution point enabled • Data execution point disabled As opposed to "Clear breakpoint", a disabled breakpoint remains in the list and can be enabled again.
	*"Properties"*	The *"Breakpoint Properties"* dialog opens for editing the breakpoint parameters. This dialog is the same as *"New Breakpoint"*. In online mode, you can change the breakpoint into an execution point.
	*"Go to Source Position"*	Opens the online view of the affected block. The cursor is set at the breakpoint location.
	*"Clear All Breakpoints"*	Deletes all breakpoints and execution points in the application. The list is cleared. Not to be confused with "deactivate".
	*"Enable All Breakpoints"*	Enables all currently disabled breakpoints and execution points.
	*"Disable All Breakpoints"*	Disables all currently enabled breakpoints and execution points. The points remain in the list and can be enabled again.

See also

- *Chapter 1.4.1.20.4.5 "Dialog 'Breakpoint Properties'" on page 1151*
- *Chapter 1.4.1.20.3.7.4 "Command 'New Breakpoint'" on page 1049*
- *Chapter 1.4.1.20.3.7.5 "Command 'New Data Breakpoint'" on page 1049*
- *Chapter 1.4.1.20.3.7.7 "Command 'Enable Breakpoint'" on page 1050*
- *Chapter 1.4.1.20.3.7.8 "Command 'Disable Breakpoint'" on page 1050*
- *Chapter 1.4.1.20.3.7.9 "Command 'Toggle Breakpoint'" on page 1050*

Command 'Cross Reference List'

Symbol: 

Function: This command opens the *"Cross Reference List"* view.

Call: Menu bar: *"View"*, or *"Edit ➔ Browse ➔ Browse Cross References"*.

This view shows a list of cross-references for a symbol in the project. The symbol can be a variable, a POU (program, function block, function), or a user-specific data type (DUT). The cross-reference list offer two basic types of searches:

- Text search: By specifying a symbol name, the cross-references of all symbols in the project are displayed with their names. If multiple symbols with the same are found, then the display can be limited to individual declarations by means of the context menu.
- Declaration search: The symbol can be selected by means of the input assistant or by specifying a qualified path (for example, `Device.Application.PLC_PRG.i` or `__POOL.POU.a`). Then only the occurrence locations of this symbol are displayed, even if there exist other symbols with the same name.

Input field	Symbol name (variable name, POU name, DUT name). Input options: ● Selection of a declared symbol by means of the input assistant (button). ● Manual input of the symbol name. Triggering of the search by pressing the button or the *[Enter]* key. For the text search, you can use the placeholders "`*`" (any number of characters) or "`?`" (exactly any one character) in combination with a partial string of a variable identifier. Use the percent sign "`%`" to search for IEC addresses. Examples: "`%MW8`", "`%M*`". More options outside of cross-reference list view: ● Use the command *"Browse for Symbol ➔ Browse Cross References"* if the name of a declared symbol is selected in an editor, or if the cursor is in the name field. A search is also possible if the object is selected in the device tree or POU pool. ● Automatic if the name of a declared symbol is selected in an editor, or if the cursor is in the name field. A automatic search is also possible if the object is selected in the device tree or POU pool. Requirement: CODESYS option *"Automatically list selection in cross reference view"* is activated (category *"SmartCoding"*. The following input is valid: ● Variable name, simple or qualified. Examples: "`iVar`", "`PLC_PRG.iVar`". ● POU name: Examples: "`PLC_PRG`", "`myFB`". ● DUT name: Example: "`mySTRUCT`". ● Strings combined with placeholders: asterisk (*) for any character or question mark (?) for exactly one characters). Example: "`iVar*`" applies to `iVar1`, `iVar_glob2`, `iVar45`, etc. "`iVar?`" refers to `iVar1`, `iVar2`, `iVarX`, and so on, but not `iVar_glob2`, `iVar45` and so on... ● "`%<IEC address>`": CODESYS searches for variables that are assigned to this address and direct memory access. Example: "`%QB0`", `%Q0 := 2`.
	Open input assistant for selecting a symbol
	Perform a search
	Define columns to search for the string.
Input field	String that is searched for in the selected columns. The result locations are marked in yellow. Cross references without this string are hidden.
	Show source position of previous cross-reference, *[Shift]+[F4]*
	Show source position of next cross-reference, *[F4]*
	Limit results to current declaration: Available if multiple declarations are found for a symbol. Limits the display to the declaration that you have selected in the list.

	Show source position of selected cross-reference: The focus jumps to the occurrence location of the symbol.
	Print cross-reference list: The default dialog opens for setting up a print job.
The cross references are displayed with the following information:	
“Symbol”	The result locations for the symbols (variables, POUs, or DUTs) are grouped by declaration. The declaration occurrence comprises the root node and the occurrence locations in the project are indented below. The precise expression is displayed that has the symbol at the occurrence location. Example: If there is a global variable `i` in the project and a local declared variable `i` in a POU, then two root node entries will be listed after a text search for cross-references with the occurrences of the variable `i` below each.
“POU”	Block name; also a task name if a block call in the task configuration.
“Variable”	Only the variable name (for example, `iVar`)
“Access”	Type of access to the variable at the occurrence location: *“Declaration”* / *“Read”* / *“Write”* / *“Call”*. Special case for pointers: An assignment type `p := ADR(var1)` is displayed as `write | address` when searching for `var1`. The reason for this: Any write access to `p` is not displayed when searching for `var1`. Write access is also possible by means of pointer variables.
“Type”	Data type of the variable
“Address”	IEC address if variables are assigned Example: "`AT %QB0`".
“Position”	Location of the occurrence in the POU editor, for example line number, network number, declaration part, or implementation part. Example: "`line 1, column 1 (Impl)`".
“Object”	POU name plus complete path of the occurrence location in brackets (if this is found in the *“Devices”* view). Example: "`PLC_PRG [Device:Plc Logic:Application]`"
“Comment”	Comments if available in the declaration of the variable

The search yields all result locations in the project and in included, uncompiled libraries.

Right-click commands in the cross-reference list

“Show source position”: Opens the respective POU and marks the occurrence: for root entries, the declaration, and for subordinate entries, the respective occurrence location. As an alternative, you can double-click a line.

“Limit Results to Selected Declaration”: Limits the display of results to the selected symbol declaration if multiple declarations are found.

“Expand All”: In the list, every single result location is shown.

“Collapse All”: In the list, only the root nodes of the result locations are shown.

See also

- *Chapter 1.4.1.20.3.2.29 “Command 'Browse Cross References'” on page 974*
- *Chapter 1.4.1.8.13.1 “Using the cross-reference list to find occurrences” on page 285*
- *Chapter 1.4.1.20.3.22.3 “Command 'Limit Results to Current Declaration'” on page 1148*
- *Chapter 1.4.1.20.3.2.18 “Command 'Collapse All Folds'” on page 971*
- *Chapter 1.4.1.20.3.2.17 “Command 'Expand All Folds'” on page 971*

Command 'Browse Cross References in Classic View'

Symbol 

Function: This command opens the *“Classic Cross Reference List”* view.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog box from *“Tools ➔ Customize”* (command category *“Browse Project”*).

The view corresponds to the *“Cross Reference List”* view before CODESYS V3.5 SP6.

Command 'Call Stack'

Symbol: 

Function: This command opens the *“Call Stack”* view.

Call: Main menu *“Debug”*.

This view is very useful when you want to step into programs. It shows the current location with the complete call path.

“Application”	Name of the active application that controls the current POU
“Task”	Name of the task that controls the current POU
“POU”	Name of the POU where program execution has halted The first line in the list describes the current execution location (marked with a yellow arrow). If this location is in a block that is called by another block, then the call location is described in the second line. In turn, if the caller is called by yet another block, then that call location is described in the third line, and so on.
“Location”	Position within the POU where program execution has halted ● Line and column numbers for textual editors ● Network or element numbers for graphical editors
“Instance path”	Instance where program execution has halted

The call stack is also available in offline mode and normal online mode when you are not currently using any debugging functions). In this case, it receives the last displayed location during a stepped execution, but it is displayed in gray.

The “Call Tree” view, in contrast to the “Call Stack”, at any time provides information on the calls of a POU.

See also

- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*
Chapter 1.4.1.20.3.3.16 “Command 'Call tree'” on page 993

Command 'Call tree'

Symbol: 

Function: This command opens the *“Call Tree”* view.

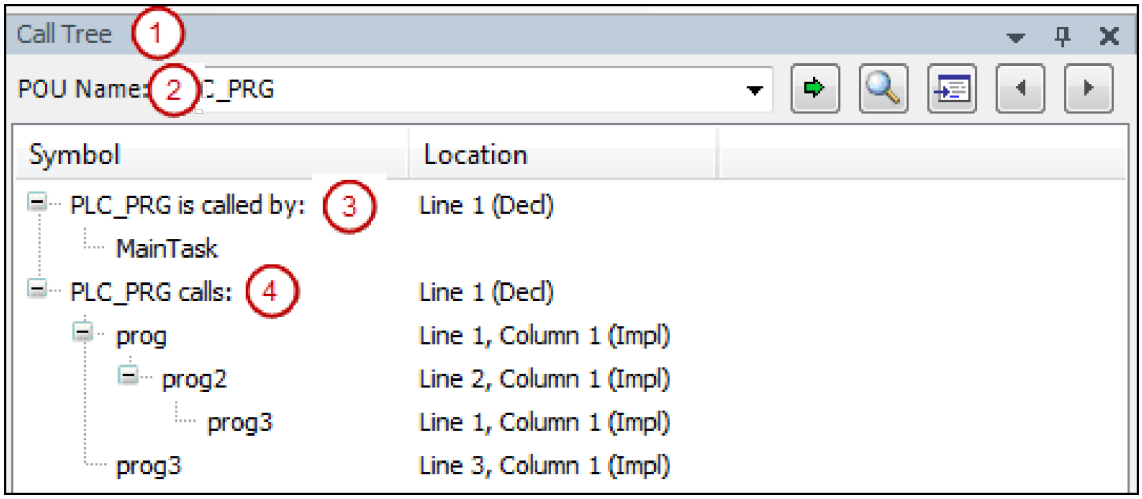
Call:

- *“View”* menu
- Context menu of a callable block in the *“Devices”* or *“POUs”* view.

View 'Call tree'

The call tree is available at all times, even before compiling the application. It is a static representation of the caller and the calls of the block that you specify explicitly. Therefore, the tree always contains two root nodes above the respective call order is displayed as successive indented entries. Recursive calls are detected quickly in this tree representation.

Example of a call tree (1) for the (2) PLC_PRG block:



- (3) Node *"<block name> is called by:"*
- (4) Node *"<block name> calls:"*

"Block name"	Name of the program block; specified manually, by dragging from another view, or by means of the button . The drop-down list includes the last specified block names.
Toolbar and keyboard usage	
: Find block	CODESYS searches for the block specified in *"Block name"* and displays its caller and calls.
: Use block from the input assistant	The *"Input Assistant"* dialog box opens for selecting a block call or instance call. The call tree is refreshed automatically after the selection.
: Show source code position of the selected block	CODESYS jumps to the occurrence location of the block in the source code of the program.
[F4]: Show source code position of the next block *[Shift]+[F4]*: Show source code position of the previous block	The selection in the call tree jumps to the next or previous block in the call structure. At the same time, the associated source code position is opened in the respective editor. Note: Double-clicking an entry in the call tree also opens the associated source code position.
Display of the call tree:	
"Symbol"	*"<block name> is called by"*: The call order is displayed for below this node. The bottom entry in this tree structure shows the start of the calls. *"<block name> calls"*: The calls from this block are displayed below this node. The bottom entry in this tree structure shows the end of the call chain.
"Position"	For the root node in the call tree: Line numbers of the declaration (*"Decl"*) of the block. For the caller or calls below the root node: Line number, column number, and network number of the position, depending on the implementation language.
Context menu for the entry selected in the tree:	
"Collapse All"	The expanded entries in the call tree are collapsed, except for the two root nodes.

"Show Source Position"	CODESYS jumps to the occurrence location of the block in the source code of the program.
"Set as New Root Node"	The entry selected in the call tree is displayed in *"Block name"*. The tree is refreshed automatically for the new root nodes.

The "Call list" view is provided for immediate information when stepping through a program, as opposed to the static call tree that provides call information about a block. The call list always shows the full call path of the current position that is reached.

See also

- *Chapter 1.4.1.11.3 "Stepping Through a Program" on page 399*
- *Chapter 1.4.1.20.3.3.15 "Command 'Call Stack'" on page 993*
- *Chapter 1.4.1.20.3.2.31 "Command 'Browse Call Tree'" on page 975*

Command 'Memory'

Symbol:

Function: In CODESYS V3.5 version earlier than SP11, the command opens the *"Memory"* view.

Call: Menu bar: *"View* ➔ *Memory"*.

As of SP11, the command provides the notice that you must install the CODESYS Memory Tools package (available in the CODESYS Store) in order to use the memory view. After installation, you can open the *"Memory"* view by clicking *"View* ➔ *Show Memory View"*.

Command 'Security Screen'

Symbol:

Function: The command opens the *"Security Screen"* view.

Call:

- *"View"* menu
- icon or in the status bar
  The icon is displayed in blue when a valid certificate is specified for the digital signature. When only one client certificate is specified for the encrypted communication, the icon remains gray, resulting in the client certificate providing no increased security for the user.

The following security features of CODESYS are configured and displayed in the view:

- Personal user certificate
- Encrypted communication
- Encryption and signatures of IEC projects
- Encryption and signature of download, online change, and boot application
- Security level

NOTICE!
When the *"Security Screen"* is opened and closed, the current settings are applied in the user options, even when no active changes have been made.

If the CODESYS Security Agent add-on product is installed, then the "Security Screen" view provides an additional "Devices" tab. This allows for the configuration of certificates for the encrypted communication with controllers.

Tab 'User'

On this tab, certificates are configured that are required for the encrypted communication and the digital signature of the user. Only certificates with private keys can be specified here. The user profile is saved as an XML file in the user options.

"User Profile and Certificate Selection"	By default, the login name for Windows is specified as the user profile.
List box with existing user profiles	: Opens the *"User Profiles"* dialog. Here you specify the name for a new user profile. : Deletes the selected user profile. This user profile is no longer displayed in the list box.
"Digital Signature"	: Opens the *"Certificate Selection"* dialog for selecting the certificate for the digital signature. One certificate can be selected. The certificate has to have a private key. : Deletes the displayed certificate. One certificate can be selected. The certificate has to have a private key.
"Project File Decryption"	: Opens the *"Certificate Selection"* dialog for selecting the certificate for decrypting project files. One certificate can be selected. The certificate has to have a private key. : Deletes the displayed certificate.

See also

- *Chapter 1.4.1.20.4.18 "Dialog 'Certificate Selection'" on page 1215*

Table 121: "Security Level"

"Activate the Use of Certificates for Enhanced Security"	
"Enforce encrypted communication"	: When the user communicates with the controller, the server certificate of the controller is used for establishing an encrypted connection. Then the entire communication is encrypted.
"Enforce encryption of project files"	: All project files of the user are encrypted with a certificate. When the project is saved, it is encrypted with the certificate specified in the project settings (*"Project Settings ➔ Security"* dialog). The selected certificate is displayed on the *"Project"* tab in the *"Project file encryption"* group. To open this project, the certificate to be encrypted has to be specified in *"Project file decryption"* with a private key.
"Enforce signing of project files"	: All project files of the user are signed with a certificate. In *"Digital Signature"*, a certificate has to be specified with a private key. When a project is saved, a signature file `<project name>.project.p7s` is generated in the project directory containing the signature.

"Enforce encryption of downloads, online changes and boot applications"	☑: The data that is downloaded to the controller has to be encrypted with a controller certificate. This certificate is defined directly either in the properties dialog of the application on the *"Encryption"* tab, or in the security screen, on the *"Project"* tab, in the *"Encryption of Boot Application, Download and Online Change"* group. Controller certificates are located in the local Windows Certificate Store in the *"PLC Certificates"* directory. If the certificates of your controller are not available in the directory, then they first have to be loaded from the controller and installed to the directory. For instructions, see the "*"Controller Certificates"*" chapter.
"Enforce signing of downloads, online changes and boot applications"	☑: The online code (downloads, online changes, and boot applications) have to be signed with a certificate with a personal key. The certificate is selected from the *"Digital Signature"* area. Requirement: The *"Encryption of boot application, download and online change"* option is selected.
"Enforce signing of compiled libraries"	☑: The *"File ➔ Save Project as Compiled Library"* command generates a signed library `<library name>.compiled-library-v3`. Requirements ● A certificate with a private key that supports code signing is available. ● A library compatibility >= CODESYS V3 SP15 is set in the project information. *"Enforce timestamping of signed compiled libraries"*: ☑: The URL of the time stamp server which created the time stamp has to be entered in the *"Timestamping server"* field. Example: `timestamp.comodoca.com/rfc3161`.

See also

- *Chapter 1.4.1.15 "Using the Command-Line Interface" on page 442*
- *Chapter 1.4.1.20.3.1.7 "Command 'Save Project as Compiled Library'" on page 960*
- *Chapter 1.4.1.16.1 "Information for Library Developers" on page 449*

Tab 'Project'

All project-specific settings are configured on this tab. These elements are active only when a primary project is loaded.

"Project file encryption"	
"Technology"	☑: Opens the *"Project Settings ➔ Security"* dialog When you select the *"Encryption"* project setting and then *"Certificates"* in the dialog, you can choose a corresponding certificate by clicking ☑. For more information, see the description of the "Project Settings: Security" dialog.
"Certificates of Users Sharing this Project"	Area for listing the certificates that encrypt the project file.

“Encryption of Boot Application, Download and Online Change”	
List of the applications of the controller	Double-clicking an application in the list opens the *“Properties ➔ Encryption”* dialog. Depending on the settings of the *“Security Level”* on the *“User”* tab of the *“Security Screen”*, the following fields are available in the open properties dialog: ● *“Encryption”* tab with active *“Certificates”* area ● *“Encryption”* tab with *“Encryption Technology”* list box. In the *“Properties ➔ Encryption”* dialog, click the button to select the controller certificate for *“Encryption of Boot Application, Download and Online Change”*. For more information, see the description of the "Properties: Encryption" dialog. Controller certificates are located in the local Windows Certificate Store in the *“PLC Certificates”* directory. If the certificates of your controller are not available in the directory, then they first have to be loaded from the controller and installed to the directory. For instructions, see the "Protecting and Saving a Project" - " Encryption with Certificates" chapter.

See also

- *Chapter 1.4.1.20.4.11.7 “Dialog 'Project Settings' - 'Security'” on page 1176*
- *Chapter 1.4.1.20.4.10.3 “Dialog 'Properties' - 'Encryption'” on page 1158*
- *Chapter 1.4.1.5.7 “Encrypting Projects with Certificates” on page 207*

Tab 'Devices'

This tab is available only after you have installed the CODESYS Security Agent add-on. For a description of this tab, see the help for the CODESYS Security Agent.

Command 'Settings of Memory Reserve for Online Change'

Function: This command opens the *“ Online Change Memory Reserve”* view.

Call: Menu bar: *“View”*.

In the view, memory reserves are configured for the function blocks during the online change.

“Scan Application”	● Searches the selected application for function blocks and displays them in the *“Function blocks”* area ● Updates the *“Function blocks”* area after the application is built again. ● Updates the *“Function blocks”* area after an online change.
Drop-down list with the applications of the open project	Selection of the application whose function blocks should be displayed and/or edited in this view.

Table 122: “Function Blocks”

“All”	All function blocks of the selected application are displayed.
“Pool”	All function blocks of the *“POUs”* view that are displayed which are referenced in the application.
“No memory-reserve”	All function blocks with a memory reserve of 0 bytes are displayed.
“<memory reserve> bytes”	Display of all function blocks with the number of bytes is displayed that is defined in *“Memory reserve”*.
Information about the function blocks Multiple selection is also possible when selecting a POU for the configuration of the memory reserve.	

“Function block”	Name of the function block
“Size”	Size of the function block Size of an instance of a function block Specified in bytes
“Number of instances ”	Number of instances of a function block in the project
“Memory reserve”	Display of the memory reserve for each instance of the function block
“Additional memory for all instances”	Product of *“Number of instances”* and *“Memory reserve”*
“Remaining memory reserve”	Number of bytes that are available as reserve.

Table 123: “Settings”

“Memory reserve (in bytes)”	Input field for the memory reserve for the selected function block. Specified in bytes Requirement: the application is not located on the controller yet or you have allowed the memory reserve to be changed by clicking the *“Edit”* button in the *“Allow editing”* area.
“Apply for Selection”	The *“Memory reserve (in bytes)”* is assigned to the function block and the table column *“Memory Reserve”* is updated. In multiple selection, the specified value is assigned to each function block. In order to update the columns *“Size”*, *“Number of Instances”*, *“Additional Memory for All Instances”*, and *“Remaining Size of the Memory Reserve”*, click *“Build ➔ Build”*, and then click the *“Scan Application”* button.

Table 124: “Enable Editing”

“Enable”	The input field *“Memory reserve (in bytes)”* is editable. This button is modified in *“Editable”*.

Table 125: “Information”

“Number of FBs”	Total number of function blocks in the application
“Additional memory for all instances”	Sum of the memory reserves of all function block instances of the application. Specified in bytes

See also

- *Chapter 1.4.1.20.3.6.6 “Command 'Online Change'” on page 1033*

Command 'Start Page'

Symbol: 

Function: This command opens the *“Start Page”* view.

Call: Main menu *“View”*

The view includes some basic commands and a list of recently opened projects. In addition, the CODESYS homepage is displayed.

If you access the Internet through a proxy, then you can save the authentication data in the project options (“Proxy Settings”) so you do not have to provide this data every time you use this command.

By moving the mouse pointer over the list of recently opened projects, you can remove or pin individual projects in the list. Pinned projects remain in this list until you remove the pin.

In the project options (*“Load and Save”*), you can configure whether this start page should open automatically when you start CODESYS.

See also

- *Chapter 1.4.1.20.4.13.16 “Dialog 'Options' – 'Load and Save'” on page 1196*
- *Chapter 1.4.1.20.4.13.20 “Dialog 'Options' - 'Proxy Settings'” on page 1198*

Command 'Full Screen'

Symbol: , keyboard shortcut *[Ctrl]+[Shift]+[F12]*

Function: This command switches the CODESYS display to full screen mode.

Call: Main menu *“View”*

Choosing this command displays the main window of the CODESYS user interface in full-screen mode. You can return to the previous setting by choosing the command again or with the keyboard shortcut *[Ctrl]+[Shift]+[F12]*.

Command 'Properties'

Symbol:

Function: This command opens the properties of the currently selected object in the POUs tree or device tree.

Call: Main menu *“View”*

Menu 'Project'

Command ‘Add Object’

Symbol: 

Function: This command opens a submenu with objects that contain all objects that can be inserted, depending on the current position in the *“Devices”* or *“POUs”* view.

Call: *“Project”* menu, context menu in the *“Devices”* or *“POUs”* view.

Requirement: If CODESYS is to insert the object in the device tree, select an already existing object under which the new one is can be inserted indented. If CODESYS is to insert the object in the POUs tree, set the focus in any free place in the CODESYS window.

Command ‘Add Folder’

Symbol: 

Function: This command opens a dialog box for defining a new folder in the Devices or POUs view.

Call: *“Project”* menu, context menu in the Devices or POUs view

> *You cannot structure the arrangement of device nodes and device objects through folders that you have created yourself.*

This command inserts the folder below the object that has just been selected in the tree. If no object is selected, CODESYS inserts the folder right at the top in the tree directly under the root node.

Command 'Insert Device'

Function: this command opens the dialog box *“Add Device”* for the selection of a device object that is to be inserted in the device tree below the currently selected object.

Call: Context menu of a device object in the device tree.

Requirement: An object is selected in the device tree below which a device object can be inserted.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*

Dialog box 'Add device'

Function: Depending on the currently selected position in the device tree, the dialog box offers a selection of the devices that can be inserted at this point. In addition, it contains the commands also available in the context menu: *“Insert Device”*, *“Add Device”*, *“Plug Device”*, *“Update Device”*.

Requirement: The devices are installed in the device repository on the local system.

> *If you have opened the dialog box, it always displays the selection to suit the object currently selected in the device tree until you click “Close”.*

“Name”	Name with which the device is to appear in the device tree. Must be a valid IEC identifier.

Table 126: “action”

“Add device”	CODESYS inserts the selected device indented below the selected object in the device tree.
“Insert device”	CODESYS inserts the selected device at the same level as the selected object below it in the device tree.

“Plug device”	CODESYS inserts the selected device in the selected slot. If the slot is already occupied, the existing module is replaced by the new one.
“Update device”	CODESYS replaces the device selected in the device tree by the one selected. Please note: Depending on the device, this may cause the configuration already done in the device editor to be overwritten with the default values!

“String for the full text search”	This field is editable after clicking in it. For any character string entered, only those devices that include the character string are displayed in the lower view. The matched string is highlighted in yellow for these devices.
“Vendor:”	Drop-down list with manufacturers whose available devices are displayed.
“Group by category”	☑: The available devices (newest version) are sorted by category. The category is defined in the device description file. ☐: The available devices appear flat and alphabetically sorted.
“Display all versions (for experts only)”	☑: In addition, all other available versions of the devices can also be selected. ☐: Only the newest version of each device is available for selection
“Display outdated versions”	☑: In addition, outdated versions of the devices can also be selected. Outdated versions result, for example, from the update of plug-ins. ☐: Outdated device versions are not displayed.

The information provided by the device description file is displayed: device name, vendor, categories, version, order number and a short description, device-specific bitmap.

See also

- *Chapter 1.4.1.20.3.4.31 “Command 'Insert Device'” on page 1017*
- *Chapter 1.4.1.20.3.4.4 “Command 'Plug Device'” on page 1003*
- *Chapter 1.4.1.20.3.4.6 “Command 'Update Device'” on page 1005*

Command 'Plug Device'

Function: Like the command *“Add Device”*, this command opens the dialog box *“Add Device”* for the selection of a device object that is to be inserted in the device tree in the currently selected slot.

Call: Context menu of the slot of a device object in the device tree.

Requirement: The slot of a device object is selected in the device tree.

An empty slot is identified by the symbol and the entry "<empty> (<empty>)". An occupied slot is given the symbol and the name of the device.

In the case of an occupied slot, this command replaces the existing module with the new one.

See also

- *Chapter 1.4.1.20.3.4.3 “Command 'Insert Device'” on page 1002*
- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*

Command 'Scan for Devices'

Function: The command establishes a brief connection to the hardware and determines the devices in the network. Then you can apply the devices found into the device tree of your project.

Call: Menu bar: *“Project”*; context menu of a device object in the device tree

Requirement: The communication settings to the controller are correct. The gateway and the PLC are started. The device supports the scan function.

The following devices provide the scan function: EtherCAT master, EtherNet/IP Scanner (IEC), Sercos master, CANopen Manager, CANopen Manager SIL2, PROFINET controller und PROFIBUS DP master.

You can perform the device scan immediately if the scan function is permanently implemented in the PLC. When scan function is implemented in a library, you have to log in only one time to download the library to the controller.

The command refers to the master controller selected in the device tree. For example, an already inserted PROFINET IO controller can be selected and the command used to determine the I/O devices and I/O modules assigned to it.

After performing the scan operation, the *“Scan Devices”* dialog opens and displays the found devices.

Dialog 'Scan Devices'

Table 127: “Scanned Devices”

“Device name, Device type, Address, Station name, etc.”	Data about the scanned device depending on network type. When you change a value in the list of scanned devices, the value is shown in italics. This indicates that the new value has been changed in the editor in CODESYS, but not in the device. When you download the value to the device, it is shown normally. Value that indicate differences between the project and the scanned device are shown in orange. If multiple device descriptions are available for the scanned device, then the name is displayed in bold. The selection of the matching device description is resolved differently for different fieldbuses. For more detailed information, see the corresponding fieldbus chapters. If a device description cannot be found, then the following message is shown: "Attention! The device was not found in the repository." Depending on the bus system, additional information is displayed, such as manufacturer number and product number. The device cannot be inserted into the project without the installed device description.
“Show differences to project”	☑: The table in the dialog also shows additional configured devices (in the device tree of the project). ☐: The table shows all scanned devices. The configured devices are not shown.
“Scan for Devices”	Starts a new search.
“Copy All Devices to Project”	The device that is selected in the table is inserted into the device tree in the project. If nothing is selected, then all scanned devices are shown.

NOTICE!

If you insert devices, which are available in the device tree, to the device tree with *“Copy All Devices to Project”*, then the following should be noted. The data of the *“Process Data”* and *“<...> I/O Mapping”* tabs of the existing devices can be overwritten with the data of the recently inserted devices.

Table 128: “Configured Devices”

This part of the dialog is visible only when you select the *“Show differences to project”* option. Differences between the scanned and configured devices are color-coded. Devices displayed in green are identical on both sides. Devices displayed in red are available only in the view of the scanned or configured devices.	
	If you have selected a device in both views, then the scanned devices are inserted above the selected configured device.
	If you have selected a device in both views, then the scanned devices are inserted below the selected configured device.
	If you have selected a device in both views, then the configured devices are replaced by the selected scanned device.
	All scanned devices are copied to the project.
	Deletes the selected configure device.

The dialogs for the scan differ depending on the type of device. See the help pages for the respective device editor.

Command 'Update Device'

Function: Like the command *“Add Device”*, this command opens the dialog box *“Add Device”* for the selection of a device object. This object is inserted in the device tree in place of the currently selected object.

Call: Context menu of a device object in the device tree.

Requirement: An object is selected in the device tree below which a device object can be inserted.

With this command you can insert either a different version of a device or a different type of device in place of the previous one.

The symbolic device name used in the device tree is retained, but the device type specified in parentheses behind it changes if a different type has been selected. Thus if only the device version is changed, the object entry appears unchanged.

If the device type does not change, the configuration tree indented below the device entry concerned is retained. In this case the configuration settings also remain the same. Inconsistencies in the configuration resulting from the device update are reported by CODESYS at the next compilation of the application. This also concerns implicitly inserted libraries, which CODESYS does not remove accordingly during a device update.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*
- *Chapter 1.4.1.20.3.4.3 “Command 'Insert Device'” on page 1002*

Command 'Acknowledge Diagnosis', 'Acknowledge Diagnosis for Subtree'

Function: The command acknowledges a diagnosis message.

Call: Context menu of a device object in the device tree

Requirement: The project is in online mode.

The *“Acknowledge diagnosis”* command acknowledges the diagnosis messages of an individual device. The *“Acknowledge Diagnosis for Subtree”* command also acknowledges the diagnosis messages of all subordinate devices. The diagnosis message of a pending malfunction is indicated by a red exclamation mark at the device object. The diagnosis message of a corrected malfunction is indicated by a gray exclamation mark.

Command 'Edit Object'

Function: This command opens the object in its editor.

Call: Main menu *“Project”*, context menu.

Requirement: An object is selected in the device tree or in the *“POUs”* view.

Command 'Edit Object with'

Function: When multiple objects are available for an object, this command opens a dialog box for selecting an editor.

If only one editor is available for an object, then this command opens the object in that editor.

Call: Main menu *“Project”* or shortcut menu (right-click)

Requirement: An object is selected in the device tree or in the *“POUs”* view.

In the standard installation of CODESYS, there is no object that has multiple available editors.

Command 'Check integrity'

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

Call: Main menu *“Project”*, Context menu.

Requirement: A project is open.

Command 'Edit Object (Offline)'

Function: The command opens the object offline in the editor.

Call: Main menu *“Project”*, Context menu

Requirement: The application is in online mode. An object is selected in the device tree or in the *“POUs”* view.

The command allows you to edit objects in online mode. After editing you transfer the changes to the controller by use of the command *“Online ➔ Online Change”* or *“Online ➔ Load”*.

See also

- *Chapter 1.4.1.20.3.6.6 “Command 'Online Change'” on page 1033*
- *Chapter 1.4.1.20.3.6.5 “Command 'Load'” on page 1032*

Command 'Set Active Application'

Function: This command sets the selected application as the active application.

Call: Main menu *“Project”*, or right-click the *“Application”* object.

Requirement: The project has at least two applications. The selected application is not active.

Online actions apply only to the active application. The name of an active application is displayed in bold typeface in the device tree.

Command 'Project information'

Symbol: 

Function: This command opens the dialog box *“Project Information”*.

Call: Main menu*“Project”*

When you execute the command in the project for the first time, CODESYS creates the *“Project Information”* object.

See also

- *Chapter 1.4.1.2.3.1 “Retrieving and Editing Project Information” on page 191*
- *Chapter 1.4.1.2.3.2 “Making project settings” on page 193*
- *Chapter 1.4.1.20.2.21 “Object 'Project Information'” on page 919*

Command 'Project Settings'

Symbol: 

Function: This command opens the *“Project Settings”* dialog box.

Call: *“Project”* menu or double-click on the object *“Project Settings”* in the *“POUs”* view

Requirement: A project is open.

See also

- *Chapter 1.4.1.2.3.2 “Making project settings” on page 193*
- *Chapter 1.4.1.20.4.11 “Dialog 'Project Settings'” on page 1170*
- *Chapter 1.4.1.20.2.20 “Object 'Project Settings'” on page 918*

Command 'Project Environment'

Function: This command opens the *“Project Environment”* dialog box.

Call: *“Project”* menu

Requirement: A project is open.

this command is for checking the currentness of software and files integrated in the project and enables them to be updated.

See also

- *Chapter 1.4.1.20.4.12.1 “Dialog 'Project Environment' – 'Library Versions'” on page 1182*
- *Chapter 1.4.1.20.4.12.6 “Dialog 'Project Environment' – 'C Code Modules'” on page 1184*
- *Chapter 1.4.1.20.4.12.2 “Dialog 'Project Environment' - 'Compiler Version'” on page 1182*
- *Chapter 1.4.1.20.4.12.3 “Dialog 'Project Environment' - 'Device Versions'” on page 1183*
- *Chapter 1.4.1.20.4.12.4 “Dialog 'Project Environment' – 'Visualization Profile'” on page 1183*
- *Chapter 1.4.1.20.4.12.5 “Dialog 'Project Environment' – 'Visualization Styles'” on page 1184*
- *Chapter 1.4.1.20.4.12.7 “Dialog 'Project Environment' – 'Visualization Symbols'” on page 1185*

Command 'Project Localization' - 'Create Localization Template'

Function: This command opens the *“Create Localization Template”* dialog. Define here which information should be exported from the project to a translation template (`*.pot` file).

Call: Menu bar: *“Project ➔ Project Localization”*.

Requirement: A project is open.

Dialog 'Create Localization Template'

This dialog is used for selecting the textual information that should be used in the localization template.

Table 129: "Include the Following Information"

"Names"	Texts, such as dialog captions and object names in the device tree
"Identifier"	Variable identifier (example: `Counter`)
" Strings"	Example: `'count'` in the following declaration: `strVar: STRING := 'count';`
"Comments"	Comment texts in the POUs

"Position information"	Selection of which positions of the selected text categories in the project should be included in the translation file. The position information is located in the first line(s) of a segment for a translation. Example:
	`#: D:\Proj1.project\Project_Settings:1`
	`msgid "Project settings"`
	`msgstr ""`
	● *"All"*: All detected positions of the text are listed.
	● *"First appearance"*: In the translation file, the position is included in the project where the text to be translated appears for the first time.
	● *"None"*

"Generate"	This button opens the dialog for saving a file. The translation template is created in a text file of type `*.pot` (portable object template). Each further generation creates a completely new template file.

See also

- Help about CODESYS Visualization: Multi-language capability

Command 'Project Localization' - 'Manage Localizations'

Function: This command opens the *"Manage localizations"* dialog. Select the desired localization language in the dialog or the original version of the project. You can still accept the localization files `*.<language>.po` into the project or remove them.

Call: Menu bar: *"Project ➔ Project localization"*.

Requirement: A project is open.

Dialog 'Manage localizations'

"Available Localizations"	List of the localization files available in the project. Example:
	`proj1-de.po`
	`proj1-en.po`
	`<original version>`
	The original version is always available. The project can be edited only in the original version.

"Add"	This button opens the dialog for selecting an additional `po` file from the file system.
"Remove"	This button removes the `po` file, which is selected on the left side, from the project.
"Default localization"	☑: The selected localization is for the default localization. The entry is display in bold.

“Switch Localization”	Use this button to switch to the selected localization.
“OK”	The project is displayed in the language that is provided by the file selected below the files. If you select *“<original version>”*, then the project appears in the editable non-localized version.

See also

- *Chapter 1.4.5.6 “Setting Up Multiple Languages” on page 1286*

Command 'Project Localization' - 'Toggle Localization'

Symbol: 

Function: This command switches between the currently set project localization and the <original version>.

Call: Menu bar: *“Project ➔ Project Localization”*; button in the *“Manage Localizations”* dialog;  button on the toolbar.

Requirement: A project is open. A default localization for the project is defined in the *“Manage Localizations”* dialog.

See also

- Help about CODESYS Visualization: Multi-language capability
- *Chapter 1.4.1.20.3.4.17 “Command 'Project Localization' - 'Manage Localizations'” on page 1008*

Command 'Document'

Symbol: 

Function: This command opens the *“Document Project”* dialog box, where you can define the project documentation. This includes the selection of objects in the open project that you want to print.

Call: Main menu *“Project”*

Table 130: “Document Project” dialog box

“Please select the objects which are to be printed”	Project tree view In this view, you can select or clear objects for printing. All objects are selected by default.
“Title page”	CODESYS creates a title page named "Project Documentation" with the following information: ● File: project file name ● Date: Creation date of the project documentation ● Profile: CODESYS profile of the project
“Table of contents”	CODESYS creates a table of contents for the project documentation.
“Preview”	CODESYS creates and opens a print preview of the project documentation.
“Select”	CODESYS opens a drop-down list of all or single object types for the project documentation.
“Deselect”	CODESYS opens a drop-down list of all or single object types that should be excluded from the project documentation.
“OK”	The *“Print”* dialog box opens.

See also

- *Chapter 1.4.1.20.4.11.6 “Dialog 'Project Settings' - 'Page Setup'” on page 1175*
- *Chapter 1.4.1.20.3.1.12 “Command 'Print'” on page 963*

Command 'Compare objects'

Function: To compare similar objects within a project.

Call: Main menu *“Project”*, Context menu.

Requirement: Both projects have to be open.

Chapter 1.2.13.4 “Comparing objects” on page 59

Command 'Compare'

Symbol: 

Function: This command opens the *“Project Comparison”* dialog. In this dialog, you define the reference project to compare with the current project. You configure the comparison process by means of options. When the dialog is exited, the comparison starts and the result is shown in the view *“Project Compare - Differences”*.

Call: Menu bar: *“Project ➔ Compare”*.

Requirement: A project is open.

See also

- *Chapter 1.4.1.4 “Comparing projects” on page 195*
- *Chapter 1.4.1.20.3.4.22 “Command 'Commit Accepted Changes'” on page 1014*

Dialog 'Project Comparison'

Table 131: “Compare the currently open project with:”

“Project on disk”	Path of the reference project on the file system.
“Project in a source control database”	*“Host”*: Name of the host where the source code management is located. *“Port”*: Number of the port for connecting to the source code management. *“Location”*: Path of the reference project. Requirement: The project is linked to source code management (for example, Professional Version Control).

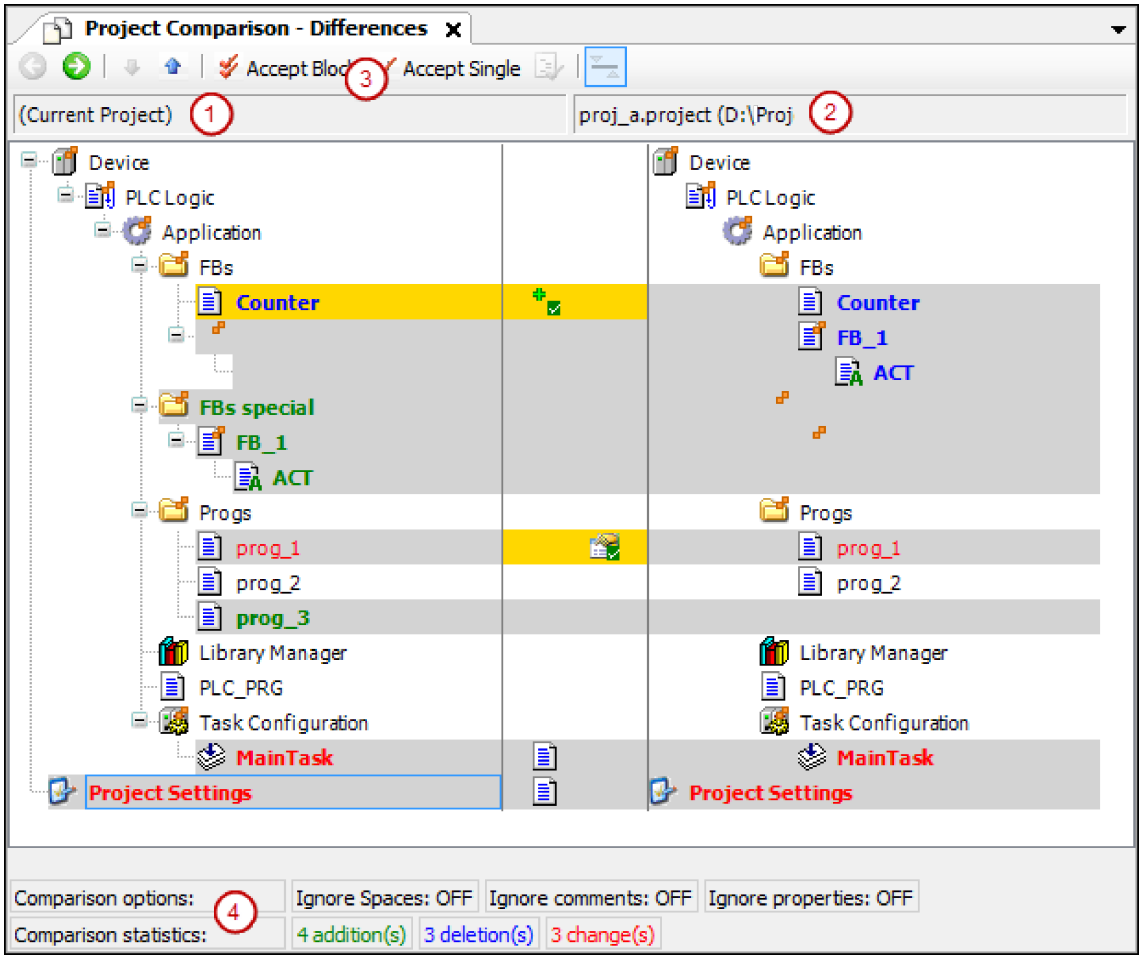
Table 132: “Compare Options”

“Ignore whitespace”	☑: Whitespace differences between the current project and the reference project are ignored.
“Ignore comments”	☑: Comments in the programming code are excluded from the comparison.
“Ignore properties”	☑: Object properties are excluded from the comparison.

“OK”	Starts the project compare and displays the result in the view *“Project compare - Differences”*.

View 'Project Comparison' - 'Differences'

The project compare view opens when you click *"OK"* to close the *"Project Compare"* dialog.




```
(1) Object tree of the current project
(2) Object tree of the reference project
(3) Command 'Accept Block', command 'Accept Single'
(4) Compare options, configured in 'Project Compare' dialog
    Compare statistics: added, deleted, and changed objects
```

Table 133: Toolbar

	Switches to the detailed compare view *"Project Comparison' - '<object name> Differences"* for the object selected in the tree. Alternative: Double-click the object.
	Selects the next bottom object in the device tree where differences were detected.
	Selects the next top object in the device tree where differences were detected.
"Accept Block"	The block (selected object with all subordinate objects and units) is selected for acceptance from the reference block to the current block. Repeated clicking of *"Accept Block"* undoes the effects of its last change.
"Accept Single"	The object is selected in the current object for acceptance from the reference line.
	Requirement: The properties, access rights, or contents of the objects selected in the object tree are different. Opens the *"Accept"* dialog.

Table 134: Display of differences with colors, and symbols

Black font	Objects are identical.
Object name with	Child objects of the object are different
Gray highlight	Objects are different.
Gray highlight + bold blue font	Object is only in the reference project.
Gray highlight + bold green font	Object is only in the open project (not in reference project).
Gray highlight + red font +	Object has different properties.
Gray highlight + red font +	Access rights of object and reference object are different.
Gray highlight + bold red font +	Implementation of objects is different. Double-click the line to display the object-specific compare view.
Yellow highlight	Object is activated for acceptance.
Yellow highlight +	Adding the reference object to the open project is activated.
Yellow highlight +	Deleting the object (in the open project) is activated.
Yellow highlight +	Acceptance of the properties of the reference project is activated.
Yellow highlight + red font +	Acceptance of the access rights of the reference project is activated.
Gray highlight + bold red font +	Acceptance of the implementation of the reference project is activated.
"Compare options"	Defined comparison options in the *"Project Comparison"* dialog.
"Compare statistics"	Number of additions, deletions, and changes in the current project, as compared to the reference project. *"Change"* means differences of an object available in both projects.
×	The dialog prompt opens: *"Do you want to commit the changes which you made in the diff view?"* *"Yes"*: The contents, properties, or access rights of the objects highlighted in yellow are modified in the project. Now they correspond to the reference project. Then the project compare view is closed completely.

View 'Project Comparison' - '<object name> Differences'

Function: Detail compare view

Call in the project compare view:

- Select an object that is identified as having different contents which you need to view in detail. Click .
- Double-click the object.

Table 135: Toolbar

	Switch back to the project compare view.
	Selects the next line below in the code where differences were detected.

	Selects the next line above in the code where differences were detected.
"Accept Block"[2]	The block (with all subordinate lines) is selected for acceptance of the reference blocks into the current project. A block in the detailed compare view consists of the unit where the cursor is located and all corresponding units that have the same difference markers. A unit is a line, network, or element. Subsequent lines of a line are examples of corresponding units. Repeated clicking of *"Accept Block"* undoes the effects of its last change.
"Accept Single"	The line is selected in the current object for acceptance of the reference line.
	Switches between the default display where different units (lines, networks, elements) are displayed in red and another display: The units are displayed as recently added in the open project. In the reference project, they are displayed as deleted. Available within the detailed compare view only. Note: Depending on the display, detected differences in the statistics are counted as changed, inserted, or deleted.

Table 136: Display of differences with colors, and symbols

Black font	Objects are identical.
Gray highlight + bold blue font	Code is only in the reference project.
Gray highlight + bold green font	Code is only in the current project (not in reference project).
Yellow highlight	The object is activated for acceptance.
×	The dialog prompt opens: *"Do you want to commit the changes which you made in the diff view?"* *"Yes"*: The code highlighted in yellow is accepted into the project. The code corresponds to the reference project. Then the detailed view is closed and the project view is displayed. You can continue working with project compare.

Dialog 'Accept'

Table 137: "Which meta data should be accepted?"

"Access rights"	☑: Access rights that are selected for acceptance.
"Accepted groups"	Grouping with access rights accepted by the reference project. A group is accepted if it is present in both projects with different access rights. Example: `Group_A`
"Unaccepted groups (missing in a project) "	The group is not accepted if it is not present in one of the two projects.
"Properties"	☑: Properties activated for accept Requirement: The properties of the reference object and object are different.

"OK"	Settings are accepted.

Command 'Commit Accepted Changes'

Symbol:

Function: This command commits the accepted differences from the project comparison to the current project.

Call: *"Project ➔ Commit Accepted Changes"*.

Requirement: Changes from the project comparison have been accepted.

> *The changes are only copied to the project. This command does not save them to the hard disk.*

See also

- *Chapter 1.4.1.4.2 "Opening the detailed compare view" on page 196*

Command 'Map pool devices'

Symbol:

Function: Maps imported devices from the device pool to already configured devices below a PLC.

Call: Main menu *"Project"*, Context menu.

Requirement: A project is open.

Chapter 1.8.1.1.7 "Arrange or map devices imported to the device pool" on page 4115

Command 'Export'

Function: This command opens a dialog box for exporting objects from a project to an XML file.

Call: Menu bar: *"Project"*.

Dialog 'Export' This dialog box lists all objects from the device tree, POU tree, and module tree that CODESYS can export.

One file per subtree	☑: CODESYS generates a separate export file for each subtree that is located directly under the root node and includes selected files. ☐: CODESYS generates one export file for all selected objects.
"Saved version"	This version should correspond to the target version where the export file will later be imported. If the current project contains plug-ins or add-ons that are not available in the selected memory format (profile), then the *"Extend Profile"* dialog box opens. In this dialog box, the selected profile can be extended with the add-ons.

See also

- *Chapter 1.4.1.20.3.1.5 "Command 'Save Project as'" on page 958*
- *Chapter 1.4.1.20.3.4.25 "Command 'Import'" on page 1015*
- *Chapter 1.4.1.3.1 "Exporting and importing projects" on page 193*

Command 'Import'

Function: This command opens a dialog box for importing objects from an XML file.

Call: Menu *“Project”*

Requirement: A project is open.

Dialog box 'Import'

The dialog box lists all objects from the export file that CODESYS can import at this point.

“Currently selected target objects”	Object that is selected in the Device tree
“Insertable items”	Displays all objects of the export file that CODESYS can insert below the selected object.
“Show contents”	Displays the contents of the export file in a tree structure

Command 'Export PLCopenXML'

Function: This command opens a dialog box for exporting objects from a project into an XML file in the PLCopen format.

Call: Menu *“Project”*

Dialog box 'Export PLCopenXML'

The dialog box lists all objects from the Device tree that CODESYS can export into an XML file in accordance with the PLCopen format.

The PLCopenXML scheme does not permit VAR_GLOBAL and VAR_GLOBAL CONSTANT POUs to be in the same variable list. Therefore, if you wish to export both, you must first divide the variables into two separate variable lists.

See also

- *Chapter 1.4.1.20.3.4.27 “Command 'Import PLCopenXML'” on page 1015*

Command 'Import PLCopenXML'

Function: This command opens a dialog box for importing objects from an XML file in PLCopen format.

Call: Menu *“Project”*

Requirement: A project is open.

Dialog box 'Import PLCopenXML'

The dialog box lists all objects from the PLCopen export file that CODESYS can import at this point.

“Currently selected target object”	Object that is selected in the Device tree
“Insertable items”	Displays all objects of the export file that CODESYS can insert below the selected object.

The PLCopenXML scheme does not permit VAR_GLOBAL and VAR_GLOBAL CONSTANT POUs to be in the same variable list. Therefore, if you wish to export both, the variables must first be divided into two separate variable lists.

Command 'User management' – 'Log in User'

Symbol: 

This command opens the dialog box *"Login"*. Here you specify the project that you wish to edit and enter the login data for a user account with the corresponding rights. In addition, you can open the password manager from this dialog box.

The command is available in the menu *"Project ➔ User Management"*.

See also

- *Chapter 1.4.1.5.6 "Logging in via User Account and Password Manager" on page 205*

Command 'User management' – 'Log out User'

Symbol: 

The user currently logged in to the project is logged out again with this command. This takes place without a dialog box or message, unless no user is currently logged in.

The command is available in the menu *"Project ➔ User Management"*.

If the user is currently logged in to several projects or to libraries integrated in them (it does not have to be the same user account), then the dialog box *"Logout"* opens, in which the specific project or library project can be selected from which the current user is to be logged out.

The status bar always displays the user who is currently logged into the project.

A double-click on the field *"Current user"* in the status bar enables quick access to the *"Login"* or *"Logout"* dialog box.

See also

- *Chapter 1.4.1.5.6 "Logging in via User Account and Password Manager" on page 205*
- *Chapter 1.4.1.20.3.4.28 "Command 'User management' – 'Log in User'" on page 1016*

Command 'User management' – 'Rights…'

This command opens the dialog box *"Rights"*, in which you define the actions that may be carried out, the user groups that may carry them out and the project objects on which they may be carried out.

The command is available in the menu *"Project ➔ User Management"*.

See also

- *Chapter 1.4.1.5.5 "Protecting Objects in the Project by Access Rights" on page 204*

Command 'Insert Device'

Function: Like the command *“Add Device”*, this command opens a dialog box *“Insert Device”* for the selection of a device object. This object is inserted in the device tree at the same level as the currently selected object.

Call: Context menu of a device object in the device tree.

Requirement: An object is selected in the device tree below which a device object can be inserted at the same level.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*
- *Chapter 1.4.1.20.3.4.3 “Command 'Insert Device'” on page 1002*

Command 'Generate EtherCAT XML'

The command is not integrated in the standard main menu. You can add it via the dialog box “Tools ➔ Customize” from the category “Devices”.

Function: This command opens the standard dialog box for saving a file in the local file system. You can define a name and a storage location for an xml file, in which CODESYS is to store the EtherCAT configuration of the EtherCAT master currently selected in the device tree. This may be necessary in order to operate an external EtherCAT stack.

Call: Context menu of an EtherCAT master device object in the device tree.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*

Command 'Generate Sercos SCI XML'

The command is not integrated in the standard menu. You can add it via the dialog box “Tools ➔ Customize” from the category “Devices”.

Function: This command opens the standard dialog box for saving a file in the local file system. You can define a name and a location for an xml file in which CODESYS then stores the configuration data of the sercos master currently selected in the device tree. This may be necessary in order to operate an external sercos stack.

Call: Context menu of a sercos master device object in the device tree.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*
- *Chapter 1.4.1.1.2.1 “Customizing menus” on page 180*

Command 'Disable Device' – 'Enable Device'

Function: This command switches back and forth between the enabled (activated) and disabled (deactivated) states of a device in the bus system.

Call: Context menu of a device object in the device tree.

Requirement: The project is in offline mode. The bus driver must support the function.

A disabled device is not taken into account and is not addressed. Note that with some bus systems the deactivation of a node can lead to the master stopping.

The entry of a disabled device in the tree appears in light-gray lettering. When logging in, disabled devices are additionally marked with a red triangle.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*

Command 'Edit I/O Mapping'

Function: This command opens the *“Edit I/O Mapping”* dialog box. This displays all I/O mappings of the currently selected device object, including I/O mappings of all additional device objects that are inserted in the device tree below this object.

Call: Context menu of a device object in the device tree.

Dialog box 'Edit I/O mapping'

You can edit the I/O mapping in this dialog box in exactly the same way as in the dialog box *“I/O mapping”* of the individual device editors. The respective other dialog boxes are directly updated accordingly.

“Search”	Input field for a search string for the mapping table. The search results are marked in yellow.
“Filter”	Drop-down list for filtering I/O assignments displayed listed in the mapping table: ● *“Show all”* ● *“Show outputs only”* ● *“Show inputs only”* ● *“Show unmapped variables only”* ● *“Show mapped variables only”* ● *“Show mappings to existing variables only”* ● *“ Show mappings to new variables only”*

In the context menu you will find among other things the following commands:

“Export Mappings to CSV”: Stores the mappings of a device and its sub-devices in an external file. To do this you select the device in the device tree or in the mapping list.

“Import Mappings from CSV”: Inserts mappings from a file created beforehand by export.

See also

- *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*
- *Chapter 1.4.1.20.3.4.35 “Command 'Edit I/O Mapping'” on page 1018*
- *Chapter 1.4.1.20.3.4.37 “Command 'Export Mappings to CSV'” on page 1019*
- *Chapter 1.4.1.20.3.4.36 “Command 'Import Mappings from CSV'” on page 1018*

Command 'Import Mappings from CSV'

Function: The command opens the default dialog for opening a file in the local file system. The filter is set to the file format `CSV` in order to import the I/O mapping configuration of a device from the file which was exported previously by means of the *“Export Mappings to CSV”* command. CODESYS writes the configuration to the selected device.

Call: Context menu of a device object in the *“Devices”* view.

Requirement: A project is open with a device and an I/O mapping configuration. The device matches the exported `CSV` file.

NOTICE!

I/O mapping configurations are stored in `CSV` files with the semicolon separator. These files can be edited manually. If the files are edited manually, then it is imperative that this format is retained in order to import successfully. Note: The entries of the file to the I/O mapping of the device are assigned by the device name and the parameter name. Parameter names that are not unique are numbered sequentially in this file (@<n>).

Fields without contents in the `CSV` file are ignored at import. To remove an existing entry in the I/O mapping by importing, you have to add a space in the respective field in the `CSV` file.

See also

- *Chapter 1.4.1.20.3.4.37 “Command 'Export Mappings to CSV'” on page 1019*
- *Chapter 1.4.1.7.1 “Configuring Devices and I/O Mapping” on page 213*

Command 'Export Mappings to CSV'

Function: The command opens the default dialog for saving a file to the local file system. The filter is set to file format `CSV`. After specifying a name and a location, CODESYS stores the I/O mapping configuration in a `CSV` file with the semicolon separator.

Call: Context menu of a device object in the *“Devices”* view.

Requirement: A device object with an I/O mapping configuration is selected in the device tree.

Parameter names that are not unique are numbered sequentially in this file (@<n>).

See also

- *Chapter 1.4.1.20.3.4.36 “Command 'Import Mappings from CSV'” on page 1018*
- *Chapter 1.4.1.7.1 “Configuring Devices and I/O Mapping” on page 213*

Command 'Read PLC Parameter File to Configuration'

Function: This command reads the configuration file `IoConfig.par` of the PLC and stores the values in the project. Such a file is created if the parameters of the PLC have been changed by another device, for example via a visualization. Then these parameters are changed only in the memory of the PLC, but not in the configuration of the project.

Call: Context menu of the PLC device object

Requirement: You have made the command available using the dialog in *“Tools ➔ Customize”*.

Command 'Online Config Mode'

Function: This command is for switching the online configuration mode on and off. At switch-on it establishes a connection to the PLC and loads an implicitly created application *“HiddenOnlineConfigModeApp”* to the PLC. Depending on the device, CODESYS goes into simple online configuration mode or a dialog box appears for selecting between simple and advanced online configuration mode.

Call: Context menu of the PLC object in the device tree

Requirement: The communication settings for the PLC device are correctly set.

Simple online configuration mode:

This command creates the implicit application `HiddenOnlineConfigModeApp` and loads it to the controller. The application automatically initializes all inputs and outputs of the controller once. After that you can access the I/Os as follows:

- Read I/Os
- Write outputs
- Diagnosis (in the device tree and on the *"Status"* tab of the device editor)
- Scan (of the current hardware)
- Interactive online functions, if supported (for example, writing asynchronous messages)

Advanced online configuration mode (parameter mode):

If there are already applications on the PLC and the controller supports it, the command first opens the dialog box *"Devices"*, which displays the applications existing on the controller. From this dialog you can connect via the button *"Parameter mode"* to the PLC and then access the values of the device parameters without having to log in with a real application.

Writing and forcing in the I/O mapping

In online configuration mode the writing and forcing of values on the "I/O Mapping" tab works differently to the way it works in real online mode. The outputs are written immediately after insertion into the table. There is no "Prepared Value" column; instead, the initial values can be changed directly after a double-click on the column "Current Value".

Dialog box 'Config application mode'

This dialog box appears after the command *"Online Config Mode"* if the device supports the advanced online configuration mode and there are already real applications on the controller.

"Parameter mode"	The controller configuration in the project is compared with that on the device. If they correspond, CODESYS establishes a connection to the PLC. Unlike the simple online configuration mode it permits the reading and – if supported by the driver – the writing of parameters in the generic device editor. The applications already loaded to the device remain unchanged in this case!
"Config application mode"	CODESYS switches to the 'simple online configuration mode'.

Command 'Runtime licensing'

Symbol: 

Function: Management of runtime licenses on the PLC. The use of some libraries and devices require the PLC to have a runtime license.

Call: Main menu *"Project"*, Context menu. Displayed only offline.

Requirement: A project is open. Log-in required for managing runtime licenses without need for memory card.

The license status of a PLC can be displayed at any time ↳ *Chapter 1.6.6.2.2.5 "View license information" on page 3672*.

Menu 'Build'

Command 'Generate Code'

Symbol  ; shortcut: *[F11]*

Function: The command starts the code generation for the active application.

Call: Menu bar: *“Build”*

When generating code with this command, code is generated as when downloading the application to the PLC, but the code is not transferred to the PLC. At this time, other source code tests are performed As a result, you can check the code for bugs that were not detected by the compiler and for fixing any bugs before the code is used in online mode.

See also

- *Chapter 1.4.1.10.4 “Generating Application Code” on page 389*

Command 'Clean'

Function: This command deletes the build information for the active application.

Call: Main menu *“Build”*.

During the last download, the build information was created and saved to a file (*.compileinfo).

After a cleaning process, an online change is no longer possible for the affected application. The application must be fully downloaded to the controller again.

See also

- *Chapter 1.4.1.20.3.5.3 “Command 'Clean All'” on page 1021*

Command 'Clean All'

Function: This command deletes the build information for all applications in the project.

Call: Main menu *“Build”*.

During the last download, the build information was created in the local file system and saved to a file (*.compileinfo).

This command requires a download before another login An online change is no longer possible. As compared to the *“Clean”* command (only the active application), CODESYS regenerates the language model for all objects, which is very time-consuming.

NOTICE!
Reconsider carefully whether or not executing this command is really necessary. If you only want to rebuild and download the active application, then execute the *“Clean”* command.

See also

- *Chapter 1.4.1.20.3.5.2 “Command 'Clean'” on page 1021*

Command 'Build'

Function: The command starts the build operation for the active application.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog from *“Tools ➔ Customize”* (command category *“Build”*).

During the build operation, CODESYS performs a syntactic validation of all objects in the application. However, code is not generated like at log in to the target system or download of the application. The build operation is always performed automatically when you log in with a changed program.

When the check is complete, CODESYS displays any error messages or warnings in the message view (*“Build”* category).

If the program has not been changed since it was compiled without errors the last time, then it is not recompiled. The message "The application is current" appears. If the syntactic validation is repeated, then you must execute the *“Rebuild”* command.

See also

- *Chapter 1.4.1.20.3.5.5 “Command 'Rebuild'” on page 1022*

Command 'Rebuild'

Function: The command starts the build operation for the active application, even if the last build contained errors.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog from *“Tools ➔ Customize”* (command category *“Build”*).

See also

- *Chapter 1.4.1.20.3.5.4 “Command 'Build'” on page 1022*

Command 'Generate Runtime System Files'

Function: The command generates a C stub file and an M4 interface file from the current library project. These files are used as the basis for creating an external library file.

Call: Menu bar: *“Build”*

Requirement: A library project is open.

The command opens the *“Generate Files for Runtime System”* dialog.

"Output directory"	Directory where CODESYS creates the runtime system files. Click the ... button to open the default dialog for browsing the file system.
"Component names"	Name of the library project
"Which files do you want to create?"	
"M4 interface file"	☑: Interface file <project name>Itf.m4 with definitions. Example of m4 file:
"C stub file"	☑: Stub file for reprogramming the library in C. Example of stub file:
"Options"	

SysFileItf.m4 - Editor

Datei Bearbeiten Format Ansicht ?

```
/**
 * <interfacename>SysFile</interfacename>
 * <description></description>
 *
 * <copyright></copyright>
 */

SET_INTERFACE_NAME(`SysFile')

/** EXTERN LIB SECTION BEGIN **/

#ifdef __cplusplus
extern "C" {
#endif

/**
 * <description>FileStatus</description>
 */
typedef enum
{
        FS_OK = 0,
        FS_NO_FILE = 1,
        FS_ILLEGAL_POS = 2,
        FS_FULL = 3,
        FS_EMPTY = 4,
        FS_BUFFER_FULL = 5
} FileStatus;

/**
 *      ANSI C File Access Modes
 * *    AM_READ                          Open an existing file with Rea
 * *    AM_WRITE                         Create new file with Write acc
 * *    AM_APPEND                        Open an existing file with App
 * *    AM_READ_PLUS            Open an existing file with Read/Write
 * *    AM_WRITE_PLUS           Create new file with Read/Write access
 * *    AM_APPEND_PLUS          Open an existing file with Append (rea
 *
 */
typedef enum
{
        AM_READ = 0,
        AM_WRITE = 1,
        AM_APPEND = 2.
```

SysFile.c - Editor

Datei Bearbeiten Format Ansicht ?

```
#include "CmpStd.h"
#include "CmpErrors.h"
#include "CmpItf.h"
#include "SysFileDep.h"

void CDECL CDECL_EXT sysfilegetpath(sysfilegetpath_struct *p)
{

}

void CDECL CDECL_EXT sysfilerename(sysfilerename_struct *p)
{

}
```

“Export referenced types included in libraries”	☑: The referenced types are included in the export.
“Use original type names ”	☑: The type names of the library project are used.

Command 'Check all Pool Objects'

Symbol ; shortcut: *[F11]*

Function: The command starts a build operation (a syntax check) for all pool objects that are managed in the POU view and as a result are available throughout the project. First and foremost, this is useful when creating libraries.

Call: Menu bar: *“Build”*

Requirement: A library project is open.

> **NOTICE!**
> The command does not result in code generation. In addition, no file is created in the project directory with information about the build operation.

See also

- *Chapter 1.4.1.20.4.14.1 “Dialog 'Customize' - 'Menu'” on page 1206*
- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*

Command 'Generate Code for Active Application'

Function: The command generates the code for the application of a library project.

Call: Menu bar: *“Build”*

Requirement: The project contains an application.

- A library project is open.
- The library project contains an application.

When generating code with this command, code is generated as when downloading the application to the PLC, but the code is not transferred to the PLC. At this time, other source code tests are performed As a result, you can check the code for bugs that were not detected by the compiler and for fixing any bugs before the code is used in online mode.

See also

- *Chapter 1.4.1.20.3.5.6 “Command 'Generate Runtime System Files'” on page 1022*

Command 'Check All Application Objects'

Function: This command starts a build operation for all objects of the active application, even for the POUs that are not used by the application. After the build operation, the errors that were found in the unused objects are also displayed in the message window.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog from *“Tools ➔ Customize”* (command category *“Build”*).

Requirement: An application of the open project is active.

> **NOTICE!**
> The command does not result in code generation. In addition, no file is created in the project directory with information about the build operation.

See also

- *Chapter 1.4.1.20.4.14.1 “Dialog 'Customize' - 'Menu'” on page 1206*

Command 'Check Library Compatibility'

Function: The command triggers a check whether the currently opened library project is compatible with the last installed version of this library (next lower version number) .

Call: By default the command is not available in any menu. You can add it to a menu by using the *“Tools ➔ Customize”* dialog, command category *“Build”*.

Requirement: A library project is opened.

The check regards differences in the implemented interfaces of a method. So, after the check you will get displayed error messages in the messages window in the following cases:

- Adding or removing inputs or outputs of function blocks, functions or methods
- Changing the data type of inputs or outputs
- Modifying the implemented interfaces of a method

See also

- *Chapter 1.4.1.20.4.14.1 “Dialog 'Customize' - 'Menu'” on page 1206*
- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*

Command 'C Integration' - 'Update C Sources'

Function: this command opens the dialog *“Update C Sources”* for updating the objects in the project that have changed in the source directory on the disk.

Call: Menu bar: *“Build”*; context menu.

Requirement: An object *“C Code Module”* or *“C Implemented Library”* is selected. When adding the C-code module in the dialog *“Add C Code Module”*, you have activated the option *“Check folder for source code changes”*.

“File”	File that has changed on the disk.
“Action”	Action that is executed in CODESYS if you click *“Update”*.
“Update options”	● *“Remove IEC interfaces due to changed header files”* ● *“Export source files to the monitored project folder”*
“Refresh”	CODESYS updates the listed files.

See also

- *Chapter 1.4.1.20.3.5.12 “Command 'C Integration – Open in IDE'” on page 1025*
- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Command 'C Integration – Open in IDE'

Function: The command opens the *“C Code Module”* in the associated IDE (Integrated Development Environment).

Call: Main menu *“Create”*, context menu

Requirement: You have opened an object *“C Code Module”* and the associated IDE is not opened.

If the IDE is closed, CODESYS checks whether the files have been changed and, in such a case, a dialog box appears for confirming the update of the C-code module in CODESYS.

See also

- *Chapter 1.4.1.20.4.10.5 “Dialog 'Properties' – 'Build' (C-integration)” on page 1160*
- *Chapter 1.4.1.20.3.5.11 “Command 'C Integration' - 'Update C Sources'” on page 1025*
- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Command 'C Integration' - 'Export C sSurces'

Function: The command exports all C-code files of a C-code module and saves them in the folder that you select in the dialog *“Find Folder”*.

Call: Menu bar: *“Build”*; context menu.

Requirement: A C code module is selected in the device tree.

See also

- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Command 'C Integration – Create Stub Implementation in C'

Function: This command creates C-stubs for the selected POU and stores them in the *“Extensions”* folder in the objects *“iec_external.c”* and *“iec_external.h”*.

Call: Main menu *“Build”*, context menu

Requirement: A POU that is inserted under the object of the type *“C Code Module”* is selected in the device tree. The application has been compiled without errors.

See also

- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Command 'Create IEC Interface'

Function: The command creates corresponding IEC objects from the selected file with the format `*.h` or `*.hhp` and stores these IEC objects in the folder *“IEC interface”*.

Call: Main menu *“Build”*, context menu

Requirement: You have selected an imported C-code file of the format `*.h` or `*.hpp` in the device tree below the object *“C Code Module”*.

If you select the command and the header file is free of errors, the dialog box *“C Functions”* opens with a list of the functions of the file that is to be exported.

Dialog box 'Create IEC interface'

“Function”	List of the functions You select the functions for which a corresponding IEC object is to be created.
“Import”	CODESYS generates corresponding IEC objects for the selected C-functions and stores them in the folder *“IEC interface”* below the object *“C Code Module”*.

See also

- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Command 'Generate Disassembly File'

Function: This command generates a disassembly file `<project name>.asm` from the current project and saves it in the file directory in the project folder.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog box from *"Tools ➔ Customize"* (command category *"Build"*).

See also

- *Chapter 1.4.1.20.4.14.1 "Dialog 'Customize' - 'Menu'" on page 1206*

Menu 'Online'

Command 'Choose Active Application'

Symbol: 

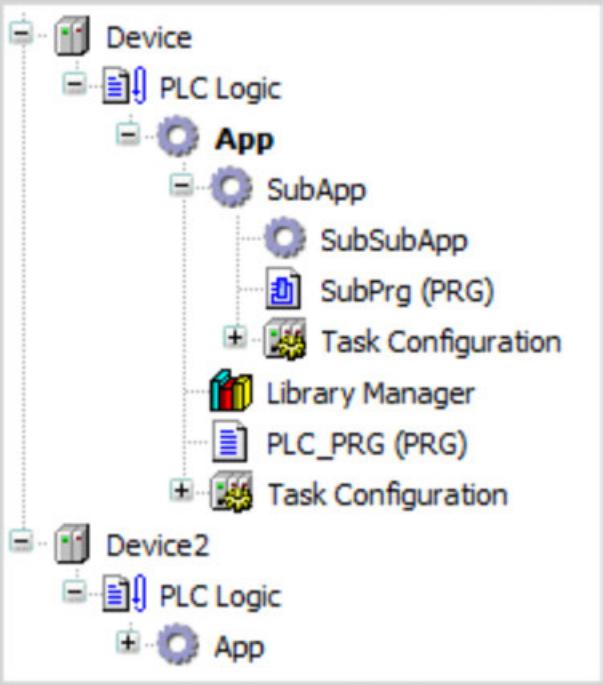
The command is implemented as a list box from which you can set an application active. By default, the list box is located on the toolbar.

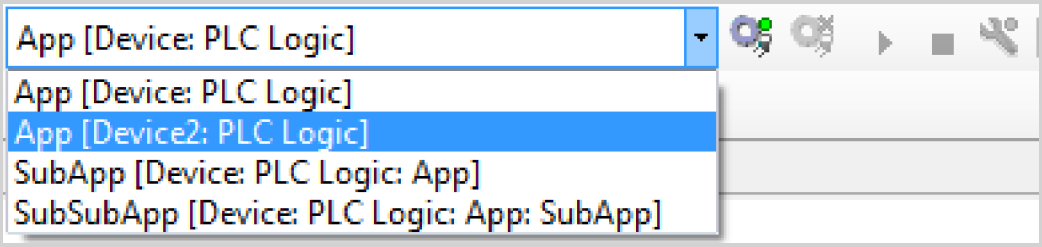
Function: The list box displays the currently active application with its device path.

Call: The list box contains all applications that are organized in the *"Devices"* view. By clicking an entry in the list box, you activate the selected application.

Requirement: The project has multiple applications.

Example





When you call commands in the “Build” oder “Online” menus, these commands are applied to the active application. This is displayed in the list box, and also displayed in bold in the device tree. In particular, this applies to the “Build ➔ Build” and “Online ➔ Login” commands.

You can also access these commands using the command icons on the toolbar where the list box is located. When the command icons are called, they are also applied to the active application.

However, if you call a command from the context menu of a device object in the device tree, then the command is applied to the corresponding object. For example, by calling , you can establish a connection to an application on the device which is not active.

See also

- *Chapter 1.4.1.20.3.4.12 “Command 'Set Active Application'” on page 1006*
- *Chapter 1.4.1.20.3.5.4 “Command 'Build'” on page 1022*
- *Chapter 1.4.1.20.3.6.2 “Command 'Login'” on page 1028*

Command 'Login'

Symbol: ; shortcut: *[Alt]+[F8]*.

Function: The command connects the application to the target system (PLC to simulated device) and starts the online mode.

Call: Menu bar: *“Online”*; context menu of an *“Application”* object

Requirement: The application contains no errors and the communication settings are configured.

A dialog prompt opens if the communication settings are incorrect. You can then switch directly to the *“Communication Settings”* of the PLC.

If you click *“Login”* from the online menu, then the currently active application is connected to the target system. If you choose this command from the context menu (right-click) while an application is selected in the device tree, then that application is logged in, even if it is not set as the active application.

If an online user management is configured on the target device, then you are prompted for user data when you log in. The *“Device User Login”* dialog opens for this.

CAUTION!

Check controller accessibility

For security reasons, controllers should not be accessible from the Internet or untrusted Networks under any circumstances! In particular, the TCP/IP programming ports (usually UDP-Ports 1740..1743 and TCP-Ports 1217 + 11740 or the controller specific ports) should not be accessible from the internet without protection. In case Internet access to the controller is needed, using a a safe mechanism is absolutely mandatory, such as VPN and password protection of the controller.

see also: *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

NOTICE!

If a safety controller is inserted below a controller, then this command can interrupt the communication connections **temporarily**.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, see the "Subordinate Safety Controllers" chapter.

Possible situations when logging in:

- A later version of the device description (than in the project) is on the PLC. A warning prompt is displayed with the option to cancel the process.
- The application does not exist on the PLC: You are prompted to confirm the download.
- The application is already on the PLC and has not been changed since the last download. The login continues without any more prompts.
- The application exists on the PLC, but it has been changed since the last download.
  You are prompted to select one of the following options:
  – Login with online change (Note the information about online changes in the help page "Command 'Online Change' ".)
  – Login with download
  – Login without any change

  The position also provides the option of updating the boot application on the PLC.
- An unknown version of the application exists on the PLC. CODESYS prompts you to replace it.

- A version of the application exists on the PLC and is running. CODESYS prompts you to log in anyway and overwrite the currently running application.
- The application on the PLC is currently halted at a breakpoint. You are logged out and the program has been changed: CODESYS prompts you with a warning that the PLC will be stopped completely if an online change or download occurs. This happens also if several tasks exist and the breakpoint affects only one of them.

Click *"Details"* in the dialogs above to open the *"Application Information"* dialog.

> *In CODESYS V3.5 SP17 and higher, only exactly one CODESYS instance can ever be logged in to an application of a controller. If a second CODESYS instance wants to log in to the same application of the same controller, then an error message is displayed.*

See also

- *Chapter 1.4.1.20.3.6.6 "Command 'Online Change'" on page 1033*
- *Chapter 1.4.1.9.5 "Subordinate safety controller" on page 378*

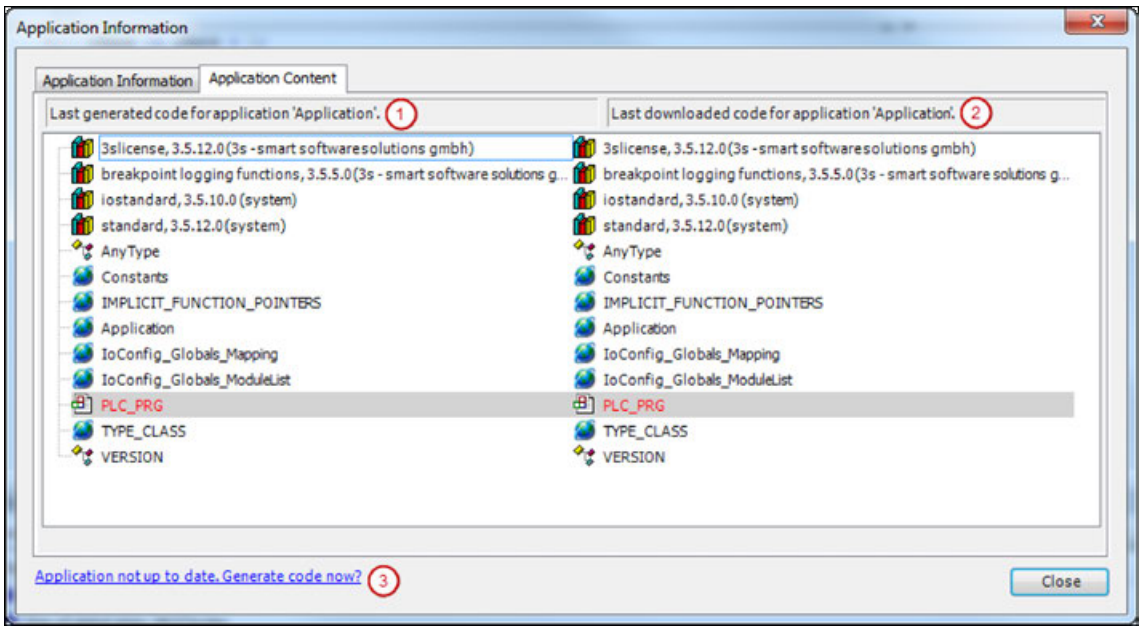
Dialog 'Application Information' (Details)

The dialog provides two tabs with comparative information about the application changed in the development system and its previous version currently located on the PLC. There are two tabs:

- *"Application information"*: The application properties of the *"Application in the IDE"* (Integrated Development Environment) are compared with those of the *"Application in the PLC"*: Project name, Last modification, IDE version, Author, Description. In addition, CODESYS shows the objects that have changed since the last download.
- *"Application contents"*: When the *"Download application info"* is selected, the contents of the applications on both the (1) development system and (2) PLC can be compared. The *"Download application info"* option is located on the *"Application Build Options"* tab of the application properties.

  If the code in the development system is not current, then (3) *"Application not up to date. Generate code now?"* appears at the bottom left of the dialog. Execute this command to update the application source code.

  This detailed information can help you to better assess the effects of login in the current situation and to make a decision about downloading the new application.



The comparison can also be displayed in the device editor (“Applications” tab) by clicking “Content”.

See also

- *Chapter 1.4.1.20.2.8.4 “Tab 'Applications'” on page 845*
- *Chapter 1.4.1.4 “Comparing projects” on page 195*

Unknown applications on the PLC

If one or more applications are already on the PLC, but are not in the project, then CODESYS opens a dialog with a list of these applications. You can then define whether an application should be deleted before loading the current application from the PLC. This also applies to child applications that are on the PLC, but have been deleted from the project in the meantime.

Compiling the project before login

If an application program has not been compiled since the last change, then CODESYS compiles the project before login. This operation is the same as the *“Generate Code”* command when logged out.

If compile errors occur, then a dialog prompt opens. The errors are displayed in the message view in the *“Build”* category. You can then decide whether or not you log in without downloading the program to the PLC.

See also

- *Chapter 1.4.1.20.3.5.4 “Command 'Build'” on page 1022*

Error at login

If an error occurs when logging in to the PLC, then CODESYS cancels the loading operation with an error message. The error dialog gives you the options of showing the error details. If an exception was thrown and the text *SOURCEPOSITION* is included in the log, then you can display the affected function in the editor by clicking *“Show in Editor”*. The cursor jumps to the line containing the error.

Messages during the download operation

If CODESYS downloads the project to the PLC at login, then the following information is printed to the message view:

- Generated code size
- Size of the global data
- Resulting memory requirement on the PLC
- List of the affected blocks (for online change)

In online mode, you cannot change the settings of the devices or modules. You have to logout of the application for changing device parameters. Depending on the bus system, there may be some special parameters that you can also change in online mode.

CODESYS saves the view configuration separately in online and offline mode. In addition, views are closed that cannot be used in any operating mode. For this reason, the view can change automatically at login.

Command 'Logout'

Symbol:  , keyboard shortcut: *[Ctrl]+[F8]*.

Function: This command disconnects the application from the target system (controller or simulated device) and returns to offline mode.

Call: Main menu *“Online”*, or context menu of the *“Application”* object.

Command 'Create Boot Application'

Function: This command generates a boot application.

Call: Main menu *“Online”*.

A boot application is the application that is started automatically when the controller is switched on or started.

In offline mode, you can save the boot application in any directory. In online mode, CODESYS save the boot application to the target device. The file name is `<application name>.app`.

See also

- *Chapter 1.4.1.10.6 “Generating boot applications” on page 391*
- *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*

Command 'Load'

Function: This command causes a compilation of the active application with subsequent download to the controller.

Call: Menu bar: *“Online”*.

Requirement: The application is in online mode.

When you execute this command, CODESYS performs a syntax check and generates the application code. This code is downloaded to the PLC. Furthermore, CODESYS generates the build log `<project name>.<device name>.<application ID>.compile info` in the project directory.

NOTICE!
During loading all variables are re-initialized with the exception of persistent variables.

NOTICE!
If a safety controller is inserted below a controller, then this command can interrupt the communication connections **temporarily**.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, see the "Subordinate Safety Controllers" chapter.

The description of the *“Login”* command describes the possible situations when logging in and loading.

If you attempt to download an application when the same version of the application is already on the PLC, then you get the message: "Program is unchanged. Application was not downloaded". CODESYS downloads the application to the PLC.

During loading a record of the actions being executed (generation of code, execution of initialization, etc.) appears in the Message window in the message category *“Compile”*. Furthermore, information is displayed regarding the memory ranges, the size of the code, the global data and the allocated memory. For the purpose of clarity, as opposed to the online change, the modified function blocks are no longer listed.

See also

- *Chapter 1.4.1.20.3.6.2 “Command 'Login'” on page 1028*
- *Chapter 1.4.1.9.5 “Subordinate safety controller” on page 378*

Command 'Online Change'

Function: The command is used for initiating an online change on the current application. When this is done, CODESYS re-downloads only the changed parts of an application that is already running on the PLC.

Call: Menu bar: *“Online”*; context menu of an *“Application”* object

Requirement: The application is in online mode.

The command is available in the context menu if an application is selected in the device tree. In this way, you can perform an online change just for one application, even if that application is not currently active.

CAUTION!

An online change modifies the running application program and does not cause a restart.

Make sure that the new application code still has the required effect on the controlled system.

Depending on the controlled plant, the plant and workpieces may be damaged or the health and life of persons could be endangered.

NOTICE!

1. When an online change is performed, the application-specific initializations (example: homing) are not executed because the machine retains its status. For this reason, the new program code may not have the intended effect.

2. Pointer variables retain their value from the last cycle. When a pointer refers to a variable whose value was changed in an online change, the variable no longer yields the correct value. Make sure that the pointers are re-assigned in each cycle.

3. After the parent application has been changed, a child application is removed from the controller when an online change is performed.

NOTICE!

For compiler version >= 3.5.0.0, a fast online change is performed for minor changes. In this case, only the modified blocks are compiled and downloaded. In particular, no initialization code is generated. This means that also no code is generated when variables with the `init_on_onlchange` attribute are initialized. As a rule, this has no effect because the attribute is used primarily for initializing variables with addresses. However, it cannot happen that a variable changes its address during an online change.

To secure the effect of the `init_on_onlchange` attribute in the entire application code, you must deactivate the fast online change in general for the application by using the compiler definition `no_fast_online_change`. To do this, insert the definition in the application *"Properties"* (*"Build"* tab).

At the time of download, CODESYS also lists the changed interfaces, affected variables, and all blocks with new generated code in the *"Build"* category of the message view. If memory locations change, a dialog will inform you of possible problems in conjunction with pointers.

In the "Online Change Memory Reserve" view, memory reserves can be configured for the online change so that instance variables do not have to be moved in the memory when changing a function block in an online change.

What prevents an online change?

There are actions in CODESYS after which an online change on a controller is no longer possible. Afterwards, the application always has to be completely recompiled. A typical case is the *"Clean All"* action which deletes the compile information stored at the last download. However, these kind of actions typically generate a warning which you have to acknowledge.

But there are also "normal" editing actions that result in an online change not being possible at the next login. Therefore, pay attention to the following symbol in the status bar when editing in the program POUs: . When this symbol turns red in color (), only a full download to the controller can be performed. Double-clicking the symbol opens the *"Application Information"* dialog with a list of differences to the last download. In the dialog, you also find information about which of the changes prevent an online change.

Actions and changes in different areas of an application that prevent an online change:

Check functions	Activation or removal of a check function (`CheckBounds`, `CheckRange`, `CheckDiv`, etc.) Change in an interface of a check function (also the insertion and deletion of local variables)
Task configuration	Change in the configuration settings
Project settings	Change of the *"Compile Options"* in the *"Settings"* section (Unicode, replace constants, logging in, breakpoints) Change in the *"Compiler defines"*
Application properties	Change of the *"Target system memory settings"* (*"Build"* tab)
POU properties	Change of the *"External implementation"* option (*"Build"* tab)
Task-local global variable list	All changes

Function block	Change of the base POU of a function block (`EXTENDS FBbase`), also the insertion or deletion of such a base POU Change in the interface list (`IMPLEMENTS ITF`). Exception: Adding a new interface at the end of a list
Data type	Change of the data type of a variable from a user-defined data type to another user-defined data type (for example, from `TON` to `TOF`) Change of the data type from a user-defined data type to a base type (for example, from `TON` to `TIME`) Note: As a workaround, you should always change the name of the variable together with the data type. Then the variable is initialized as a new variable and the old one is removed. Then an online change is possible.
Alarm configuration	Change in the alarm database configuration Change of the number of latch variables (also has an effect on the memory format in the database) Change to the configuration of the distributed alarms
Data source	All changes in the configuration
Device configuration	Change in the device tree (also by the *“Update Device”* command) Change in a device configuration: By default, changes to device parameters are not capable of online change. However, exceptions can be configured in the device description. Note: I/O mapping to variables is possible by online change.
Visualization	Toggling of the overlay function Before V3.5 SP6: Change in the configuration of the trace element Note: In V3.5 SP6 and higher, the following applies: For online changes that affect visualizations or affect the data of the application (for example, a new variable is inserted), the visualization is completely reinitialized. For TargetVisu, for example, this means that the visualization closes and reopens with the start page. For WebVisu, the visualization also restarts with the start visualization after a short waiting period.
Unit conversion	Insertion or removal of objects for unit conversion
Trend	Change of the number of variables or maximum number of variables. Change of the number of variables with a description or special line settings

See also

- *Chapter 1.4.1.20.3.5.3 “Command 'Clean All'” on page 1021*
- *Chapter 1.4.1.10.4 “Generating Application Code” on page 389*
- *Chapter 1.4.1.13.1 “Executing the online change” on page 439*
- *“ Dialog 'Application Information' (Details)” on page 1030*
- *Chapter 1.4.1.19.6.2.20 “Attribute 'init_on_onlchange'” on page 705*
- *Chapter 1.4.1.20.2.27.1 “Tab 'Configuration'” on page 942*
- *Chapter 1.4.1.20.4.11.3 “Dialog Box 'Project Settings' - 'Compileoptions'” on page 1173*
- *Chapter 1.4.1.20.4.10.4 “Dialog 'Properties' - 'Build'” on page 1159*

Command 'Source Download to Connected Device'

Function: This command loads the project source code (as project archive) to the controller currently connected.

Call: Main menu *“Online”*.

Requirement: The application is in online mode.

See also

- *Chapter 1.4.1.20.3.1.10 “Command 'Source Upload'” on page 962*
- *Chapter 1.4.1.20.3.1.11 “Command 'Source Download'” on page 963*

Command 'Download Manager'

Symbol:

Function: Download or create a boot project from the project devices or update the firmware oft he device.

Call: Main menu *“Online”*, Context menu.

Requirement: A project is open.

Command 'Multiple Download'

Function: The command causes the code generation of the applications contained in the project as well as the loading of the applications to the corresponding controllers.

Call: Menu bar: *“Online”*

The command opens a dialog with a list of the applications. In this dialog, select the applications that are to be loaded. Then, CODESYS performs the syntax check of these applications and generates the respective code. The code is then downloaded to the respective PLC.
For each selected application, CODESYS generates a build log with the name `<project name>.<device name>.<application ID>.compileinfo` in the project directory.

NOTICE!

If a safety controller is inserted below a controller, then this command can interrupt the communication connections **temporarily**.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, see the "Subordinate Safety Controllers" chapter.

Dialog 'Multiple Download'

“Please select the items to be downloaded”	: Selection of the applications. The applications are thereby also loaded to different controllers.
“Move Up”, *“Move Down”*	Change of the order of download of the applications. The applications are downloaded to the PLCs in the order of this list. By default, this list is alphabetically sorted. Parent-child relationships of applications are thereby taken into account.
“OK”	Checks the syntax of all selected applications. Afterwards, the communication with the associated controller is verified for each application before the download takes place.

Table 138: “Online Change Options”

If an earlier version already exists on the PLC and is different from the current version, then the following options are provided:	
“Try to perform an online change. If this is not possible, perform a full download.”	Activated by default. If an online change cannot be executed for one of the applications, then a download is performed.
“ Force an online change. If this is not possible, cancel the operation.”	If an online change cannot be performed for (at least) one of the applications, then no download is performed and the online change is terminated (for example, if you have executed the command *“Clean All”* beforehand).
“Always perform a full download.”	Downloads all parts of the applications to the PLC, regardless of any existing versions.

For selected applications that do not exist on the PLC yet, CODESYS performs a download automatically to the PLC.

Table 139: “Other Options”

“Delete all applications on the PLC which are not part of the project. ”	☑: Corresponding applications are deleted
“Start all applications after download or online change”	☑: The applications are started after the download or online change.
“Do not release forced variables”	☑: If an application with forced variables exists on the controller, and if the implementation of this application has been changed, then no download is performed for this application. The message *“Error: Skipped because one or more variables have been forced”* appears for this application in the window *“Multiple Download - Result”*.

Note that variables with the key attribute `PERSISTENT RETAIN` are not generally initialized. If you change the data layout, however, the persistent variables are automatically re-initialized.

After completion of the download a listing of all selected applications appears in the download order that you configured. In addition, you are shown information on the success of the download for each application in the *“Multiple Download - Result”* dialog:

- *“Created”*: A new application has been created and downloaded to the controller.
- *“Not changed”*: The application which exists on the controller has not been changed.
- *“Online changed”*: The application which exists on the controller has been modified by an online change.
- *“Downloaded”*: The application which exists on the controller has been replaced by a new created application.
- *“Skipped due to impossible online change”*: An online change could not be performed for the application. The application was not changed.
- *“Error”*: An error has occurred for this application during download. More details may be displayed.
- *“Cancelled by user”*: The operation has been aborted by the user.

See also

- *Chapter 1.4.1.9.5 “Subordinate safety controller” on page 378*
- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.10.4 “Generating Application Code” on page 389*
- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.20.3.6.5 “Command 'Load'” on page 1032*

Command 'Reset Cold'

Function: The command results in a cold start of the active application on the controller.

Call: Menu bar: *"Online"*

Requirement: The application is in online mode.

> **NOTICE!**
> If a safety controller is inserted below a controller, then this command can interrupt the communication connections **temporarily**.
>
> Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.
>
> For more information, see the "Subordinate Safety Controllers" chapter.

After restarting with *"Reset Cold"*, the following happens:

- Application code is retained on the controller.
- Variables are initialized (with the initialization value or the default initialization value 0), and the previous values are lost.
- Retain variables are initialized, and the previous values are lost.
- Persistent variables are retained with values.
- Breakpoints that were set in the code are retained with their status (for example, activated or deactivated).
- The application goes into the *"STOP"* state.

You can also select the command while debugging the application when it halts at a breakpoint in the *"HALT ON BP"* state. Then either the warm start is executed immediately, or the remaining statements of the current cycle are processed. Therefore, a message window opens for you to select the next action. However, the message window opens only if the runtime system is capable of restarting the cycle without terminating it first.

After the reset, you can run the application as usual and, for example, start the execution by clicking *"Debug ➔ Start"*.

See also

- *Chapter 1.4.1.11.5 "Resetting applications" on page 404*
- *Chapter 1.4.1.8.19.1 "Preserving data with persistent variables" on page 304*
- *Chapter 1.4.1.11.2 "Using Breakpoints" on page 395*
- *Chapter 1.4.1.19.2.12 "Persistent Variable - PERSISTENT" on page 535*
- *Chapter 1.4.1.9.5 "Subordinate safety controller" on page 378*
- *Chapter 1.4.1.20.3.6.11 "Command 'Reset Warm'" on page 1038*
- *Chapter 1.4.1.20.3.6.12 "Command 'Reset Origin'" on page 1039*

Command 'Reset Warm'

Function: The command results in a warm start of the active application on the controller.

Call: Menu bar: *"Online"*

Requirement: The application is in online mode.

NOTICE!

If a safety controller is inserted below a controller, then this command can interrupt the communication connections **temporarily**.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, see the "Subordinate Safety Controllers" chapter.

After restarting with *“Reset Warm”*, the following happens:

- Application code remains loaded on the controller.
- Variables are initialized (with the initialization value or the default initialization value 0).
- Retain variables are retained with values.
- Persistent variables are retained with values.
- Breakpoints that were set in the code are retained with their status (for example, activated or deactivated).
- The application goes into the *“STOP”* state.

You can also select the command while debugging the application when it halts at a breakpoint in the *“HALT ON BP”* state. Then either the warm start is executed immediately, or the remaining statements of the current cycle are processed. Therefore, a message window opens for you to select the next action. However, the message window opens only if the runtime system is capable of restarting the cycle without terminating it first.

After the reset, you can run the application as usual and, for example, start the execution by clicking *“Debug ➔ Start”*.

See also

- *Chapter 1.4.1.11.5 “Resetting applications” on page 404*
- *Chapter 1.4.1.8.19.1 “Preserving data with persistent variables” on page 304*
- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*
- *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*
- *Chapter 1.4.1.9.5 “Subordinate safety controller” on page 378*
- *Chapter 1.4.1.20.3.6.10 “Command 'Reset Cold'” on page 1038*
- *Chapter 1.4.1.20.3.6.12 “Command 'Reset Origin'” on page 1039*

Command 'Reset Origin'

Function: The command results in a reset origin of the active application on the controller.

Call: Menu bar: *“Online”*

Requirement: The application is in online mode.

NOTICE!

If a safety controller is inserted below a controller, then this command can interrupt the communication connections **permanently**.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, see the "Subordinate Safety Controllers" chapter.

After restarting with *"Reset Origin"*, the following happens:

- The application code is deleted, and as a result the application has no state.
- Variables are deleted, and the values are lost.
- Retain variables are deleted, and the values are lost.
- Persistent variables are deleted, and the values are lost.
- Breakpoints that were set in the code are lost.

See also

- *Chapter 1.4.1.11.5 "Resetting applications" on page 404*
- *Chapter 1.4.1.8.19.1 "Preserving data with persistent variables" on page 304*
- *Chapter 1.4.1.11.2 "Using Breakpoints" on page 395*
- *Chapter 1.4.1.19.2.12 "Persistent Variable - PERSISTENT" on page 535*
- *Chapter 1.4.1.9.5 "Subordinate safety controller" on page 378*
- *Chapter 1.4.1.20.3.6.11 "Command 'Reset Warm'" on page 1038*
- *Chapter 1.4.1.20.3.6.10 "Command 'Reset Cold'" on page 1038*

Command 'Reset Origin Device'

Function: The command opens a dialog to reset the device to its factory settings. All applications, boot applications, and remanent variables will be deleted from the device. Depending on the version of the device, a selection of the elements to be deleted can be made in this dialog. When these elements are unselected in the dialog, they are not deleted during the reset and remain on the controller. By default, all elements are selected and everything is deleted. Elements that are not available for selection are generally also deleted.

Call: Right-click a programmable device in the device tree.

NOTICE!

If a safety controller is inserted below a controller, then this command can interrupt the communication connections **permanently**.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, see the "Subordinate Safety Controllers" chapter.

After restarting with *"Reset Origin Device"*, the following happens:

- All applications are reset as with the *"Reset Origin"* command.
- All files, which are not deleted by the *"Reset Origin"* command, are deleted (for example, files from visualization, alarms, and recipes).

- The user management is deleted.
- All certificates which are currently managed by the runtime system are deleted.

Note: When resetting the device, the objects selected in this dialog are also deleted. If not all displayed objects are selected in this dialog, then possibly other objects can no longer be used or they are also deleted. .	
"Delete"	☑: The object is deleted when the *"Reset Origin Device"* command is executed.
"Object"	Objects that can be excluded from *"Delete"*. The listed objects depend on the version of the controller. In version 3.5.16.20 and higher, the following objects can be excluded from the delete operation. ● *"User Management"* ● *"PLC Logic"* ● *"Certificates"*

See also

- *Chapter 1.4.1.20.3.6.12 "Command 'Reset Origin'" on page 1039*
- *Chapter 1.4.1.20.3.6.11 "Command 'Reset Warm'" on page 1038*
- *Chapter 1.4.1.20.3.6.10 "Command 'Reset Cold'" on page 1038*
- *Chapter 1.4.1.9.5 "Subordinate safety controller" on page 378*

Command 'Logoff Current Device User'

Symbol: 

Function: This command logs out the user currently logged in to the controller (device). If CODESYS still has a connection to the controller, then it will be disconnected.

Call: Main menu *"Online"*.

Requirement: The application is in online mode.

You can manage the device user management in the "Users and Groups" tab and "Access control" of the device editor. The commands in the "Online ➔ Security" menu provide another simple option for protecting access to the target device.

See also

- *Chapter 1.4.1.20.2.8.13 "Tab 'Users and Groups'" on page 860*
- *Chapter 1.4.1.10.3 "Handling of Device User Management" on page 385*

Command 'Download'

Function: This command loads the compiled project in the PLC.

Call: Main menu *"Online"*, Context menu.

Requirement: A project is open. Log-in required for download.

Command 'Add Device User'

Symbol: 

Function: This command configures a new device user and adds this user to the administrator group.

Call: Menu bar: *“Online ➔ Security”*

Requirement: The device supports a device user management. You are logged in to the device as a user.

You can manage the device user management in the “Users and Groups” tab and “Access control” of the device editor. The commands in the “Online ➔ Security” menu provide another simple option for protecting access to the target device.

See also

- *Chapter 1.4.1.20.2.8.13 “Tab 'Users and Groups'” on page 860*
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

This command opens the *“Add Device User”* dialog. Here you define the access data of the new user.

The dialog corresponds to the dialog in the *“Users and Groups”* tab of the device editor for adding a new user.

Please use a strong password as follows:

- Password length >= 8 characters (best >= 12)
- Use uppercase and lowercase
- Include numbers
- Use special characters
- Do not use existing names or sequence of characters that are easy to guess (for example, “123”, “abc”, “qwerty”)

CAUTION!
After performing this action, you can no longer use a blank username and password to log in. You must remember your password.

See also

- *Chapter 1.4.1.20.3.6.17 “Command 'Remove Device User'” on page 1042*
- *Chapter 1.4.1.20.3.6.18 “Command 'Change Password Device User'” on page 1043*

Command 'Remove Device User'

Symbol: 

Function: This command removes a user from the user management on the target system (device).

Call: Menu bar: *“Online ➔ Security”*

Requirement: You are logged in to the device as a user.

You can manage the device user management in the “Users and Groups” tab and “Access control” of the device editor. The commands in the “Online ➔ Security” menu provide another simple option for protecting access to the target device.

See also

- *Chapter 1.4.1.20.2.8.13 “Tab 'Users and Groups'” on page 860*
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

This command opens the *“Remove Device User”* dialog. Specify the user name and password of the user to be removed and click *“OK”* to confirm.

CAUTION!
After performing this action, you can no longer use this removed user account to log in. If this user is the only one on the target system, then a dialog prompt notifies you that this user cannot be removed.

See also

- *Chapter 1.4.1.20.3.6.16 “Command 'Add Device User'” on page 1041*
- *Chapter 1.4.1.20.3.6.18 “Command 'Change Password Device User'” on page 1043*

Command 'Change Password Device User'

Symbol:

Function: The command changes the password for the user who is currently logged on the PLC.

Call: *“Online* ➔ *Security”* menu

Requirement: You are logged in to the device as a user.

The command opens the *“Change Password for Device User”* dialog for defining a new password. You have to specify the old password again.

NOTICE!
After performing this action, you can no longer use the previous password to log in.

Make sure that you use a strong password. Note the following:

- Password length >= 8 characters (best >= 12)
- Use uppercase and lowercase
- Include numbers
- Use special characters
- Do not use existing names or sequence of characters that are easy to guess (for example, "123", "abc", "qwerty")

See also

- *Chapter 1.4.1.20.3.6.16 “Command 'Add Device User'” on page 1041*
- *Chapter 1.4.1.20.3.6.17 “Command 'Remove Device User'” on page 1042*
- *Chapter 1.4.1.10.3 “Handling of Device User Management” on page 385*

Command 'Stop Execution on Handled Exceptions'

Function: This command halts the application where the error is located despite a programmed exception handling.

Call: This command is not available by default, but it can be configured from the *“Tools* ➔ *Customize”*, *“Add Command”* dialog box (*“Online”* category).

Requirement: The application is in online mode and contains a programmed exception handling with the `__TRY` and `__CATCH` operators.

If you have configured this command from the *“Online”* menu and you call it from there, then the currently active application is affected. Furthermore, this command can help you to detect errors.

See also

- *Chapter 1.4.1.19.3.61 “Operators '__TRY', '__CATCH', '__FINALLY', '__ENDTRY'” on page 619*
- *“Adding commands” on page 181*
- *Chapter 1.4.1.20.3.8.16 “Command 'Customize'” on page 1071*

Command 'Connect to Device'

Function: The command establishes a connection to the device currently selected in the device tree.

Call: Context menu of the device.

Requirements: A device is selected in the device tree. The communication settings are configured correctly.

See also

- *Chapter 1.4.1.20.3.6.21 “Command 'Disconnect from Device'” on page 1044*

Command 'Disconnect from Device'

Function: The command disconnects the connection from a device.

Call: Context menu of the device.

Requirements: A device is selected in the device tree.

See also

- *Chapter 1.4.1.20.3.6.20 “Command 'Connect to Device'” on page 1044*

Command 'Wink'

Symbol: 

Function: The command causes an LED of a connected controller to blink. As a result, the hardware can be identified clearly.

Call: The command is not in any menu by default. You can add it to a menu by means of the *“Tools ➔ Customize”* dialog, in the *“Online”* command category.

Requirement: The controller supports this function and the connection parameters are configured correctly.

Command 'Simulation

Function: The command switches the development system to simulation mode.

Call: Menu bar: *“Online”*

In simulation mode, you can start and debug the active application an on a simulated target device. A physical target device is not necessary for testing the online behavior of an application. When logging in for the first time, you are prompted whether the application should be created or loaded. For a simulated device, you do not have to configure the communication settings. In CODESYS simulation mode, the entry of the controller in the device tree is displayed in italics.

NOTICE!

No C code for simulation mode

In simulation mode, C code is not generated and loaded to the runtime system. To simulate the code contained in the C modules anyway, you can implement it for this purpose in the respective IEC objects of the C code module.

After successful login, the red triangle symbol (⚠) in the device tree indicates simulation mode. You can use the corresponding online commands for testing the application.

To switch off simulation mode, log out of the controller and execute the *"Simulation"* command again.

The command affects the active application only.

Differences between simulation mode and operation with a physical controller

	Simulation	Physical Controller
Real-time behavior / multicore	● Runs in the CODESYS process with normal priority ● Single-core --> Worse real-time behavior	● Real-time operating system ● Single-core or multicore
Architecture scope	● Simulation 64-bit (depends on the CODESYS installation): --> Possible compile error in the IEC application if the application has been previously run only as 32-bit (for example, use of `DWORD` as POINTER)	● Controller 32-bit
FPU (rounding error)	● Uses FPU of the PC ● Different configuration of the FPU exceptions	● Uses FPU of the controller or FPU emulation ● Different configuration of the FPU exceptions
Handling of exceptions	● Exception handling of the Windows Runtime System	● Exception handling of the controller

	Simulation	Physical Controller
External libraries (Cmp/Sys/CAA/OEM/…)	● Only a few external Cmp/SysLibs are physically available. As compared to embedded, more SysLibs could also be available. ● Other implementation/behavior of the SysLibs (Windows in contrast to the OS of the controller) ● "Unresolved Reference error“ on download is ignored. The application can still be downloaded to the controller and started. If the missing functions are actually called, they return nonsense values. Therefore, an IEC implementation can also be specified for external POUs. This substitute IEC code is then executed in the simulation.	● "Unresolved Reference error" on download when external libraries do not exist in the controller
I/O drivers	● I/O configuration is generated but not evaluated. ● Fieldbus stacks are not evaluated. ● I/O channels are not updated and no bus telegrams are sent.	● Mostly no restriction, but depends on the capabilities of the controller
SoftMotion drivers	● All SoftMotion axes are set to virtual and therefore simulated.	● Mostly no restriction, but depends on the capabilities of the controller

See also

- *Chapter 1.4.1.11.1 “Testing in simulation mode” on page 394*

Command 'Operating Mode'

Function: The commands set the controller to a state which prevents accidental change to the project.

Call: Menu bar: *“Online ➔ Operating Mode”*

You can use these commands, for example, to lock the state of a controller in order to prevent the controller from switching to another state while you program another controller.

When programming is complete, the controller should then be switched to a defined and externally visible state that is set exactly the same way after restarting.

The , , and symbols in the status bar indicate the current operating mode. Double-clicking one of these symbols opens a help window.

If it supports the controller, then you can switch the controller to the following operating modes:

- *“Debug”*: No restrictions
- *“Locked”*: The current state of debugging is locked on the application. No additional breakpoints can be set and no additional variables can be forced. Writing variables is still possible and breakpoints which are already set remain active.
  Only the "RUN" state of an application is preserved in *“Locked”* operating mode even if the controller is restarted.
  With this operating mode, a developer can prevent himself or another developer from changing the application on the controller, for example by setting or deleting a breakpoint, by forcing, or by making changes to the file system. This operating mode is helpful to prevent a download to an incorrect controller when, for example, multiple controllers of a plant are programmed.
- *“Operational”*:
  This operating mode makes sure that the controller reloads the same applications after a restart and that no debug features are active anymore. The operating mode is set when a controller is completely programmed and should be accepted or already is.
  Conditions for activating the *“Operational”* mode
  – A boot application for each application has to exist on the controller.
  – There must not be any active breakpoints set.
  – All applications have to be running.
  – There must not exist any forced values.
  – Furthermore, the device can define more of its own restrictions.

The *“Locked”* and *“Operational”* operating modes are different in the use cases and in the requirements for activating the operating mode. However, for both operating modes the runtime system prevents the following actions:

- Regarding the application
  – Download of an application
  – Online change
  – Force variables
  – Set breakpoints
  – Stop application
  – Reset application
  – Start application
  – Delete application
- Regarding the file transfer of the controller
  – Download of a file to the controller
  – Delete a file on the controller
  – Rename a file on the controller
  – Create a directory on the controller
  – Delete a directory on the controller
  – Rename a directory on the controller

You cannot switch the operating mode between “Locked” and “Operational”.

Command 'Virtual mode'

Function: *“Virtual Mode”* option enables virtual mode for Automation Builder.

Call: Main menu *“Online”*, Context menue

Requirement: A project is open. Command is only available, if a license for advanced simulation support is acitvated.

Command 'Virtual system testing'

Function: The *“Virtual system testing”* editor contains settings for the virtual devices and the simulation set-up and control.

Call: Main menu *“Online”*, Context menu

Requirement: A project is open. Command is only available, if a license for advanced simulation support is acitvated.

Menu 'Debug'

Command 'Start'

Symbol: ; keyboard shortcut: *[F5]*

Function: This command starts the application (status: *“RUN”*).

Call: Menu bar: *“Debug”*; context menu of object: *“Application”*

Requirement: The application is in online mode and its status is *“STOP”*.

Executing this command from the *“Debug”* menu will affect the application that is currently in focus.

See also

- *Chapter 1.4.1.10.5 “Downloading the application code, logging in, and starting the PLC” on page 391*

Command 'Stop'

Symbol: ; keyboard shortcut: *[Shift]+[F8]*

Function: This command stops the application (status: *"STOP"*).

Call: Menu bar: *"Debug"*; context menu of object: *"Application"*

Requirement: The application is in offline mode and its status is *"RUN"*.

Executing this command from the *"Debug"* menu will affect the application that is currently in focus.

Command 'Single Cycle'

Keyboard shortcut *[Ctrl]+[F5]*

Function: This command executes the active application for one cycle.

Call: Main menu *"Debug"*.

Requirement: The application is in online mode and the program is halted at a program step.

Command 'New Breakpoint'

Symbol: , keyboard shortcut *[Alt]+[F7]*.

Function: This command opens the *"Breakpoint Properties"* dialog box.

Call: Main menu *"Debug"*.

Requirement: The application must be in online mode.

With the "Toggle Breakpoint" command, you can set a new breakpoint directly at the current cursor position in online mode.

See also

- *Chapter 1.4.1.11.2 "Using Breakpoints" on page 395*

Command 'New Data Breakpoint'

Symbol:

Function: The command opens the *"New breakpoint"* dialog.

Call: Menu bar: *"Debug"*

Requirement:

- The application is in online mode.
- The device description file of the target device contains the entries for the "data breakpoints" functionality. Currently, data breakpoints are possible only with the CODESYS Control Win V3.

See also

- *Chapter 1.4.1.20.4.8 "Dialog 'New Breakpoint'" on page 1154*
- *Chapter 1.4.1.11.2 "Using Breakpoints" on page 395*

Command 'Edit Breakpoint'

Symbol:

Function: This command opens the *“Breakpoint Properties”* dialog box.

Call: Main menu *“Debug”*.

Requirement: The application is in online mode and the cursor is halted at a breakpoint.

See also

- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*

Command 'Enable Breakpoint'

Function: This command enables a disabled breakpoint.

Call: Main menu *“Debug”*.

Requirement: The application is in online mode and the cursor is halted at a disabled breakpoint.

See also

- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*

Command 'Disable Breakpoint'

Function: This command disables an enabled breakpoint.

Call: Main menu *“Debug”*.

Requirement: The application is in online mode and the cursor is halted at an enabled breakpoint.

See also

- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*

Command 'Toggle Breakpoint'

Keyboard shortcut *[F9]*

Function: This command sets a breakpoint or clears an existing breakpoint.

Call: Main menu *“Debug”*.

Requirement: The application is in online mode. The cursor is positioned at a breakpoint.

See also

- *Chapter 1.4.1.11.2 “Using Breakpoints” on page 395*

Command 'Step Over'

Symbol  , shortcut *[F10]*

Function: The command executes the statement where the program is currently located and halts before the next statement in the POU.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode and the program is halted at the current break position (debug mode).

If the executed statement contains a call (from a program, function block instance, function, method, or action), then the subordinate POU is processed completely in one step and returned to the call. Then it halts before the next statement (in the next line of code).

Click “Step Into” to jump to a subordinate POU and process it in single steps.

See also

- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*
- *Chapter 1.4.1.20.3.7.11 “Command 'Step Into'” on page 1051*

Command 'Step Into'

Symbol , shortcut *[F11]*

Function: The command executes the statement where the program is currently located and halts before the next statement.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode and the program is halted at the current break position (debug mode).

If the executed statement contains a call (from a program, function block instance, function, method, or action), then the program execution jumps to this subordinate POU. Its code opens in a separate editor. The first statement there is executed and the program execution halts before the next statement. The new current breakpoint position is then in the called POU.

Click “Step Over” to remain in the currently active POU and execute the call in one step.

See also

- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*
- *Chapter 1.4.1.20.3.7.10 “Command 'Step Over'” on page 1050*

Command 'Step Out'

Symbol , shortcut *[Ctrl]+[F11]*

Function: The command executes the program until the next return and halts afterwards.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode and the program is halted at the current break position (debug mode).

If the current breakpoint position is in a subordinate POU, then this is run through to the end. Then the program execution jumps back to the calling point in the calling POU and halts there (in the line with the call).

If the current breakpoint position is in the main program, then the POU is run through to the end. Then the program execution jumps back to the beginning (to the program start at the first line of code in the POU) and halts there.

See also

- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*

Command 'Run to Cursor'

Symbol: 

Function: The command executes a program until a specified position as marked by the cursor.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode and the program is halted at the current break position (debug mode). Moreover, you have marked any line of code in any POU with the cursor.

The statements between the current breakpoint position and the cursor position are executed in one step. Then the execution halts at the cursor position, which then becomes the next breakpoint position. Remember that the line of code where you placed the cursor is reached but not executed.

See also

- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*

Command 'Set Next Statement'

Symbol: 

Function: The command determines which statement is executed next.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode and the program is halted at the current break position (debug mode). Moreover, you have marked any line of code in any POU with the cursor.

The line of code marked with the cursor becomes the current breakpoint position without executing the statements in between or the statement that jumped to it.

See also

- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*

Command 'Show Next Statement'

Symbol: 

Function: The command displays the program statement that is processed in the next step.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode and the program is halted at the current break position (debug mode). The break position is in a line of code that you cannot see.

The command makes the window with the current breakpoint position active (in the code highlighted in yellow and marked with the symbol) and makes the breakpoint position to become visible. This is useful if you have multiple editors open and the breakpoint position is hidden in an inactive editor.

See also

- *Chapter 1.4.1.11.3 “Stepping Through a Program” on page 399*

Command 'Force Values'

Keyboard shortcut: *[F7]*

Function: The command sets a permanent predefined value to a variable on the controller.

Call: Menu bar: *“Debug”*

Requirement: The application is in online mode.

CAUTION!
Unusual changes to variable values in an application currently running on the controller can lead to undesired behavior of the controlled machinery.
Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled machinery, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

With this command, CODESYS permanently sets one or more variables of the active application to defined values on the PLC.

A forced value is marked with the forced symbol ( ).

For more information about the functionality of forcing and the p of values, see the "Forcing and Writing of Variables" help page.

By default, the “Force Values [All Applications]” command, which applies to all application in the project, and is not included in a menu.

See also

- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *Chapter 1.4.1.20.4.7 “Dialog Box 'Prepare Value'” on page 1153*
- *Chapter 1.4.1.20.3.7.18 “Command 'Unforce Values'” on page 1054*

Command 'Write Values'

Keyboard shortcut *[Ctrl]-[F7]*

Function: This command sets a predefined value to a variable on the controller one time.

Call: Main menu *“Debug”*.

Requirement: The application is in online mode.

CAUTION!
Unusual changes to variable values in an application currently running on the controller can lead to undesired behavior of the controlled machinery.
Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled machinery, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

With this command, one or more variables of the **active application** are set to defined values on the controller one time. Writing is done one time at the beginning of the next cycle.

Values are prepared by

- Clicking in the field *“Prepared value”* in the declaration section
- Clicking in the inline monitoring field in the implementation section
- Clicking in the field *“Prepared value”* in the watch window

The command “Write Values [All Applications]” affects all application in the project and is not included in a menu by default.

See also

- *Chapter 1.4.1.20.3.7.16 “Command 'Force Values'” on page 1053*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*

Command 'Unforce Values'

Keyboard shortcut *[Alt]+[F7]*

Function: This command resets the forcing of all variables. The variables receive their current values from the PLC.

Call: *“Debug”*.

Requirement: The application is in online mode.

The *“Remove Force List”* command has the same functionality as this command with one difference. If the *“Remove Force List”* command cannot be executed for all forced values, then no message is displayed.

CAUTION!

Unusual changes to variable values in an application currently running on the PLC can lead to undesired behavior of the controlled machinery.

Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled system, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

The command “Force Values [All Applications]” affects all application in the project and is not included in a menu by default.

See also

- *Chapter 1.4.1.20.3.7.16 “Command 'Force Values'” on page 1053*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*

Command 'Force All Values from <Device.Application>'

Function: This command resets all values of variables from the selection <Device.Application> to predefined values permanently.

Call:

- Context menu of the application in the device tree
- Context menu in the editor of a POU from the selected application

Requirement: The application is in online mode.

CAUTION!
Unusual changes to variable values in an application currently running on the controller can lead to undesired behavior of the controlled machinery.
Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled machinery, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

With this command, CODESYS permanently sets one or more variables of the active application to defined values on the PLC. This is done at the beginning and end of a processing cycle. Order of processing: 1) read inputs, 2) force values, 3) process code, 4) force values, 5) write outputs.

You can prepare values as follows:

- Click in the *“Prepared value”* field in the declaration part and type in the value. For Boolean variables, you change the value by clicking the field.
- Click in the inline monitoring field in the implementation part of the FBD/LD/IL editor
- Click in the *“Prepared value”* field in the monitoring view and type in the value.

A forced value is marked with the forced symbol ( ).

CODESYS forces the value until you explicitly lift it by

- Clicking *“Unforce Values”*
- Clicking *“Unforce All Values from <Device.Application>”*
- Lifting the force in the *“Prepare Value”* dialog
- Logging out of the application

The command “Force Values [All Applications]” affects all application in the project and is not included in a menu by default.

See also

- *Chapter 1.4.1.20.4.7 “Dialog Box 'Prepare Value'” on page 1153*
- *Chapter 1.4.1.20.3.7.18 “Command 'Unforce Values'” on page 1054*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *Chapter 1.4.1.20.3.7.21 “Command 'Unforce All Values from <Device.Application>'” on page 1056*

Command 'Write All Values from <Device.Application>'

Function: This command resets all values of variables from the selection <Device.Application> to predefined values one time.

Call:

- Context menu of the application in the device tree
- Context menu in the editor of a POU from the selected application

Requirement: The application is in online mode.

CAUTION!
Unusual changes to variable values in an application currently running on the controller can lead to undesired behavior of the controlled machinery.
Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled machinery, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

With this command, one or more variables of the selected <Device.Application> are set to defined values on the PLC one time. Writing is done one time at the beginning of the next cycle.

You can prepare values as follows:

- Click in the "Prepared value" field in the declaration part and type in the value. For Boolean variables, you change the value by clicking the field.
- Click in the inline monitoring field in the implementation part of the FBD/LD/IL editor
- Click in the "Prepared value" field in the monitoring view and type in the value.

See also

- *Chapter 1.4.1.20.3.7.17 “Command 'Write Values'” on page 1053*
- *Chapter 1.4.1.20.3.7.19 “Command 'Force All Values from <Device.Application>'” on page 1054*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*

Command 'Unforce All Values from <Device.Application>'

Function: This command resets the forcing of all values of the variables from the selected <Device.Application>. The variables receive their current values from the PLC.

Call:

- Context menu of the application in the device tree
- Context menu in the editor of a POU from the selected application

Requirement: The application is in online mode.

CAUTION!

Unusual changes to variable values in an application currently running on the controller can lead to undesired behavior of the controlled machinery.

Evaluate possible dangers before forcing variable values. Take the respective safety precautions. Depending on the controlled machinery, the result may lead to damage to machinery and equipment or injury to health and life of personnel.

See also

- *Chapter 1.4.1.20.3.7.19 “Command 'Force All Values from <Device.Application>'” on page 1054*
- *Chapter 1.4.1.20.3.7.18 “Command 'Unforce Values'” on page 1054*

Command 'Flow Control'

Function: This command activates and deactivates the flow control.

Call: Menu*“Debug”*

Requirement: The application is in online mode.

NOTICE!

An active flow control extends application runtime.

When *“Confirmed Online Mode”* is activated in the communication settings, a dialog prompt appears when switching on the flow control to cancel the process.

When flow control is activated, it is not possible to use breakpoints or step through the program.

See also

- *Chapter 1.4.1.11.6 “Flow Control” on page 406*

Menu 'Core Dump'

Command 'Load Core Dump'

Function: CODESYS scans the project directory for core dump files. When a new core dump is forced with the *"Create Core Dump"* command, the dump file is automatically loaded from the controller to the project directory. If multiple core dump files are available, then CODESYS prompts you to choose whether the latest file should be opened in the project. You can also select one of the other files.

When a file is loaded into the project, an online view of the application appears with state of the application at the time when the core dump was generated. You can then view the variable values afterwards. Finally, the call tree is also available.

Call: Main menu *"Debug ➔ Core Dump"*.

Requirement: The application is in offline mode.

> **NOTICE!**
> You can close the core dump view only by clicking *"Close Core Dump"*. The *"Logout"* command has no effect in this view.

See also

- *Chapter 1.4.1.20.3.7.23.2 "Command 'Create Core Dump'" on page 1057*
- *Chapter 1.4.1.20.3.7.23.3 "Command 'Close Core Dump'" on page 1058*

Command 'Create Core Dump'

Function: This command causes CODESYS to check whether a core dump file is already available on the controller.

If a core dump file is available, then CODESYS prompts you to load this file to the project directory.

With the following requirements, CODESYS generates a new dump file with the current application data:

- A core dump file is still not available or CODESYS has rejected a core dump file from being loaded.
- The application is currently stopped at breakpoint or an exception has occurred.

The generated core dump file is saved directly to the project directory: `<project name>.<device name>.<application name>.<application Guid>.core`. You can cancel the file generation by clicking the button in the status bar.

The amount of detail in the dump depends on the support from the runtime system. Runtime systems that are appropriate for this purpose generate just one dump in the case of an exception error. The core dump output from clicking *"Load Core Dump"* can therefore be used for error analysis.

Call: Main menu *"Debug ➔ Core Dump"*.

Requirement: The application is in online mode.

See also

- *Chapter 1.4.1.20.3.7.23.1 "Command 'Load Core Dump'" on page 1057*

Command 'Close Core Dump'

Function: This command closes the core dump view of the application that is open in the project.

Call: Main menu *“Debug ➔ Core Dump”*.

Requirement: The application is in offline mode and you have loaded a core dump file to the project from the controller.

Command 'Load Device Log from Core Dump'

Function: This command imports the controller log list that was saved with the last generated core dump. The log list is displayed in the same view as in online mode in the *“Log”* tab of the device editor.

Call: Main menu *“Debug ➔ Core Dump”*.

Requirement: The application is in offline mode and a core dump is open in the project.

See also

- *Chapter 1.4.1.20.2.8.8 “Tab 'Log'” on page 848*

Command 'Display Mode' - 'Binary', 'Decimal', 'Hexadecimal'

Function: These commands in the *“Display Mode”* submenu are used for setting the format of values in the display mode when monitoring in online mode.

Call: Main menu *“Debug”*.

Requirement: The project is in either online or offline mode.

The "Binary" and "Hexadecimal" display modes are unsigned, and "Decimal" is signed.

See also

- *Chapter 1.4.1.12.1.1 “Calling of monitoring in programming objects ” on page 410*

Menu 'Tools'

Command 'IP-Configuration'

Function: Scan the project for IP address, device ID and other Informations.

Call: Main menu *“Tools”*, Context menu

Requirement: -

Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675

Command 'Install additional licence'

Function: Installs additional engineering license.

Call: Main menu *“Tools”*, Context menu

Requirement: -

Command 'Migrate third party devices'

Function: After a selection of a previous version profile, all the third party devices which have been installed inside this version profile are listed and can migrated.

Call: Main menu *“Tools”*, Context menu

Requirement: -

Chapter 1.6.6.1.5 “Migration of third party devices” on page 3658

Command 'Package Manager'

Symbol: 

Function: The command opens the *“Package Manager”* dialog where you install, uninstall, and manage packages.

Call: Menu bar: *“Tools”*

You can also call the Package Manager as a standalone application from the command line.

Table 140: “Currently Installed Packages”

List of installed packages with *“Name”*, *“Version”*,*“Installation date”*, *“Update info”*, *“License info”* If a package originates from the CODESYS Store, then CODESYS identifies it with the red package symbol instead of the yellow symbol. When an update is available, CODESYS indicates this with an entry in the *“Update info”* column and with the symbol.	
“Refresh”	Refreshes the list
“Install”	Opens the standard dialog for finding a file in the file system. By default, the file type is *.package. You can also install two versions of a package. After you select the package, the *“Check package signatures”* dialog opens. ● In the dialog, the package is displayed with the information about signing. Detailed information about signing is displayed in the tooltip and also in a dialog which opens when you double-click a package. ● *“Allow unsigned and self-signed packages”* : The package should be installed although it is unsigned or self-signed. After the package is selected, the installation wizard opens with the dialogs: ● *“Installation - License Agreement”* In this dialog, CODESYS also displays the *“Checksum”* of the package. Displayed only when the package has a license agreement. ● *“Choose Setup Type”* The options are package-dependent. – *“Complete setup”*: CODESYS installs all components – *“Typical setup”*: CODESYS installs a standard set of components as defined in the package – *“Custom setup”*: CODESYS installs those components which are selected in a dialog ● *“Installation - Target Versions”*: You select which of the existing target versions should be updated by the package installation. You have to select at least one version profile. When this dialog is successfully completed, the selected package is ready for installation. You have to close all CODESYS instances in order for the package installation to be automatically started and run.
“Uninstall ”	Uninstalls the selected package ● When the *“Display versions”* option is not selected, CODESYS uninstalls all versions of the selected package. ● When the *“Display versions”* option is selected and you have selected a package node on the top level, CODESYS uninstalls all versions of the selected package. ● When the *“Display versions”* option is selected and you have selected an individual package version, CODESYS uninstalls exactly this version. When this dialog is closed, all CODESYS instances have to be closed in order for the package uninstallation to start.

"Details"	Opens the *"Details"* dialog for the selected package with the following tabs: ● *"Package Details"* – *"Name"*: Package name – *"Version"* – *"Checksum"*: SHA-1 checksum of the package – *"Vendor"* – *"Copyright"* – *"Description"* – *"Installation date"* ● *"License Agreement"* ● *"Installations Log"*
"Search updates in background"	☑: CODESYS automatically searches for updates every time the programming system is started and then one time every hour.
"Display versions"	☑: Displays all versions of the installed package.

You can compare the "Checksum" with the package checksum from the package vendor. CODESYS displays this checksum in the "Details" dialog and in the "Installation - License Agreement" dialog of the installation wizard. You do this to make sure that you have installed an original package.

If you have installed a newer version of the programming system in the same installation directory as the previous version, the license information about the previously installed package remains unchanged and CODESYS displays the information in the "Package Manager" dialog.

Table 141: "Updates"

"Search Updates"	Searches for the selected package on your system and in the CODESYS Store Updates. CODESYS displays the found updates in the *"Update Info"* column of the package list.
"Download"	Installs the update package from the *"Download Package"* dialog. In the *"Download Package"* dialog, click the *"Download and Installation"* button for this.

Table 142: "CODESYS Store"

"Rating"	Give an rating of the package
"CODESYS Store"	Link to the homepage of the Store

Command 'Library Repository'

Symbol:

Function: The command opens the *"Library Repository"* dialog. In this dialog you define which libraries are installed on the local system and are thus available for your application.

Call: Menu bar: *"Tools"*

Dialog 'Library Repository'

Table 143: “Location”

Display of the directory on the local system in which the library files are located. The libraries in this *“Location”* are listed in the *“Installed libraries”* area.	
“Edit Locations”	Opens the *“Edit Repository Locations”* dialog.

1. *You can only use empty directories for new repositories.*
2. *You can also use existing repositories as locations.*
3. *The "System" repository is not editable; CODESYS indicates this by the italic lettering of the entry.*

Table 144: Dialog “Edit Repository Locations”

List of the repositories with *“Location”* and *“Name”*	
“Add”	Creates a new repository. Opens the *“Repository Location”* dialog. The selected directory (*“Location”* input field) has to be either empty or an existing valid repository. *“Name”* is the input field for a symbolic repository name.
“Edit”	Opens the *“Repository Location”* dialog (see *“Add”*)
“Remove ”	A dialog box open, asking whether only the entry should be removed from the list of repositories, or whether the directory with the library files should be deleted from the file system. If you want to delete the directory, you have to confirm this.

Table 145: “Installed libraries”

List of the libraries in a tree structure. Display of each library with category, name, company and version. The icon to the left of the name indicates whether the library is digitally signed or unsigned.	
“Company ”	List box for filtering the displayed libraries.
“Install”	Opens the *“Select Library”* dialog. Possible filters: ● *“Compiled CODESYS library files (* .compiled-library)”*. ● *“Compiled CODESYS library files (* .compiled-library-v3)”* ab V3 SP15 ● *“Library files (*.library)”* for still uncompiled library projects ● *“All files (*.*)”*
“Uninstall”	Uninstalls the selected library.
“Export”	Opens the default dialog for saving the library project to the local file system. The file type is `Library files (*.library)`, `Compiled library files (*.compiled-library)`, or `Compiled library files (*.compiled-library-v3)`.
“Find”	Searches for libraries and function blocks. Opens the *“Find Library”* dialog. When you enter a string in the input field, CODESYS displays the libraries that it finds with a corresponding string.
“Details”	Opens the *“Details”* dialog with details from the project information of the library for the selected version of a library. You find the following information by clicking *“More”* in the *“Details”* dialog: ● *“Size”*: Specified in bytes ● *“Created”*: Creation date ● *“Changed”*: Date of the last change ● *“Last access”*: Date ● *“Attributes”* ● *“Properties”*

“Dependencies”	For the selected library, the *“Dependencies”* dialog opens, showing the dependencies on other libraries. *“Title”*, *“Version”* and *“Company”* are shown for each library reference. References that function via placeholders are displayed according to the syntax: #<placeholder name>.
“Group by category”	● ☑: Grouping by library category ● ☐: Alphabetical sorting The categories are defined by external description files '*.libcat.xml'.

Table 146: “Library Profiles”

A library profile defines the library version with which CODESYS resolves a library placeholder if a certain compiler version is set in the project.	
“Import”	Imports a `*.libraryprofile` file. If the import already contains existing placeholder entries, a query appears asking whether CODESYS should overwrite it.
“Export”	Exports an xml file with the extension `*.libraryprofile` with the assignments of the selected placeholder entries; you can only select a single entry of a *“Compiler version”*.

Placeholder resolutions can also be defined in the target device currently in use and even by a specific local specification in the Placeholders dialog in the Library Manager.

See also

- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*
- *Chapter 1.4.1.20.2.21 “Object 'Project Information'” on page 919*
- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*
- *Chapter 1.4.1.20.4.2 “Dialog 'Library Reference Conversion'” on page 1150*
- *Chapter 1.4.1.16.4 “Exporting library files” on page 451*

Command 'License Manager'

Symbol: 

Function: This command opens the wizard for configuring licenses for CODESYS add-on products. The wizard starts with the *“License Manager - Select target”* dialog.

Call: Menu bar: *“Tools”*.

The License Manager can handle licenses for CODESYS add-on products on the local computer, as well as licenses for RTS add-on products on devices. It supports both the installation in a soft container and on a dongle.

Dialog 'License Manager - Select Target'

This is the start dialog of the License Manager wizard. Here you decide where the license will be installed.

“Workstation”	Local computer
“Device”	Controller. The connection to this device must be configured correctly in order to license (*“Communication Settings”* tab of the device editor).

After clicking *“Next”*, you decide the container where you want to manage the licenses.

Dialog 'License Manager - Select container'

“Dongle”	A corresponding dongle must be connected to the computer or device. Not all devices support dongles.
“Soft container”	"CODESYS Security Key. A corresponding soft container must be registered in the CodeMeter Control Center. The CODESYS installation provides an existing soft container.

If you are installing a product on your local computer (*“Workstation”*), then the *“License Manager”* opens immediately for the specific selection of the dongle or soft container, and the next actions. This happens after you choose the container type and click *“Next >”*.

If you are licensing the add-on product for a controller, then first the dialog opens for selecting the device in the network after you click *“Next >”*. This dialog corresponds to the classic view of the *“Communication Settings”* tab of the device editor.

Dialog 'License Manager'

“Container”	Depending on whether *“Dongle”* or *“Soft container”* was selected: Drop-down list of all CODESYS dongles or soft containers that were found on the computer or device.
“Products”	List of all installed products that are subject to licensing. A prepended symbol indicates the existence and validity of the license. On the right side of the window, the following information is displayed for the selected product and corresponding licenses: *“Name”* *“Company”* *“Unit counter”* *“License quantity”* *“Usage period”* *“Feature map”* *“Activation time”* *“Expiration time”* *“Firm code”* *“Product code”* *“Description”*

“Install Licenses”	Opens the dialog *“Install licenses on <computer> - Select Operation”*: ● *“Activate license”*: Opens the dialog *“Install licenses on <computer> - Activate License”* (see more below) ● *“Request license”*: Opens the dialog *“Install licenses on <computer> - Request License”* (see more below) ● *“Install license”*: Opens the dialog *“Install licenses on <computer> - Install License”* (see more below)
“Additional Functions”	Opens the menu with the following actions: ● *“Return license”*: Opens the *“Return Licenses”* (see more below) ● *“Restore license”*: This function is available in the case of device licensing only. Opens the *“Restore Licenses”* dialog (see more below)

Table 147: “Install Licenses on <computer> - Activate License”

This is the recommended way to activate a license available via the License Server when you have an Internet connection. Requirement: The computer has an Internet connection.	
“Ticket ID”	Input field for the ticket ID that you received from the software vendor. The ticket ID consists of five sets of five alphanumerical characters (for example: `LYSQ3-ZU93K-24LWC-XGWJ8-5AY7H`).
“License server”	Drop-down list of the license server that provides the license for activating the product. You receive the server URL from the software vendor.
“Select Ticket from Repository”	Opens the *“License Repository”* dialog.
“Next”	CODESYS connects to the license server. ● If the specified ticket contains only one license, then a dialog opens to confirm the successful activation after completion of the server action. ● If the specified ticket contains multiple licenses, then the dialog *“Install licenses - Select Licenses”* opens with a list of these licenses (see description below).

Table 148: “Install Licenses - Select Licenses”

Selection of the licenses to be activated for the ticket which you specified in the dialog *“Install Licenses - Activate License”*.	
“Name”	Product name
“Available”	Number of available licenses
“Used”	Number of used licenses
Total	Sum of all used and available licenses
Next	CODESYS connects to the license server. After successful completion of the server action, a dialog opens with the confirmation of the activation.

Table 149: “Install Licenses on <computer> - Request License”

If the computer does not have an Internet connection, then you can generate a context file from this dialog. The file "WibuCmRaC" is then transmitted to the license server via an Internet-enabled computer. When activation is complete, a license update file "WibuCmRaU" is provided for download.	
“Software vendor”	Input field for firm codes from the software vendor that provided the license for activating the product. As an alternative, you can select the software vendor from the drop-down list.
“Context file”	Location and name

Table 150: “Install Licenses on <computer> - Install License”

If you downloaded a license update file from the Internet during software activation, then you can use this dialog to install the license on your dongle. To do this, specify the path of the license update file in the input field.

Table 151: “Return License”

If the license permits, you can return it in order to reactivate it later on another computer.	
“Ticket ID”	Field for specifying the ticket ID that was used for licensing.
“License server”	Drop-down list for selecting the license server that provides the license for activating the product. You receive the server URL from the software vendor.
“Load License(s)”	Button for showing all current licenses installed for the given ticket ID on the server in the *“Licenses”* window.

“Licenses”	List of licenses available on the server for the given ticket ID. The following information for the selected license is displayed next to the window on the right: ● *“Name”* ● *“Number of activations”* ● *“Return allowed”* ● *“Activation type”* ● *“Activation date”* ● *“Firm code”* ● *“Comment”*
“Return License(s)”	Button for returning the selected license(s). These can be reactivated later on another system.

Table 152: “Install Licenses - Restore Licenses”

When activated, device licenses are saved to a file (`*.WibuCmRau`) on the local computer and in the "CODESYS Central License Server". If lost, they can be restored from this file to the identical device.	
“Ticket ID”	Field for specifying the ticket ID that was used for licensing that has already occurred.
“Restore”	If a corresponding license backup file is found, then the license is reactivated in the device.

See also

- *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*
- *Chapter 1.4.1.20.3.8.7 “Command ‘License Repository’” on page 1066*

Command ‘License Repository’

Symbol: 

Function: This command opens the dialog box *“License Repository”* for viewing information about the individual licenses.

Call: Main menu *“Tools”*

Requirements: CODESYS is in offline or online mode.

In the license repository, after entering the ticket number, you can obtain information about the licenses concerned from the central license server.

To do this you can paste the ticket number(s) from the clipboard or import it/them from a text file.

Table 153: “Tickets”

<List of the ticket IDs imported into the repository for components requiring licenses>	
“Licenses”	If you select an entry in the list of tickets, the name and the status of the licensed component are displayed here. : License available and valid : License found, but invalid : License not found In the right-hand part of the dialog box you will then receive then the following information about this license:
	“Name”: name of the product to be licensed *“Item number”*: item number in the license server. *“Return allowed”*: It is possible to have this license deactivated so that it can be re-activated on another system. *“Can be activated”*: you can have the license activated via the license manager. *“Activation quantitiy”*: number of activations that have taken place so far. *“Activation date”*: date of the current activation *“Container serial ”* *“Firm codes”* *“Comment”*

“Import Tickets”	The standard dialog box for browsing the local file system appears. If you open a text file containing one or more “tickets”, i.e. license numbers, these are imported into the repository. Alternatively you can also insert the numbers from the clipboard into the list.

Command 'Device Repository'

Symbol:

Function: This command opens the *“Device Repository”* dialog. This dialog is used for managing the devices that are installed on the local system and can be integrated into CODESYS projects.

Call: Menu bar: *“Tools”*.

Dialog 'Device Repository'

CAUTION!

Do not change the internal device repository manually. Do not copy any files to or from the repository. Always use the device repository dialog to install or uninstall devices.

Table 154

“Location”	Shows the device repository directory on the local system. The list box shows the currently set save locations. By default, CODESYS creates the system repository during installation. The devices of the selected location are listed in the *“Installed device descriptions”* field.
“Edit Locations”	Opens the *“Edit Repository Locations”* dialog.

Table 155: Dialog 'Edit Repository Locations'

List of the repositories with *“Location”* and *“Name”*.	
“Add”	Creates a new repository. Opens the *“Repository Location”* dialog. The selected directory (*“Location”* input field) must be empty or it must be a valid repository.
“Edit”	Opens the *“Repository Location”* dialog (see *“Add”*).
“Remove ”	A dialog prompt opens for you to decide whether the respective directory should also be deleted from the hard disk.

Table 156: “Installed Device Descriptions”

List of device descriptions in multilevel tree structure. Shows all device descriptions with *“Name”*, *“Vendor”*, and *“Version”*. The top nodes represent device categories, for example PLCs, fieldbuses, and logical devices.	
“String for full-text search in all devices”	This field is editable after clicking in it. For any character string entered, only those devices that include the character string are displayed in the lower view. The matched string is highlighted in yellow for these devices.
“Vendor”	Drop-down list with manufacturers whose available devices are displayed.
“Install”	Opens the *“Install Device Description”* dialog. For the default devices with file type "*.devdesc.xml". You can also select manufacturer-specific description files, such as "*.gsd" files for PROFIBUS DP modules, "*.eds" and "*.dcf" files for CAN devices. When you click *“OK”* to confirm the selection, CODESYS inserts the new device into the device repository. If an error occurs during installation (for example, missing files that are referenced by the device description), then CODESYS reports the error to the lower part of the device repository dialog.
“Uninstall”	Removes the selected device. If you delete the device from the device repository, then it is no longer available for use in the programming system.
“Renew Device Repository”	Updates all devices in the device repository. When new versions of import plug-ins are available, some device descriptions may be outdated. The affected devices are marked with a warning symbol (⚠). This command opens a dialog to confirm the update.
“Download Missing Device Descriptions”	Opens when you use devices in your project that are not available in the device repository. When you execute this command, a list of missing devices is displayed. There you can select the corresponding devices for download.
“Details”	Opens the *“Details”* dialog for the selected device description. This dialog provides additional information from the device description file.

NOTICE!
During installation, CODESYS copies the device description files and all additional reference files to an internal location. Therefore, any changes to the original files no longer influence the installed devices. You must reinstall the devices to make any changes effective. We recommended that you change the internal version number of a device description after a modification.

See also

- *Chapter 1.4.1.17.1 “Installing devices” on page 452*
- *Chapter 1.4.1.20.4.13.5 “Dialog 'Options' – 'Device Description Download'” on page 1190*

Command 'Create Device list CSV'

Symbol:

Function: MS Excel template of device list for device import is opened.

Call: Main menu *“Tools”*, Context menu

Requirement: -

Chapter 1.8.1.3.2 “Creating CSV device list” on page 4119

Command 'Multi Online Change'

Function: The MultiOnlineChange tool/plug-in for Automation Builder enables firmware update, download and online change of the same project to several AC500 V2 PLCs.

Call: Main menu *“Tools”*, Context menu

Requirement: -

Command 'Device ECAD data'

Function: Automation Builder provides an ECAD interface for exchanging the PLC configuration data with EPLAN Electric P8 and Zuken E3. This feature removes double data entry between electrical engineering in the ECAD tool and the control logic programming in Automation Builder by synchronizing the PLC hardware including topology and I/O signals between these tools.

Call: Main menu *“Tools”*, Context menu

Requirement: A project is open.

Chapter 1.8.1.1 “Exporting and importing ECAD data (PBF)” on page 4112

Command 'OPC UA Information Model Repository'

Function: The command opens the *“OPC UA Information Model”* dialog. The OPC UA information models, which are installed on the local system and can be integrated in CODESYS projects, are managed in the dialog.

Call: Menu bar: *“Tools”*

Table 157: Dialog 'OPC UA Information Model'

"Location"	Displays the OPC UA information model directories on the local system. The list box shows the currently set locations. By default, CODESYS creates the system repository during installation. The information models of the selected location are listed in the *"Installed OPC UA information models"* area.
"Edit Locations"	Opens the *"Edit Repository Locations"* dialog.
"Installed OPC UA information models"	
List of installed information models. Double-click to open installed information model documentation. Note: The information models of this repository can also be added to project archives.	
"Install "	Opens the *"Select Installed OPC UA Information Model(s)"* dialog. ● File type: OPC UA Information Models `*NodeSet2.xml` (example: *"Informationmodel.NodeSet2.xml"*. When you click *"Open"*, the selected information model is inserted in the repository. ● File type: All files `*.*`: You can select an OPC UA documentation, for example, in PDF or Word format. When you click *"Open"*, the *"Assign Documentation OPC UA Information Models"* opens. For a description of the dialog, see below.
"Uninstall"	Uninstalls the selected OPC UA information model. When you delete the information model from the repository, it is no longer available in the development system for use in the CODESYS Development System.
"Details"	Opens the *"Details"* dialog for the selected information model. The dialog includes additional information about the information model. In *"Alias"*, you can specify an alias name for the URI. Moreover, information is displayed as to whether or not a documentation for the information model is available. ● *"Model URI"* ● *"Publication date"* ● *"Publisher"* ● *"Repository"* ● *"Alias"* ● *"Documentation available"*: – *"Yes"*: The *"Uninstall documentation"* button is available. – *"No"*: The *"Install documentation"* button is available. ● *"Install documentation"*: Opens the *"Select OPC UA Information Model Documentation"* dialog. The data type `OPC UA Information Model Documentation (*.pdf)` is set as default in the dialog.
"Documentation"	Opens the installed documentation for the selected information model. If no documentation is installed for the selected information model, then the command is disabled.
"Display all versions"	All installed versions of the information model are displayed in a tree structure.

See also
- ↳ *Chapter 1.4.1.20.2.15 "Object 'OPC UA Information Model'" on page 877*

Command 'Scripting' - 'Execute Script File'

Symbol: 

Function: This command opens a dialog for selecting and then executing the script file (*.py).

Call: Menu bar: *"Tools ➔ Scripting"*.

Command 'Scripting' - 'Enable Script Tracing'

Symbol: 

Function: This command makes CODESYS print all commands from the script file to the message view. Use this command for monitoring and debugging scripts. A blue frame around the symbol indicates that the option is active.

Call: Main menu *“Tools ➔ Scripting”*.

Command 'Scripting' - 'Scripts'

Function: This command executes a script that is stored in the `ScriptDir` folder.

Call: Menu bar: *“Tools ➔ Scripting ➔ Scripts”*.

Requirement: The `ScriptDir` folder exists in the CODESYS installation directory. Python scripts are stored in this folder with the file extension `.py`.

All scripts that are contained in the `ScriptDir` folder are executable as menu commands and are sorted alphabetically by file name.

Command 'Customize'

Function: This command opens the *“Customize”* dialog box, where you can customize the menus, toolbars, and keyboard shortcuts according to your individual requirements.

Call: Main menu *“Tools”*

Command 'Options'

Function: The command opens the dialog box *“Options”* for the configuration of the CODESYS options. These options define the behavior and appearance of the CODESYS user interface. CODESYS saves the settings in your current user profile on your local system. The current profile specifies the standard settings.

Call: *“Tools”* menu

See also

- *Chapter 1.4.1.20.4.13.22 “ Dialog 'Options' - 'SFC Editor'” on page 1200*
- *Chapter 1.4.1.20.4.13.3 “Dialog 'Options' - 'CFC Editor'” on page 1189*
- *Chapter 1.4.1.20.4.13.4 “Dialog 'Options' – 'Declaration Editor'” on page 1190*
- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*
- *Chapter 1.4.1.20.4.13.5 “Dialog 'Options' – 'Device Description Download'” on page 1190*
- *Chapter 1.4.1.20.4.13.9 “Dialog 'Options' - 'FBD, LD, and IL'” on page 1192*
- *Chapter 1.4.1.20.4.13.13 “Dialog 'Options' – 'International Settings'” on page 1195*
- *Chapter 1.4.1.20.4.13.14 “Dialog 'Options' – 'Libraries'” on page 1195*
- *Chapter 1.4.1.20.4.13.15 “Dialog 'Options' – 'Library Download'” on page 1195*
- *Chapter 1.4.1.20.4.13.16 “Dialog 'Options' – 'Load and Save'” on page 1196*
- *Chapter 1.4.1.20.4.13.19 “Dialog 'Options' - 'PLCopenXML'” on page 1198*
- *Chapter 1.4.1.20.4.13.20 “Dialog 'Options' - 'Proxy Settings'” on page 1198*
- *Chapter 1.4.1.20.4.13.21 “Dialog 'Options' - 'Refactoring'” on page 1199*
- *Chapter 1.4.1.20.4.13.23 “Dialog 'Options' - 'SmartCoding'” on page 1201*
- *Chapter 1.4.1.20.4.13.25 “Dialog 'Options' - 'Text Editor'” on page 1203*

Command 'Import and Export Options'

Function: This command opens the *"Import and Export Options"* dialog. Here you can configure the export and import of selected settings of the CODESYS options. The settings are saved to an XML file with the default extension (`options.xml`).

Call: Menu bar: *"Tools"*.

Dialog 'Import and Export Options'

"Export selected options"	*"Select options"*: In the table, you can select the categories of options, either user-specific or machine-specific (computer), whose current settings are to be exported to the XML file. *"File"*: Path of the export file in the local file system. Example: `D:\system1.options.xml`. Button ...: Opens the default dialog to search for an existing file in the local file system, or to create one. The *"File type"* `option export (*.options.xml)` is preset.
"Import selected options"	*"File"*: Path of the options export file whose contents are to be imported. Button ...: Opens the default dialog to search for an existing file of type `option export (*.options.xml)` in the local file system. After you click *"OK"* to close the dialog, the settings described in the file are applied to the project.

See also

- *Chapter 1.4.1.1.1 "Setting CODESYS options" on page 180*

Command 'Device Reader'

Function: The command opens the standard *"Select Device"* dialog and reads the license and product information of the selected controller. This license and product information is displayed in the *"Device Reader"* dialog.

Call: Menu bar: *"Tools"*

Requirement: No applications exist on the controller.

> *If the command is selected although an application exists on the controller, then a dialog prompts the user whether or not all applications should be removed from the controller. When the user click "No" to this dialog, the "Device Reader" command is aborted.*

Table 158: Dialog "Device Reader"

"Status of Available Device Features"	
"Product"	CODESYS product (example: `SoftMotion`)
"Feature"	Feature of *"Product"* Example: `CNC` is a *"Feature"* of `SoftMotion`.
"License Active/Count "	`Yes`: A license exists for the feature. `No`: A license does not exist for the feature. *"Count"*: Number of licenses

Menu 'Window'

Command 'Next Editor'

Keyboard shortcut: *[Ctrl]+[F6]*

Function: This command switches focus from the currently active view to the next view. The next view is identified by the tab to the right of the currently active tab.

Call: Main menu *“Window”*

Requirement: At least one object is open.

See also

- *Chapter 1.4.1.20.3.9.2 “Command 'Previous Editor'” on page 1073*

Command 'Previous Editor'

Keyboard shortcut: *[Shift]+[Ctrl]+[F6]*

Function: This command switches focus from the currently active view to the previous view. The previous view is identified by the tab to the left of the currently active tab.

Call: Main menu *“Window”*

Requirement: At least one object is open.

See also

- *Chapter 1.4.1.20.3.9.1 “Command 'Next Editor'” on page 1073*

Command 'Close All Editors'

Symbol: 

Function: This command closes all currently open editor views.

Call: Main menu *“Window”*

Requirement: At least one editor is open.

See also

- *Chapter 1.4.1.20.3.9.4 “Command 'Close All Editors of Inactive Applications'” on page 1074*
- *Chapter 1.4.1.20.3.9.16 “Command 'Close All Editors But This'” on page 1077*

Command 'Close All Editors of Inactive Applications'

Function: This command closes all editor views for objects that are located directly below a currently inactive application. Object editors in the POU view remain open.

Call: Main menu *“Window”*

Requirement: At least one object of an inactive application is open.

See also

- *Chapter 1.4.1.20.3.9.3 “Command 'Close All Editors'” on page 1073*
- *Chapter 1.4.1.20.3.9.16 “Command 'Close All Editors But This'” on page 1077*

Command 'Reset Window Layout'

Function: This command resets all currently open windows and views to their default positions. You are prompted for a confirmation before the command is executed.

Call: Main menu *“Tools”*

Command 'New Horizontal Tab Group'

Symbol: 

Function: This command moves the currently active view to a new, separate tab group below the existing one.

Call: Main menu *“Window”* or context menu of the tab

Requirement: Several editor views are open as tabs next to each other.

If you open another object in the editor, then this is automatically included in the tab group that is currently in focus.

See also

- *Chapter 1.4.1.20.3.9.7 “Command 'New Vertical Tab Group'” on page 1074*

Command 'New Vertical Tab Group'

Symbol: 

Function: This command moves the currently active view to a new, separate tab group to the right of the existing one.

Call: Main menu *“Window”* or context menu of the tab

Requirement: Several editor views are open as tabs next to each other.

If you open another object in the editor, then this is automatically included in the tab group that is currently in focus.

See also

- *Chapter 1.4.1.20.3.9.6 “Command 'New Horizontal Tab Group'” on page 1074*

Command 'Float'

Function: This command releases a docked view from its frame in the user interface and repositions it on the screen as a floating window.

Call: Main menu *"Window"*

Requirement: The application is in online mode.

This window can then be positioned outside of the user interface. Use the *"Dock"* command to return a floating window to the frame of the user interface.

See also

- *Chapter 1.4.1.20.3.9.9 "Command 'Dock'" on page 1075*

Command 'Dock'

Function: This command returns a floating window, which was released by the *"Float"* command, to the frame of the user interface.

Call: Main menu *"Window"*

See also

- *Chapter 1.4.1.20.3.9.8 "Command 'Float'" on page 1075*

Command 'Auto Hide'

Keyboard shortcut: *[F7]*

Function: This command shows or hides a view.

Call: Main menu *"Window"*

Hide simply means that CODESYS minimizes the view to a tab at the bottom of the user interface which is visible only when you move the mouse over the tab. The command functions like a check box. When a window is hidden, the check box is selected in the menu. When you click the command again, the checkbox is cleared and the window is shown.

Command 'Next Pane'

Keyboard shortcut: *[F6]*

Function: This command sets the focus on the next pane.

Call: Main menu *"Window"*

Requirement: An object is open that contains two or more panes.

Example: If an object is open in the ST editor and the cursor is currently in the declaration section, then command sets the focus to implementation section.

See also

- *Chapter 1.4.1.20.3.9.12 "Command 'Previous Pane'" on page 1075*

Command 'Previous Pane'

Keyboard shortcut: *[Shift]+[F6]*

Function: This command sets the focus on the previous pane.

Call: Main menu *"Window"*

Requirement: An object is open that contains two or more panes.

Example: If an object is open in the ST editor and the cursor is currently in the declaration section, then command sets the focus to implementation section.

See also

- *Chapter 1.4.1.20.3.9.11 “Command 'Next Pane'” on page 1075*

Command 'Toggle First Pane'

Keyboard shortcut *[Alt]+[F6]*

Function: This command shows and hides the declaration view.

Call: *“Window”*.

Requirement: The cursor is positioned in the editor of one of the following objects:

- POU
- Transition
- Method
- Get accessor method of a property
- Set accessor method of a property
- Visualization

You can also toggle the subviews by means of the buttons.

See also

- *Chapter 1.4.1.20.3.9.14 “Command 'Toggle Second Pane'” on page 1076*

Command 'Toggle Second Pane'

Function: This command shows and hides the implementation view.

Call: *“Window”*.

Requirement: The cursor is positioned in the editor of one of the following objects:

- POU
- Transition
- Method
- Get accessor method of a property
- Set accessor method of a property
- visualization

You can also toggle the subviews by means of the buttons.

See also

- *Chapter 1.4.1.20.3.9.13 “Command 'Toggle First Pane'” on page 1076*

Command 'Windows'

Function: This command opens the *“Windows”* dialog box, which lists all open objects. You can then activate or close any of the listed views.

Call: Main menu *“Window”*

Command 'Close All Editors But This'

Function: This command closes all editor views except the currently open one.

Call: Right-click the tab

Requirement: At least two objects are open.

See also

- *Chapter 1.4.1.20.3.9.3 “Command 'Close All Editors'” on page 1073*
- *Chapter 1.4.1.20.3.9.4 “Command 'Close All Editors of Inactive Applications'” on page 1074*

Command 'Select Object in Navigator'

Function: This command selects the object of the active editor in the device tree.

Call: Right-click the tab

Requirement: At least one object is open.

This command is executed automatically when you select the “Track active editor” option for the device tree.

See also

- *Chapter 1.4.1.20.3.9.18 “Command 'Select Parent Object in Navigator'” on page 1077*

Command 'Select Parent Object in Navigator'

Function: This command selects the parent object in the device tree.

Call: Right-click the tab

Requirement: At least one object is open.

See also

- *Chapter 1.4.1.20.3.9.17 “Command 'Select Object in Navigator'” on page 1077*

Commands of the Submenu 'Window'

Function: The command activates the selected window.

Call: Main menu *“Window”*

For each opened editor window the menu *“Window”* contains a command *“<n><object name>”*. Choosing this command activates the corresponding window. In offline mode CODESYS adds the extension *“(Offline)”*. To differentiate between the implementation or the instances of a function block the extension *“(Impl)”* or *“<instance path>”* is added.

Menu 'Help'

Command 'Contents'

Symbol:  ; keyboard shortcut: *[Ctrl]+[Shift]+[F1]*

Function: This command opens the CODESYS help.

Call: Menu bar: *“Help”*.

Command 'Index'

Symbol:  ; keyboard shortcut: *[Ctrl]+[Shift]+[F2]*

Function: This command opens the CODESYS help.

Call: Menu bar: *“Help”*.

An index search is not possible in the online help. The *“Index”* tab opens in the offline help.

All index entries of the help are listed alphabetically in the index view.

“ Look for ”	As you type letters into the input field, CODESYS searches automatically for matches in the index list.
“Display”	Opens the help page for the highlighted index entry in the list and displays the title of the help page and location of the help file (*.chm) in the *“Index results for <index entry>”* view. When several pages are found and then displayed in this view, then you view a specific help page by clicking its entry in the list. Clicking an entry in the index list achieves the same result.

Command 'Find'

Symbol: 

Function: This command opens the CODESYS help.

Call: Menu bar: *“Help”*.

In the online help, you can run a full-text search from the input field on the top right of the help page. The *“Find”* tab opens in the offline help.

Table 159: Tab 'Search'

“Search for”	Combo box for defining the search term or for selecting the 25 most recent search terms.
“ Search in titles only”	The search is performed only in the titles of the help pages.
“Display partial matches”	Displays terms also as search results that include the search term.
“Limit to .... matches ”	Limits the number of search results. Maximum value: 1000
“Find”	Starts the full-text search.

Command 'About'

Function: This command opens a splash screen with information about the CODESYS version and copyright. In addition, buttons are available for detailed information about the version, license, and acknowledgments.

Call: Main menu *"Help"*.

"Version Info"	Opens the *"Detailed Version Information"* dialog box with a list of CODESYS components and information about the operating system. *"Export"*: Exports the detailed version information as a *.txt file or in any other format.
"License Info"	Opens the *"License Information"* dialog box. ● *"Plug-in"*: Drop-down list for the plug-in to display the license information ● *"Software License"*: License information about selected *"Plug-in"*
"Acknowledgments"	

Menu 'SFC'

Command 'Init Step'

Symbol: 

Function: This command converts the selected step into an initial step.

Call: Main menu *"SFC"*

After you choose this command, the borders of the step element change to a double line. The previous initial step is automatically displayed as a normal step with a single-line border.

You can also activate and deactivate the property *“Init step”* in the properties dialog of a step. However, CODESYS does not automatically adjust the settings of other steps.

This command is useful if you want to convert a diagram. When you create a new SFC object, it automatically includes an initial step followed by a transition (TRUE) and a jump back to the initial step.

Please note: In online mode, it is possible to reset the diagram to the initial step using the `SFCInit` and `SFCReset` flags.

See also

- *Chapter 1.4.1.19.1.4.6 “SFC Flags” on page 481*
- *Chapter 1.4.1.19.1.4.8.6 “SFC element properties” on page 493*

Command 'Insert Step'

Symbol:

Function: This command inserts a step before the selected point.

Call: Menu bar *“SFC”*; context menu in SFC editor

The new step is named `Step<n>` by default, where `n` is an incremental number starting at `0` for the first step that is inserted in addition to the initial step. The name can be edited by clicking on it.

See also

- *Chapter 1.4.1.20.3.11.7 “Command 'Insert Step-Transition After'” on page 1081*
- *Chapter 1.4.1.20.3.11.1 “Command 'Init Step'” on page 1079*
- *Chapter 1.4.1.19.1.4.8.1 “SFC elements 'Step' and 'Transition'” on page 486*

Command 'Insert Step After'

Symbol:

Function: This command inserts a step after the selected point.

Call: Menu bar *“SFC”*; context menu in SFC editor

The new step is named `Step<n>` by default, where `n` is an incremental number starting at `0` for the first step that is inserted in addition to the initial step. The name can be edited by clicking on it.

See also

- *Chapter 1.4.1.20.3.11.7 “Command 'Insert Step-Transition After'” on page 1081*
- *Chapter 1.4.1.20.3.11.1 “Command 'Init Step'” on page 1079*
- *Chapter 1.4.1.19.1.4.8.1 “SFC elements 'Step' and 'Transition'” on page 486*

Command 'Insert Transition After'

Symbol:

Function: This command inserts a transition after the selected point.

Call: Menu bar *“SFC”*; context menu in SFC editor

The new transition is named `Trans<n>` by default, where `n` is an incremental number beginning at `0` for the first transition. The name can be edited by clicking on it.

See also

- *Chapter 1.4.1.20.3.11.7 “Command 'Insert Step-Transition After'” on page 1081*
- *Chapter 1.4.1.19.1.4.8.1 “SFC elements 'Step' and 'Transition'” on page 486*

Command 'Insert Transition'

Symbol:

Function: This command inserts a transition before the selected point.

Call: Menu bar *“SFC”*; context menu in SFC editor

The new transition is named `Trans<n>` by default, where `n` is an incremental number beginning at `0` for the first transition. The name can be edited by clicking on it.

See also

- *Chapter 1.4.1.20.3.11.7 “Command 'Insert Step-Transition After'” on page 1081*
- *Chapter 1.4.1.19.1.4.8.1 “SFC elements 'Step' and 'Transition'” on page 486*

Command 'Insert Step-Transition'

Symbol: 

Function: This command inserts a step and a transition before the selected point.

Call: Main menu *“SFC”*

If you have selected a step, then CODESYS inserts a new step-transition combination. If you have selected a transition, then a new transition-step combination is inserted.

The new step is named `Step<n>` by default, where `n` is an incremental number beginning at `0` for the first step that was inserted in addition to the initial step. The new transition is named `Trans<n>` by default. You can edit the default names directly by clicking the names.

See also

- *Chapter 1.4.1.20.3.11.7 “Command 'Insert Step-Transition After'” on page 1081*
- *Chapter 1.4.1.20.3.11.1 “Command 'Init Step'” on page 1079*
- *Chapter 1.4.1.19.1.4.8.1 “SFC elements 'Step' and 'Transition'” on page 486*

Command 'Insert Step-Transition After'

Symbol: 

Function: This command inserts a step and a transition after the selected point.

Call: Main menu *“SFC”*

If you have selected a step, then CODESYS inserts a new transition-step combination. If you have selected a transition, then a new step-transition combination is inserted.

The new step is named `Step<n>` by default, where `n` is an incremental number beginning at `0` for the first step that was inserted in addition to the initial step. The new transition is named `Trans<n>` by default. You can edit the default names directly by clicking the names.

See also

- *Chapter 1.4.1.20.3.11.6 “Command 'Insert Step-Transition'” on page 1081*

Command 'Add Entry Action'

Symbol: 

Function: This command opens the *“Add Entry Action”* dialog box where you define a new entry action. Depending on the SFC options, a dialog prompt may open for selecting the duplication mode for the new step action.

Call: Menu bar: *“SFC”*; context menu of the selected step element.

Requirement: A step element in SFC is selected.

The entry action is opened automatically in the ST editor. The step element contains an `E` in the lower left corner.

Confirmation prompt for selecting the duplication mode

Options:

- *“Copy reference: A new step will call the same actions”*: If the step is copied in SFC, the link to the step action(s) is also copied. The steps copied by each other will therefore call the same actions.
- *“Copy implementation: New action objects are created for a new step.”*: This means that the step actions for a copied step are embedded. By default, the generated action objects appear below an SFC box in the device tree or *“POUs”* view. These objects contains a copy of the original implementation code of the respective action.

  The display of the embedded objects can be activated and deactivated in the tree by means of the *“Show Embedded Objects”* and *“Hide Embedded Objects”* commands in the context menu of an SFC object.

See also

- *Chapter 1.4.1.20.4.13.22 “ Dialog 'Options' - 'SFC Editor'” on page 1200*
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.19.1.4.8.2 “SFC Element 'Action'” on page 488*
- *Chapter 1.4.1.20.3.11.25 “Command 'Do Not Display Embedded Objects'” on page 1088*

Command 'Add Exit Action'

Symbol: 

Function: This command opens the *“Add Exit Action”* dialog box where you define a new exit action. Depending on the SFC options, a dialog prompt may open for selecting the duplication mode for the new step action. For more information, refer to the help page for the *“Add Exit Action”* command.

Call: Menu bar: *“SFC”*; context menu of the selected step element.

Requirement: A step element in SFC is selected.

See also

- *Chapter 1.4.1.20.3.11.8 “Command 'Add Entry Action'” on page 1082*
- *Chapter 1.4.1.20.4.13.22 “ Dialog 'Options' - 'SFC Editor'” on page 1200*
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.19.1.4.8.2 “SFC Element 'Action'” on page 488*
- *Chapter 1.4.1.20.3.11.25 “Command 'Do Not Display Embedded Objects'” on page 1088*

Command 'Parallel'

Symbol: 

Function: This command converts the selected alternative branch into a parallel branch.

Call: Main menu *“SFC”*

Requirement: The horizontal connecting line of a branch is selected.

Please note that after you convert a branch, you must check and modify the layout of the steps and transitions before and after the branch.

See also

- *Chapter 1.4.1.20.3.11.11 “Command 'Alternative'” on page 1083*

Command 'Alternative'

Symbol: 

Function: This command converts the selected parallel branch into an alternative branch.

Call: Main menu *“SFC”*

Requirement: The horizontal connecting line of a branch is selected.

Please note that after you convert a branch, you must check and modify the layout of the steps and transitions before and after the branch.

See also

- *Chapter 1.4.1.20.3.11.10 “Command 'Parallel'” on page 1082*

Command 'Insert Branch'

Symbol: 

Function: This command inserts a branch to the left of the selected point.

Call: Main menu *“SFC”*

This command functions similar to the *“Insert Branch Right”* command.

See also

- *Chapter 1.4.1.19.1.4.8.3 “SFC element 'Branch'” on page 491*
- *Chapter 1.4.1.20.3.11.13 “Command 'Insert Branch Right'” on page 1083*

Command 'Insert Branch Right'

Symbol: 

Function: This command inserts a branch to the right of the selected point.

Call: Main menu *“SFC”*

The type of inserted branch depends on the selected element.

- If the uppermost element of the selected elements is a transition or an alternative branch, then CODESYS inserts an alternative branch.
- If the uppermost element of the selected elements is a step, a macro, a jump, or a parallel branch, then CODESYS inserts a parallel branch with the `Branch<x>` jump marker, where `x` is an incremental number. You can edit the default name of the jump marker or define the jump marker as a jump destination.
- If a common element of an existing branch (horizontal line) is selected, then CODESYS inserts the new branch line as a branch line on the far right. If an entire branch line of an existing branch is selected, then CODESYS inserts the new branch line directly to the right as a new branch line.

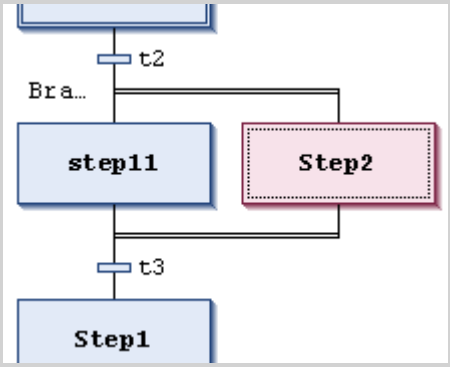
Please note: You can convert a branch into another type with the “Alternative” and “Parallel” commands.

Example of parallel branch

The following image shows a new inserted parallel branch generated by the *"Insert Branch Right"* command while the `Step11` step was selected. CODESYS automatically inserts a step (`Step2` in the example).

Processing in online mode: If `t2` yields TRUE, then CODESYS executes `Step2` immediately after `step11` and before `t3` is passed.

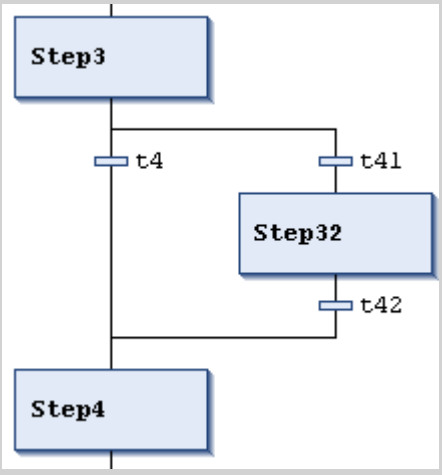
Thus, CODESYS processes both branch lines as opposed to alternative branches.



Example of alternative branch

The following image shows a new inserted alternative branch generated by the *"Insert Branch Right"* command while the `t4` transition was selected. CODESYS automatically inserts a step (`Step32` in the example), a preceding transition, and a subsequent transition (`t41, t42`).

Processing in online mode: If `Step3` is active, then CODESYS passes the subsequent transitions (`t4, t41`) from left to right. The first branch line of the main branch with the first transition yielding `TRUE` is passed. Therefore, only one branch line is processed as opposed to with a parallel branch.



See also

- *Chapter 1.4.1.19.1.4.8.3 "SFC element 'Branch'" on page 491*
- *Chapter 1.4.1.20.3.11.12 "Command 'Insert Branch'" on page 1083*
- *Chapter 1.4.1.20.3.11.10 "Command 'Parallel'" on page 1082*
- *Chapter 1.4.1.20.3.11.11 "Command 'Alternative'" on page 1083*

Command 'Insert Action Association'

Symbol:

Function: This command assigns an IEC action to a step.

Call: Main menu *"SFC"*

Requirement: A step is selected.

CODESYS inserts the action element to the right of the selected step element.

If you have already assigned one or more actions to the step, they are displayed in an action list. The new action is then inserted as follows:

- If you selected the step element, the action is inserted as the first action of the step at first position of the action list.
- If you selected one of the available actions in the action list, then the action is inserted directly above the selected action.

The left section of the action element includes the qualifier (`N` by default). You enter the action name in the right section. To set this value, click in the box to obtain an editing frame. You must have already created this action as a POU in the project.

You can also edit the qualifier. Valid qualifiers are described in the chapter “Qualifiers for Actions in SFC".

See also

- *Chapter 1.4.1.20.3.11.15 “Command 'Insert Action Association After'” on page 1085*
- *Chapter 1.4.1.19.1.4.4 “Qualifiers for Actions in SFC” on page 479*

Command 'Insert Action Association After'

Symbol: 

Function: This command assigns an IEC action to a step.

Call: Main menu *“SFC”*

Requirement: A step is selected.

This command functions similar to the *“Insert Action Association”* command. The difference between the two commands is that CODESYS inserts the new action in the last position of the action list, not the first position. If you select an action in the action list, then CODESYS inserts the new action at the bottom of the list, not at the top.

See also

- *Chapter 1.4.1.20.3.11.14 “Command 'Insert Action Association'” on page 1084*
- *Chapter 1.4.1.19.1.4.4 “Qualifiers for Actions in SFC” on page 479*

Command 'Insert Jump'

Symbol: 

Function: This command inserts a jump element before the selected element.

Call: Main menu *“SFC”*

Requirement: A step is selected.

CODESYS automatically inserts the jump with the `Step` destination. Then, you still have to replace this jump destination with an actual destination by using the input assistant.

See also

- *Chapter 1.4.1.19.1.4.8.4 “SFC element 'Jump'” on page 492*
- *Chapter 1.4.1.20.3.11.17 “Command 'Insert Jump After'” on page 1085*

Command 'Insert Jump After'

Symbol: 

Function: This command inserts a jump element after the selected element.

Call: Main menu *“SFC”*

CODESYS automatically inserts the jump with the `Step` destination. Then, you still have to replace this jump destination with an actual destination by using the input assistant.

See also

- *Chapter 1.4.1.19.1.4.8.4 “SFC element 'Jump'” on page 492*
- *Chapter 1.4.1.20.3.11.16 “Command 'Insert Jump'” on page 1085*

Command 'Insert Macro'

Symbol:

Function: This command inserts a macro element before the selected element.

Call: Main menu *“SFC”*

The new macro is named `Macro<x>` by default, where `x` is an incremental number beginning at `0` for the first macro. You can edit the default name directly by clicking the name.

To edit the macro, click *“Zoom Into Macro”* in the macro editor.

See also

- *Chapter 1.4.1.20.3.11.20 “Command 'Zoom Into Macro'” on page 1086*
- *Chapter 1.4.1.20.3.11.19 “Command 'Insert Macro After'” on page 1086*

Command 'Insert Macro After'

Symbol:

Function: This command inserts a macro element after the selected element.

Call: Main menu *“SFC”*

This command functions similar to the *“Insert Macro”* command.

See also

- *Chapter 1.4.1.20.3.11.20 “Command 'Zoom Into Macro'” on page 1086*
- *Chapter 1.4.1.20.3.11.18 “Command 'Insert Macro'” on page 1086*

Command 'Zoom Into Macro'

Symbol:

Function: This command opens a macro for editing in the macro editor.

Call: Main menu *“SFC”*

Requirement: A macro is selected.

By choosing this command, CODESYS closes the main view of the SFC editor and opens the macro editor. This is also an SFC editor for editing the section of the SFC diagram that is displayed as a macro box in the main view.

Click *“Zoom Out of Macro”* to return to the main view.

See also

- *Chapter 1.4.1.20.3.11.21 “Command 'Zoom Out of Macro'” on page 1086*

Command 'Zoom Out of Macro'

Symbol:

Function: This command closes the macro editor and returns to the main view of the SFC editor.

Call: Main menu *“SFC”*

Requirement: A macro is open in the macro editor.

See also

- *Chapter 1.4.1.20.3.11.20 “Command 'Zoom Into Macro'” on page 1086*

Command 'Paste After'

Symbol: 

Function: This command pastes the elements from the clipboard after the selected position.

Call: Main menu *“SFC”*.

Command 'Change Duplication' - 'Set'

Function: This command embeds every step action or transition, which is called by a step or transition in the SFC box, with the caller. In this way, the action or transition object can be called only from exactly this caller (pseudo-embedding). The result is that copying step and transition elements that call cations or transitions automatically creates new action or transition objects. the implementation code is also copied.

Call: Menu bar: *“SFC”*.

For more details about duplication mode, refer to the help page for the SFC element properties and the instructions for adding step actions.

Pseudo-embedded objects can be hidden in the “Devices” or “POUs” view by means of a command.

See also

- *Chapter 1.4.1.19.1.4.8.6 “SFC element properties” on page 493*
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*

Command 'Change Duplication' - 'Remove'

Function: This command removes the embedding of action, transition, and property objects by a step or transition that calls it for the entire SFC box. In this way, the pseudo-embedding of the action, transition, or property objects is removed. If step or transition elements are copied, which call actions, transitions, or properties, then the copying calls the same actions and transitions as the source.

Call: Menu bar: *“SFC”*.

For more details about duplication mode, refer to the help page for the SFC element properties and the instructions for adding step actions.

Pseudo-embedded objects can be hidden in the “Devices” or “POUs” view by means of a command.

See also

- *Chapter 1.4.1.19.1.4.8.6 “SFC element properties” on page 493*
- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.20.3.11.25 “Command 'Do Not Display Embedded Objects'” on page 1088*

Command 'Do Not Display Embedded Objects'

Function: This command causes action and transition objects, which are embedded in an SFC box by a step or transition, do not appear in the tree.

Call: Context menu of an SFC box in the *“Devices”* or *“POUs”* view.

See also

- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*

Menu 'CFC'

Command 'Edit Worksheet'

Function: This command opens the *“Edit Worksheet”* dialog box in which you set the size of the worksheet.

Call: Main menu *“CFC”*

Requirements: A CFC editor is active.

Dialog box 'Edit worksheet'

“Use following dimensions”	Here is where you set the size of the worksheet. Your change is only accepted if the size is sufficient for the existing program.
“Adapt the dimensions automatically”	Automatically adapts the size of the worksheet to the size of your program.
“Move the working sheet origin relatively”	Shifts the worksheet along the x or y axis. The input of negative numbers is permitted.

Command 'Edit Page Size’

Function: This command opens the *“Edit Page Size”* dialog box, in which you change the size of the page-oriented CFC editor.

Call: Main menu *“CFC”*

Requirements: A page-oriented CFC editor is active.

Dialog box 'Edit page size' dialog box

“Width”	Width of the page (minimum 24, maximum 1024). Elements outside of the working area are marked red.
“Height”	Height of the page (minimum 24, maximum 1024). Elements outside of the working area are marked red.
“Margin width”	Width of the margin (minimum 6, maximum 25% or page width).
“Set as standard for new CFC objects”	☑: The current settings are selected as standard for new CFC objects.

See also

- *Chapter 1.4.1.19.1.6.5.1 “CFC element 'Page'” on page 522*

Command 'Negate'

Symbol: 

Function: This command negates the selected function block input or function block output.

Call: Main menu *“CFC”*, context menu

Requirements: A CFC editor is active. A function block input or function block output is selected.

Command 'EN/ENO'

Symbol: 

Function: This command adds a boolean input *“EN”* (Enable) and a boolean output *“ENO”* (Enable Out) to the selected function block.

Call: Main menu *“CFC”*, context menu

Requirements: A CFC editor is active. A function block is selected.

The added input *“EN”* activates the function block. The function block is executed only if the input is `TRUE`. The value of this signal is output at the *“ENO”* output.

Command 'None'

Symbol: ; keyboard shortcut : *[Ctrl]+[M]* (to toggle between *“S”*, *“R”*, *“REF”*, and None)

Function: The command removes a Reset (R), Set (S), or REF from the input of the *“Output”* element.

Call: Menu bar: *“CFC ➔ Set/Reset”*; context menu: *“Set/Reset”*

Requirement: A CFC editor is active. The input of an *“Output”* element is selected.

Command 'R (Reset)'

Symbol: ; keyboard shortcut : *[Ctrl]+[M]* (to toggle between *“S”*, *“R”*, *“REF”*, and None)

Function: The command adds a Reset to the input of a Boolean *“Output”* element.

Call: Menu bar: *“CFC ➔ Set/Reset”*; context menu: *“Set/Reset”*

Requirement: A CFC editor is active. The input of an *“Output”* element is selected.

If an *“Output”* element has a Reset input, then the Boolean output value is set to *“FALSE”* as soon as the value of the input is *“TRUE”*. The *“FALSE”* value at the output is retained, even if the input value changes.

See also

- *Chapter 1.4.1.20.3.12.7 “Command 'S (Set)'” on page 1091*

Command 'S (Set)'

Symbol: ; keyboard shortcut : *[Ctrl]+[M]* (to toggle between *“S”*, *“R”*, *“REF”*, and none)

Function: The command adds a Set (S) to the input of a Boolean *“Output”* element.

Call: Menu bar: *“CFC ➔ Set/Reset”*; context menu: *“Set/Reset”*

Requirement: A CFC editor is active. The input of an *“Output”* element is selected.

If an *“Output”* element has a Set input, then the Boolean output value is set to *“TRUE”* as soon as the value of the input is *“TRUE”*. The *“TRUE”* value at the output is retained, even if the input value changes.

See also

- *Chapter 1.4.1.20.3.12.6 “Command 'R (Reset)'” on page 1091*

Command 'REF= (Reference Assignment)'

Symbol: ; keyboard shortcut : *[Ctrl]+[M]* (to toggle between *“S”*, *“R”*, *“REF”*, and None)

Function: The command assigns a reference to an *“Output”* element.

Call: Menu bar: *“CFC ➔ Set/Reset”*; context menu: *“Set/Reset”*

Requirements: A CFC editor is active. The input of an *“Output”* element is selected.

Example:

Declaration:

```
ref_int : REFERENCE TO INT;
a : INT;
```

CFC:

This corresponds to the ST code: `ref_int REF= a;`

For more information, see the description for the data type `REFERENCE TO`.

See also

- *Chapter 1.4.1.8.3.2.2 “ Programming in the CFC editor” on page 246*
- *Chapter 1.4.1.19.5.13 “Reference” on page 658*

Command 'Display Execution Order'

Function: The command temporarily shows a numbered tag for all CFC elements of the programming object.

Call

- Menu bar: *“CFC ➔ Execution Order”*
- Context menu in the CFC editor

Requirement: A CFC editor is active and the *“Auto Data Flow Mode”* property is selected.

The numbers represent the automatically determined execution order. The execution order is determined by data flow. In the case of multiple networks, it is determined by their topological position in the editor.

The tags are hidden as soon as you click in the CFC editor.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.3.12.10 “Command 'Set Start of Feedback'” on page 1092*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*

Command 'Set Start of Feedback'

Symbol: 

Function: The command defines the selected element as the starting point within a feedback.

Call:

- Menu bar: *“CFC ➔ Execution Order”*
- Context menu: *“Execution Order”*

Requirement: A CFC editor is active and the *“Auto Data Flow Mode”* property is selected. Moreover, a network of the CFC POU contains a feedback, and an element within the feedback is selected.

In the CFC editor, the starting point within the feedbacks is decorated with the symbol. Then the element has the lowest number in the execution order within the feedbacks. At runtime, the processing of the feedback begins with this element.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.3.12.9 “Command 'Display Execution Order'” on page 1092*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*

Command 'Send to Front'

Symbol: 

Function: The command numbers the elements so that the selected elements are located at the front of the execution order.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and the *“Explicit Execution Order Mode”* property is selected. At least one element is selected.

The selected elements get the lowest numbers beginning at 0 while keeping the previous order. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*
- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Command 'Send to Back'

Symbol: 

Function: The command numbers the elements so that the selected elements are located at the end of the execution order.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and the *“Explicit Execution Order Mode”* property is selected. At least one element is selected.

The selected elements get the highest numbers while keeping the previous order. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Command 'Move Up'

Symbol: 

Function: The command numbers the elements so that the selected elements are located one position forward.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and at least one element is selected. The *“Explicit Execution Order Mode”* property is selected.

The selected elements get a numbering decreased by one while keeping the previous order. The selected elements are processed one position earlier. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Command 'Move Down'

Symbol: 

Function: The command numbers the elements so that the selected elements are located one position backward.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and at least one element is selected. The *“Explicit Execution Order Mode”* property is selected.

The selected elements get a numbering increased by one while keeping the previous order. The elements are processed one position later. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*
- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Command 'Set Execution Order'

Function: The command opens a dialog for setting the number of the selected element to any value.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and the *“Explicit Execution Order Mode”* property is selected. Exactly one element is selected.

The selected element gets the number specified in the dialog. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*
- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Command 'Order by Data Flow'

Function: The command numbers the elements in the program by data flow, or in the case of multiple networks by their topological position in the editor.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and the *“Explicit Execution Order Mode”* property is selected.

The command is also available when no element is selected.

The execution order is determined by data flow. In the case of multiple networks, it is determined by their topological position of the networks. All numbered elements of the POU are set accordingly. Afterwards, the execution order is identical to that in auto data flow mode. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*
- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Command 'Order by Topology'

Function: The command orders the execution order of the elements by their topological position from right to left and from top to bottom.

Call: Menu bar: *“CFC ➔ Execution Order”*; context menu: *“Execution Order”*

Requirements: A CFC editor is active and the *“Explicit Execution Order Mode”* property is selected. At least one element is selected.

The command applies to all elements in the program, even if not all elements are selected when the command is executed. The topological positions of the elements are retained anyway.

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.4.10.13 “Dialog 'Properties' - 'CFC Execution Order'” on page 1165*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*

- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*

Command 'Edit Parameters'

Function: This command opens the *“Edit Parameters”* dialog box, where you change the constant input parameters of a function block.

Call: Main menu *“CFC ➔ Edit Parameters”*, or *“Right-Click ➔ Edit Parameters”*, click the *“Parameter”* function block.

Requirements: A CFC editor is active. An instantiated function block has VAR_INPUT CONSTANT variables in its declaration.

This functionality applies only to blocks that are inserted in a CFC with CODESYS >= V3.5 SP4.

CODESYS displays blocks with VAR_INPUT CONSTANT variables by the word *“Parameter”* in the lower left corner of the block.

Dialog Box 'Edit Parameters'

“Parameters”	Name of the variable
“Type”	Data type of the variables
“Value”	Click into the field to type a value.
“Initial Value”	Initialization Value
“Category”	Additional information about the parameters; these values are defined by attributes and cannot be changed in this dialog box. • `parameterCategory` • `parameterUnit` • `parameterMinValue` • `parameterMaxValue`
“Unit”	
“Min”	
“Max”	
“Delete Prepared Parameters”	This command is active when you write a prepared value (*“Debug ➔ Write Value”*).

When you exit the field and the dialog box by clicking *“OK”*, the value changes are applied to the project.

Example of a block with constant inputs

```
FUNCTION_BLOCK FB1
VAR_INPUT CONSTANT
    {attribute 'parameterCategory':='General'}
    {attribute 'parameterUnit':= 'm/s'}
    {attribute 'parameterMinValue':= '0'}
    {attribute 'parameterMaxValue':= '100'}
    fbin1:INT;
    fbin2:DWORD:=24354333;
    fbin3:STRING:='abc';
END_VAR
```

> *This functionality and the declaration of variables with keyword `VAR_INPUT CONSTANT` applies only to the CFC editor. In the FBD editor, CODESYS always shows all input parameters on the block, regardless of whether or not they are declared as `VAR_INPUT` or `VAR_INPUT CONSTANT`. CODESYS also does not make a distinction about this in text editors.*

See also

- *Chapter 1.4.1.19.1.6.4 “CFC Editor in Online Mode” on page 516*
- *Chapter 1.4.1.20.3.12.19 “Command 'Save Prepared Parameters to Project'” on page 1097*

Command 'Save Prepared Parameters to Project'

Function: This command saves the prepared parameter values to the project.

Call: Main menu *“CFC”*.

Requirements: A CFC editor is active. Parameter values of function block instances are changed in online mode. You are in offline mode.

If the values of constants on the controller are different from the values in the application, then this is indicated by a red asterisk next to the parameter field. Clicking *“Incur Prepared Parameters”* saves the controller values to the application.

See also

- *“Changing of constant input parameters of function block instances” on page 518*
- *Chapter 1.4.1.20.3.12.18 “Command 'Edit Parameters'” on page 1096*

Command 'Connect Selected Pins'

Symbol: 

Function: The command establishes a connection between the selected pins.

Call: Main menu *“CFC”*, context menu

Requirements: A CFC editor is active. Precisely one output and several inputs are selected.

In order to select the pins you must keep the *[CTRL]* key pressed while clicking on the pins. Then you execute the command.

See also

- *Chapter 1.4.1.20.3.12.23 “Command 'Select Connected Pins'” on page 1098*

Command 'Unlock Connection'

Symbol: 

Function: This command unlocks a disabled connection.

Call: Main menu *“CFC ➔ Routing”*, context menu *“Routing”*

Requirements: A CFC editor is active. A connection or a connection mark is selected.

You obtain a disabled connection if you change the connections of the automatic routing. If you wish to carry out automatic routing again, you must first unlock a disabled connection.

With a mouse-click on the icon of a disabled connection you can similarly unlock this connection.

See also

- *Chapter 1.4.1.8.3.2.2 “ Programming in the CFC editor” on page 246*
- *Chapter 1.4.1.19.1.6.5.12 “CFC element 'Connection Mark - Source/Sink'” on page 525*

Command 'Show Next Collision'

Function: This command displays the next collision in the editor and marks the place concerned.

Call: Menu menu *“CFC ➔ Routing”*, context menu *“Routing”*

Requirements: A CFC editor is active and at least one connection with a collision is present.

This function is very useful if you operate with large networks and see only one sub-area. A collision is additionally indicated to you by the red bordered symbol in the top right corner of the editor.

Command 'Select Connected Pins'

Symbol: ; shortcut: *[Ctrl]+[Left Arrow]*, or *[Ctrl]+[Right Arrow]*

Function: The command selects all pins that are connected to the currently selected line, or connected to the currently selected connection mark in page-oriented CFC.

Call: *“CFC”* menu; context menu

Requirements: A CFC editor or a page-oriented CFC editor is active. One line and therefore exactly one connection or exactly one connection mark is selected.

See also

- *Chapter 1.4.1.19.1.6.1 “CFC Editor” on page 511*
- *Chapter 1.4.1.19.1.6.2 “CFC editor, page-oriented” on page 514*
- *Chapter 1.4.1.19.1.6.5.12 “CFC element 'Connection Mark - Source/Sink'” on page 525*

Command 'Reset Pins'

Symbol: , *[Ctrl]+[U]*

Function: The command restores the deleted pins of a box.

Call: *“CFC ➔ Pins”* menu; *“Pins”* in the context menu

Requirements: A CFC editor is active and a box is selected.

The command restores all inputs and outputs of the box as they are defined in their implementation.

See also

- *Chapter 1.4.1.20.3.12.25 “Command 'Remove Unused Pins'” on page 1098*

Command 'Remove Unused Pins'

Symbol:

Function: The command removes all unused pins of the selected element.

Call: Menu *“CFC ➔ Pins”*, context menu *“Pins”*

Requirements: A CFC editor is active. An element is selected.

See also

- *Chapter 1.4.1.20.3.12.24 “Command 'Reset Pins'” on page 1098*

Command 'Add Input Pin'

Symbol: 

Function: The command adds a further input to the selected function block.

Call: Main menu *“CFC ➔ Pins”*, context menu *“Pins”*

Requirements: A CFC editor is active. A function block is selected.

See also

- *Chapter 1.4.1.20.3.12.27 “Command 'Add Output Pin'” on page 1099*

Command 'Add Output Pin'

Symbol: 

Function: The command adds a further output to the selected function block.

Call: Main menu *“CFC ➔ Pins”*, context menu *“Pins”*

Requirements: A CFC editor is active. A suitable function block is selected.

See also

- *Chapter 1.4.1.20.3.12.26 “Command 'Add Input Pin'” on page 1099*

Command 'Route All Connections'

Symbol:

Function: This command cancels all manual changes to the connections in the program and re-establishes the original state.

Call: Main menu *“CFC ➔ Routing”*, context menu *“Routing”*

Requirements: A CFC editor is active.

CODESYS cannot automatically route connections that are fixed by control points. You must remove the control points before executing the command. Use the *“Remove Control Point”* command to do this. Furthermore you must disconnect connections that have been changed manually and are marked by the  icon. Use the *“Disconnect Connection”* command to do this.

See also

- *Chapter 1.4.1.20.3.12.29 “Command 'Remove Control Point'” on page 1099*
- *Chapter 1.4.1.20.3.12.21 “Command 'Unlock Connection'” on page 1097*

Command 'Remove Control Point'

Function: This command removes a control point.

Call: Context menu *“Routing”*

Requirements: A CFC editor is active. You have selected a connecting line.

If you move the mouse pointer over a selected connecting line, the existing control points are displayed with yellow circle symbols. Set the cursor on the control point to be deleted and execute the command from the context menu.

See also

- *Chapter 1.4.1.19.1.6.5.2 “CFC element 'Control Point'” on page 522*
- *Chapter 1.4.1.20.3.12.30 “Command 'Create Control Point'” on page 1100*

Command 'Create Control Point'

Symbol: 

Function: The command creates a control point on a connecting line.

Call: Context menu *“Routing”*

Requirements: A CFC editor is active. The cursor is over a connection.

The control point is created in the position on the connection at which the cursor is located when calling the command. The command corresponds to the *“Control Point”* element in the *“Tools”* window.

See also

- *Chapter 1.4.1.19.1.6.5.2 “CFC element 'Control Point'” on page 522*
- *Chapter 1.4.1.20.3.12.29 “Command 'Remove Control Point'” on page 1099*

Command 'Connection Mark'

Symbol: 

Function: This command switches the display of the connection between two elements back and forth between a connecting line and the use of connection marks.

Call: Main menu *“CFC”*, context menu

Requirements: A CFC editor is active. A connection or a connection mark is selected.

If you have selected a connecting line, the command removes this line and adds a *“Connection Mark - Source”* at the output of one element and a *“Connection Mark - Sink”* at the input of the other. Both are given the same name by default, “C-<n>”, where n is a sequential number.

If you select a pair of connection marks, the command converts these marks into a connecting line.

See also

- *Chapter 1.4.1.19.1.6.5.12 “CFC element 'Connection Mark - Source/Sink'” on page 525*

Command 'Create group'

Symbol: 

Function: This command groups the selected elements.

Call: Main menu *“CFC ➔ Group”*, context menu *“Group”*

Requirements: A CFC editor is active. Several elements are selected.

Grouped elements can only be moved together. The position of the elements is not affected by the grouping.

See also

- *Chapter 1.4.1.20.3.12.33 “Command 'Ungroup'” on page 1101*

Command 'Ungroup'

Symbol: 

Function: The command undoes a previous grouping.

Call: Main menu *“CFC ➔ Group”*, context menu *“Group”*

Requirements: A CFC editor is active. A grouping is selected.

See also

- *Chapter 1.4.1.20.3.12.32 “Command 'Create group'” on page 1100*

Command 'Prepare Box for Forcing'

This command is required when using compiler versions 3.5.11.x and 3.5.12.x.
The command is no longer required for compiler versions >= 3.5.13.0.

Function: The command activates and deactivates the forceability of the inputs for a function block element.

Call:

- CFC
- Context menu

Requirements: The CFC editor is in offline mode and a function block element is selected.

After executing the command, the *“Force Function Block Input”* command is available in online mode to open a dialog for forcing the box input values.

See also

- *Chapter 1.4.1.20.3.12.35 “Command 'Force Function Block Input'” on page 1101*
- *Chapter 1.4.1.19.1.6.1 “CFC Editor” on page 511*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*

Command 'Force Function Block Input'

NOTICE!
This kind of forcing uses a data breakpoint internally and is therefore different from forcing with the *“Force Values”* command or *[F7]*.

Values that were forced by the command *“Force FB Input”* do not respond to the commands *“Show All Forces”* or *“Unforce Values”*.

Function: The command opens the *“Force Value”* dialog to force the selected input of a function block. Forcing can be canceled with the same command and dialog.

Call:

- CFC
- Context menu

Requirements:

- The CFC editor is in online mode and the input of the function block is selected.
- For compiler versions 3.5.11.x and 3.5.12.x, the "forceability" of the function block is enabled by the *“Prepare Box for Forcing”* command.

In the *“Force Value”* dialog, you can either specify a value that the input of the function block should be forced, or remove the currently forced value.

After forcing, the input is highlighted in green again. Boolean inputs get a small monitoring view with the forced value. The forced value is displayed in the *“Value”* column of monitoring views (in the declaration part of the POU or in a watch list).

Dialog 'Force Value'

“Expression”	Name of the function block input. Example: `TON_1.IN`.
“Type”	Data type of the input

Table 160: “What do you want to do?”

“Set a new value to force”	: You can specify a new value in the input field. The format has to correspond to the data type.
“Remove value”	: Forcing at the input is canceled.

See also

- *Chapter 1.4.1.19.1.6.1 “CFC Editor” on page 511*
- *Chapter 1.4.1.11.4 “Forcing and Writing of Variables” on page 401*
- *Chapter 1.4.1.20.3.12.34 “Command 'Prepare Box for Forcing'” on page 1101*

Command 'Use Attributed Member as Input'

Symbol:

Function: This command allows for connecting a structure member to a scalar type input.

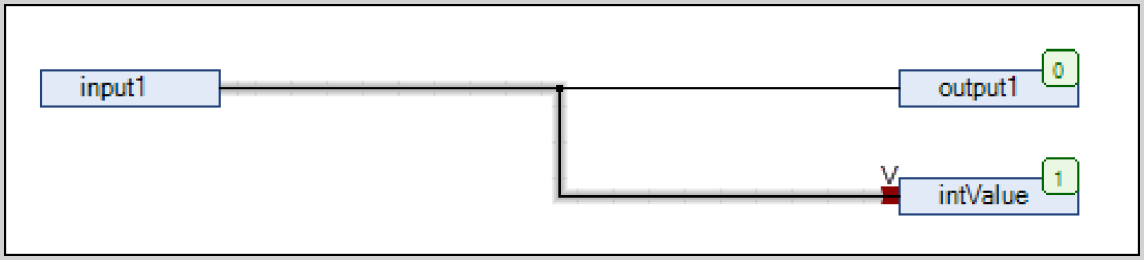
Call: Menu bar: *“CFC ➔ Pins”*; context menu: *“Pins”*

Requirements: A CFC editor is active and a function block input is selected.

The member of the structure that is connected to the input of the subsequent function block must be provided with the pragma `{attribute 'ProcessValue'}`. The data type of the structure member has to be compatible with the data type of the subsequent input. Inputs connected in this way are flagged with the "V" symbol.

Example

```
TYPE QINT :
STRUCT
    Status : STRING;
    {attribute 'ProcessValue'}
    Value1 : INT;
    Value2 : INT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    input1: QINT;
    output1: QINT;
    intValue: INT;
END_VAR
```



If you do not execute the command *“Use attributed member as input”* for this link, then a compiler error is issued.

See also

- *Chapter 1.4.1.19.6.2.37 “Attribute 'ProcessValue'” on page 726*

Menu 'FBD/LD/IL'

Command 'Insert Network'

Symbol:  , shortcut: *[Ctrl]* + *[I]*

Function: This command inserts a further network in the FBD/LD/IL editor.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The FBD, LD or IL editor is active. No box is selected.

See also

- *Chapter 1.4.1.19.1.5.4.1 “FBD/LD/IL element 'Network'” on page 504*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.2 “Command 'Insert Network (Below)'” on page 1105*

Command 'Insert Network (Below)'

Symbol:  , shortcut: *[Ctrl]+ [T]*

Function: This command inserts a further network in the FBD/LD/IL editor below the selected network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A network is selected. No box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Toggle Network Comment State'

Symbol:  , shortcut: *[Ctrl] + [O]*

Function: The command comments the selected network in or out.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A network is selected, but no box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Assignment'

Symbol  , shortcut: *[Ctrl] + [A]*

Function: This command inserts an assignment in the FBD or LD editor.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A network is selected, but no box is selected.

In IL an assignment is programmed via the operators `LD` *and* `ST`*.*

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Box'

Symbol:  , shortcut: *[Ctrl] + [B]*

Function: This command inserts a box that is available in the project at the end of the selected network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A network is selected, but no box is selected.

If you select this command the input assistant opens, where you can select the desired box.

See also

- *Chapter 1.4.1.19.1.5.4.2 “FBD/LD/IL element 'Box'” on page 505*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.8.5 “Using input assistance” on page 260*

Command 'Insert Box with EN/ENO'

Symbol:  , shortcut: *[Ctrl] + [Shift]+ [E]*

Function: This command inserts a box with a boolean input *“Enable”* and a boolean output *“Enable Out”* at the end of the selected network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: the FBD, LD or IL editor is active. A network is selected, but no box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.8.5 “Using input assistance” on page 260*

Command 'Insert Empty Box'

Symbol:  , shortcut: *[Ctrl] + [Shift] + [B]*

Function: This command inserts an empty function block at the end of the currently selected network.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A network is selected, but no box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Empty Box with EN/ENO'

Symbol: 

Function: The command inserts an empty box with a Boolean input *“Enable”* and a Boolean output *“Enable Out”* at the end of the selected network.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD editor, the IL editor or the LD editor is active. A network must be selected. No other box may be selected.

If *“Enable”* has the value `FALSE` at the time of the function block call, then the operations defined in the FB are not executed. Otherwise, if *“Enable”* has the value `TRUE`, these operations are executed. The ENO output acts as a repeater of the EN input.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Box Parallel (Below)'

Function: This command inserts an empty box parallel below the selected function block.

Call: Menu bar: *“FBD/LD/IL”*; context menu.

Requirements: A box is selected in the LD editor.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Jump'

Symbol , shortcut: *[Ctrl]+[L]*

Function: This command inserts a jump element before the selected element.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A connecting line is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.7 “FBD/LD/IL element 'Jump'” on page 506*

Command 'Insert Label'

Symbol:

Function: This command inserts a jump label into the currently selected network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A network is selected. No jump label is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.6 “FBD/LD/IL element 'Label'” on page 506*

Command 'Insert Return'

Symbol: RET

Function: This command inserts an element *“Return”* in the selected place.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD, LD or IL editor is active. A box output is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.8 “FBD/LD/IL element 'Return'” on page 506*

Command 'Insert Input'

Symbol: , shortcut: *[Ctrl]+[Q]*

Function: This command adds a further input to an extendable box (ADD, OR, ADD, MUL, SEL) above the selected input.

Call: *“FBD/LD/IL”* menu

Requirements: The FBD or LD editor is active. An input of a box is selected.

If a box is selected, the command *“Append Input”* is available in the context menu. The input is inserted at the lower end of the box.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Coil'

Symbol: , shortcut: *[Ctrl]* + *[A]*

Function: This command inserts a coil into the network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The LD editor is active. A network, a coil or a connecting line is selected, but no box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.12 “LD element 'Coil'” on page 508*

Command 'Insert Set Coil'

Symbol:

Function: This command inserts a set coil into the network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The LD editor is active. A network, a coil or a line is selected, but no box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Reset Coil'

Symbol:

Function: This command inserts a reset coil into the network.

Call: Main menu *“FBD, LD, IL”*, context menu

Requirements: The LD editor is active. A network, a coil or a line is selected, but no box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.12 “LD element 'Coil'” on page 508*
- *“Ladder diagram (LD)” on page 235*

Command 'Insert Contact'

Symbol , shortcut: *[Ctrl]* + *[K]*

Function: This command inserts a contact to the left of the selected element.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The LD editor is active. A line or a contact is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Command 'Insert Contact (Right)'

Symbol: , shortcut: *[Ctrl] + [D]*

Function: This command inserts a contact to the right of the selected element.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The LD editor is active. A line, a contact or a box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Command 'Insert Contact in Parallel (Below)'

Symbol: ; keyboard shortcut: *[Ctrl]+[R]*

Function: This command inserts a contact with lines in parallel with and below the selected element.

Call: Menu bar: *“FBD/LD/IL”*; context menu.

Requirements: The LD editor is active. A line or a contact or a box is selected.

> *You can program closed parallel branches in a LD network as short circuit evaluation (SCE) or OR constructs. SCE branches are displayed with double vertical lines, and OR branches with single lines. Refer to the help page for "Closed branches".*

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.20.3.13.20 “Command 'Insert Contact in Parallel (Above)'” on page 1109*
- *Chapter 1.4.1.19.1.5.4.14 “Closed branch” on page 509*

Command 'Insert Contact in Parallel (Above)'

Symbol: ; keyboard shortcut: *[Ctrl]+[P]*

Function: This command inserts a contact with lines in parallel with and above the selected element.

Call: Menu bar: *“FBD/LD/IL”*; context menu.

Requirements: The LD editor is active. A line, a contact or a box is selected.

> *You can program closed parallel branches in a LD network as short circuit evaluation (SCE) or OR constructs. SCE branches are displayed with double vertical lines, and OR branches with single lines. Refer to the help page for "Closed branches".*

See also

- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.14 “Closed branch” on page 509*

Command 'Toggle Parallel Mode'

Function: This command toggles a parallel branch between an OR construct and the Short Circuit Evaluation (SCE) .

Call: Menu bar: *“FBD/LD/IL”*; context menu.

Requirements: The LD editor is active. A vertical line of a parallel branch is selected.

> *You can program closed parallel branches in a LD network as short circuit evaluation (SCE) or OR constructs. SCE branches are displayed with double vertical lines, and OR branches with single lines. Refer to the help page for "Closed branches".*

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.14 “Closed branch” on page 509*

Command 'Insert Negated Contact'

Symbol: , shortcut: *[Ctrl]* + *[K]*

Function: This command inserts a negated contact to the left of the selected element.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The LD editor is active. A line or a contact is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Command 'Insert Negated Contact Parallel (Below)'

Symbol:

Function: The command inserts a negated contact with lines in parallel with and below the selected element.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The LD editor is active. A line, a contact or a box is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.8.3.1.2 “Programming ladder diagrams (LD)” on page 239*

Command 'Paste Contacts: Paste Below'

Shortcut: *[Ctrl] + [F]*

Function: This command inserts a previously copied contact with lines below the selected element.

Call: Main menu *“FBD/LD/IL ➔ Paste”*, context menu

Requirements: the LD editor is active.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Command 'Paste Contacts: Paste Above'

Shortcut: *[Ctrl] + [F]*

Function: This command inserts a previously copied contact with lines above the selected element.

Call: Main menu *“FBD/LD/IL ➔ Paste Contacts”*, context menu

Requirements: the LD editor is active. A line or a contact is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Command 'Paste Contacts: Paste Right (After)'

Shortcut: *[Ctrl] + [G]*

Function: this command inserts a previously copied contact to the right of the selected element.

Call: Main menu *“FBD/LD/IL ➔ Paste Contacts”*, context menu

Requirements: The LD editor is active. A line or a contact is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.11 “LD element 'Contact'” on page 507*

Command 'Insert IL Line Below'

Symbol: 

Function: The command inserts an instruction line below the selected line.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The IL editor is active. A line is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Delete IL Line'

Symbol:  , shortcut: *[Ctrl]+[Del]*

Function: This command deletes the selected instruction line.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The IL editor is active. A line is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Negation'

Symbol: , shortcut: *[Ctrl]* + *[N]*

Function: This command negates the following elements:

- Input/output of a box
- Jump
- Return
- Coil

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: the FBD or LD editor is active. The corresponding element is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Edge Detection'

Symbol FBD: , symbol LD: , shortcut: *[Ctrl]* + *[N]*

Function: This command inserts an edge detector before the selected box input or box output.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD or LD editor is active. A box input or box output is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Set/Reset'

Symbol: , shortcut *[Ctrl]* + *[M]*

Function: In the case of an element with a boolean output, this command switches between reset, set and no mark.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD or LD editor is active. An element with a boolean output is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Set Output Connection'

Symbol: , shortcut *[Ctrl]*+ *[W]*

Function: This command turns the selected box output into the forwarding box output.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD or LD editor is active. One of several box outputs is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Branch'

Symbol: , shortcut *[Ctrl]* + *[Shift]* + *[V]*

Function: This command creates an open line branch on the selected line.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD or LD editor is active. An input or an output of a box is selected.

See also

- *Chapter 1.4.1.19.1.5.4.9 “FBD/LD/IL element 'Branch'” on page 506*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Insert Branch Above'

Symbol:

Function: This command inserts a line branch above the selected open line branch.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD or LD editor is active. An open line branch is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.9 “FBD/LD/IL element 'Branch'” on page 506*

Command 'Insert Branch Below'

Symbol:

Function: This command inserts a line branch below the selected open line branch.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The FBD or LD editor is active. An open line branch is selected.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*
- *Chapter 1.4.1.19.1.5.4.9 “FBD/LD/IL element 'Branch'” on page 506*

Command 'Set Branch Start Point'

Symbol

Function: This command sets the starting point of a line branch on the selected line.

Call: Main menu *“FBD/LD/IL”*, context menu

Requirements: The LD editor is active. A line is selected.

See also

- *Chapter 1.4.1.19.1.5.4.14 “Closed branch” on page 509*
- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Set Branch End Point'

Symbol 

Function: This command sets the end point of a line branch on the selected line.

Call: Main menu *"FBD/LD/IL"*, context menu

Requirements: The LD editor is active. A line is selected. A starting point of the line branch has been set.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*
- *Chapter 1.4.1.19.1.5.4.14 "Closed branch" on page 509*

Command 'Update Parameters'

Function: This command enters changes to the declaration of the selected element in the diagram.

Call: Main menu *"FBD/LD/IL"*, context menu

Requirements: The FBD, LD or CFC editor is active. A box is selected. An extending change has been made to the declaration.

The command checks whether a box and its declaration in the declaration editor correspond. The change is accepted for the box only if the declaration was extended. Deletions and overwrites are not updated.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*

Command 'Remove Unused FB Call Parameters'

Symbol: 

Function: This command deletes inputs and outputs of the selected box to which no variable and no value were assigned. However, the default inputs and outputs are always retained.

Call: Main menu *"FBD/LD/IL"*, context menu

Requirements: The FBD or LD editor is active. A box is selected. The box has interfaces to which no value is assigned.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*

Command 'Repair POU'

Symbol: 

Function: This command repairs internal inconsistencies in the selected box.

Call: Main menu *"FBD/LD/IL"*, context menu

Requirements: The FBD or LD editor is active. The defective box is selected. The editor has found internal inconsistencies in the programming module that can possibly be resolved automatically. CODESYS reports the inconsistencies in the Message window.

This situation is conceivable when editing a project that was created with an older programming system version that did not yet handle the inconsistency concerned as an error.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*

Command 'View as Function Block Diagram'

CAUTION!
Loss of data! An error-free conversion requires syntactically correct code. Otherwise parts of the implementation can be lost.

Shortcut: *[Ctrl] + [1]*

Function: This command converts the active instruction list or the active ladder diagram into the function block diagram.

Call: Menu *"FBD/LD/IL ➔ View"*

Requirements: The LD or IL editor is active.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*

Command 'View as Ladder Logic'

CAUTION!
Loss of data! An error-free conversion requires syntactically correct code. Otherwise parts of the implementation can be lost.

Shortcut: *[Ctrl] + [2]*

Function: This command converts the current function block code or the active instruction list into a ladder diagram.

Call: Menu *"FBD/LD/IL ➔ View"*

Requirements: The FBD or IL editor is active.

See also

- *Chapter 1.4.1.19.1.5.1 "FBD/LD/IL Editor" on page 495*

Command 'View as Instruction List'

If necessary, IL can be activated in the CODESYS options.

CAUTION!
Loss of data! An error-free conversion requires syntactically correct code. Otherwise parts of the implementation can be lost.

Shortcut: *[Ctrl] + [3]*

Function: This command converts the active function block code or the active ladder diagram into an instruction list.

Call: Menu *"FBD/LD/IL ➔ View"*

Requirements: The LD or FBD editor is active.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Command 'Go to'

Symbol:

Function: This command allows you to jump to any network.

Call: Main menu *“FBD/LD/IL”*

Requirements: The LD, FBD or IL editor is active. A network is selected.

This command opens a dialog box with an input field. Enter the number of the desired network in the input field.

See also

- *Chapter 1.4.1.19.1.5.1 “FBD/LD/IL Editor” on page 495*

Menu 'Library'

Command 'Add Library'

Function: The command opens the *“Add Library”* dialog. In this dialog, you can add libraries to the Library Manager and then integrate them in your application.

Call: Menu bar: *“Libraries”*

Requirement: The Library Manager is open in the editor.

Dialog 'Add Library'

In the line above the library list, you can search for library names or library modules by typing an appropriate string.	
“Library”	Suitable libraries that are installed in the library repository. For example, the selection of libraries is defined in the device description or by the system integrator. By default, the displayed libraries are grouped into categories.
“Company”	Vendor of the library
“Advanced”	Opens the advanced *“Add Library”* dialog
	The displayed libraries are grouped into categories.
	The displayed libraries are listed in alphabetical order.
	All available libraries are displayed.

Specific libraries can be blacklisted in a device description. These libraries cannot be added below this device in the Library Manager.

See also

- *Chapter 1.4.1.20.3.8.5 “Command 'Library Repository'” on page 1061*

Dialog 'Add Library' – 'Advanced'

You should use this dialog only if you have expert knowledge of library referencing. Using this dialog, you can link special versions or change placeholder definitions.

> *We recommend seriously that you follow the guidelines for the creation of libraries when developing and referencing libraries.*

Table 161: Tab 'Library'

“Company”	Filtering the list according to vendor
“Group by category”	☑: Display of the libraries in a tree structure grouped in categories. ☐: Display of the libraries in alphabetical order in a flat structure.
“Display all versions”	☑: Display of all versions of the libraries. Version specification '*' means the latest version available in the repository. ☐: Display of the latest versions of the libraries only. A multiple selection of libraries is possible in this display. To do this, hold down the *[Shift]* key and select the entries.
“Details”	Opens a detailed view with the library modules.
“Library Repository”	Opens the *“Library Repository”* dialog. There you can install more libraries to your local system.

Table 162: Tab 'Placeholder'

“Placeholder name”	The input field provides a combo box for entering the valid placeholder names that are read from the currently accessible device descriptions. You can also enter a new placeholder name in order to define a free placeholder, which is not resolved by the device or by the library profile.
“Default library”	CODESYS uses this library when for any reason no device is available that the resolution defines. In this way it is possible to compile the current project without errors.

> ***Note about placeholder resolution***
> *For compiler version V3.5.8.0 and later, the following statement applies in the case of library placeholders with a resolution in the device description that are located in the Library Manager of the POU pool. This placeholder is always resolved automatically according to the description of the device that compiles the application.*

See also

- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*
- *Chapter 1.4.1.20.3.14.4 “Command 'Placeholders'” on page 1120*

Command 'Try to Reload Library'

Function: This command tries to reload the selected library.

Call: Main menu *“Library”*.

Requirement: A library is selected that failed to load.

If for any reason a library is not available in the defined repository location when a project is opened, CODESYS displays a corresponding error message. Once you have rectified the error, i.e. when the library is properly available again, you can reload the library with this command without having to leave the project.

See also

- *Chapter 1.4.1.20.3.8.5 “Command 'Library Repository'” on page 1061*

Command 'Properties'

Function: The command opens the *“Properties”* dialog for the library selected in the Library Manager.

Call

- Menu bar: *“Library”*
- Context menu of the selected library
- Symbol in the toolbar of the Library Manager

Requirement: A library is selected.

NOTICE!
This dialog is intended for library developers. Use this only if you have profound knowledge of library referencing. In addition, follow the guidelines for library developers.

See also

- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*

Dialog 'Properties'

Table 163: “General”

“Namespace”	Namespace of the selected library. By default, this is identical to the library name, unless it was defined explicitly in the project information when the library was created. You can change the namespace for the open project. Example: `LA`
“Default”:	Library that triggers the placeholder when no other trigger is defined or is possible. Requirement: The selected library is a library placeholder, and therefore the setting is available. Note: For compiler version 3.5.8.0 and higher, the following statement applies in the case of library placeholders with a resolution in the device description that are located in the Library Manager of the *“POUs”* view. This placeholder is always resolved automatically according to the description of the device that compiles the application.

If the selected library is developed in compliance with the "Guidelines for Developing Libraries", then we do not recommend that you change the following settings.

Table 164: “Version”

Selection of version constraint Requirement: The settings are available only if the selected library is **not** a library placeholder. Note: Container and interface libraries are created automatically with library references with version constraint. As far as possible, do not create libraries that include library references with version constraint. Otherwise, you reference the libraries by placeholders. Edit a placeholder resolution in the *“Placeholders”* dialog.	
“Exact version”	: (selected from list box) Version is integrated into the project. Note: This option is strongly recommended for container libraries, and it is usually preset for this library type.
“Always newest version”	: The library repository is scanned and the latest detected version is integrated. Note: If a newer library version is available, then the library POUs that are actually used can change. This option is strongly recommended for interface libraries, and it is usually preset for this library type.

Table 165: “Visibility”

“Allow only qualified access to all identifiers”	: Library POUs (and variables) are called in the project only with prepended namespace paths.
“When the current project is referenced as a library in another project ”	Note: Changing the following settings makes sense only if you created a library with your project and therefore opened a library project. In this way, the selected library is referenced in the new library.
“Make visible all IEC symbols in the project if is this reference were directly integrated here.”	: As a container library, the selected library makes the contents of the referenced library visible at the top level (later in a project). Requirement: A container project is created with a library project. A container library does not implement its own POUs, but references other libraries exclusively. It bundles libraries. A container library can be employed sensibly to bundle multiple libraries (in a reference) in a project. This option **must** be activated for each library reference. Symbolic access to library POUs: `<namespace of container library>.<POU name>` : The contents of the referenced library is accessed uniquely by means of the namespace. The path name consists of the library name and the unique name (library reference), and it is prepended to the POU name. Requirement: **No** container project is created with a library project.
“Do not show this reference in the dependency tree.”	: The selected library is not displayed in the Library Manager as a library reference (later in a project). The library is a hidden reference. Warning: If there are compile errors resulting from hidden library errors, then detecting the errors may be difficult. : The selected library is displayed as a library reference (later in a project).

“Optional (if the library is missing, no error will be reported).”	: The selected library is treated as optional. When downloading the project that references the library, no error is reported, even if the library is **not** available in the library repository.

See also

- *Chapter 1.4.1.20.3.14.4 “Command 'Placeholders'” on page 1120*

Command 'Placeholders'

Function: This command opens the *"Placeholders"* dialog box. The dialog shows information on the currently selected placeholder library and allows to assign a project-specific resolution.

Call:

- Menu *"Libraries"*
- Symbol in the symbol bar in the upper part of the Library Manager window.

Requirement: A placeholder library is selected in the Library Manager.

A placeholder library, which is included in the *"Devices"* view, will be resolved as follows:

- If you have assigned a specific resolution to the placeholder library via the dialog *"Placeholder"*, this will be applied.
- If no specific resolution is defined, it will be checked, whether there is one specified in the device description of the application..
- Afterwards the library profile will be checked for a resolution definition.
- The result is displayed in the Library Manager below the *"Effective Version"*.

A placeholder library, which is included in the *"POUs"* view, gets resolved as follows:

- A specific resolution defined in the dialog *"Placeholder"* will be **ignored**.
- For the application it will be checked whether there is a resolution defined in the device description.
- Afterwards the library profile will be checked.
- The result is displayed in the tooltip of the symbol .

See also

- *Chapter 1.4.1.16 "Using Libraries" on page 448*
- *Chapter 1.4.1.20.2.14 "Object 'Library Manager'" on page 874*
- *Chapter 1.4.1.16.1 "Information for Library Developers" on page 449*

Dialog box 'Placeholders'

"Name"	Name of the placeholder.
"Library"	Current resolution, valid for the project Double-click on the entry in order to edit the placeholder resolution. A selection list with the available library versions appears. Additionally the command *"Other Library"* is available.
Command *"Other Library"*	The command opens the dialog box *"Bibliothek durchsuchen"* for searching and installing libraries. Choose this command, if you do not want to redirect to another version, but on a specific libray.
"Info"	Type of placeholder resolution: ● Resolved by device description ● Resolved by library profile ● Resolved by <specific library>

Command 'Export Library'

Function: This command is used for saving the library file to the hard disk.

Call: Context menu of the Library Manager

Requirement: A library is selected in the Library Manager.

The command opens the standard dialog for saving a file in the local file system. The library file can have the file type `Library files (*.library)`, `Compiled library files (*.compiled-library)`, or `Compiled library files (*.compiled-library-v3)`.

See also

- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*
- *Chapter 1.4.1.16.4 “Exporting library files” on page 451*

Menu 'Image Pool'

Command 'Insert Image'

Symbol: 

Function: This command inserts a new line into an image pool.

Call: Main menu *“Imagepool”*, or right-click.

Requirements: An image pool is active and a line is selected in the image pool.

See also

- *Chapter 1.4.1.20.2.13 “Object 'Image Pool'” on page 873*

Menu 'Declarations'

Command 'Insert'

Symbol 

Function: This command inserts a new line for a variable declaration in the declaration editor and the input field for the variable name opens.

Call: Context menu in the tabular declaration editor; button in the declaration heading.

To edit the other fields of the declaration lines, double-click the fields and select the data from the drop-down lists or by means of the respective dialogs.

See also

- *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226*

Command 'Edit Declaration Header'

Function: The command opens the dialog *“Edit Declaration Header”*, which serves in the declaration editor for the configuration of a POU header.

Call: Context menu of the tabular declaration editor

Requirements: The tabular declaration editor is the active editor.

See also

- *“Declaring in the tabular declaration editor” on page 227*

Dialog 'Edit Declaration Header'

Function: The dialog is for configuring the declaration part of a POU.

Call: Click on the header bar of the tabular declaration editor, or context menu in the tabular declaration editor.

"Declaration"	Selection list for changing the POU type ● *"PROGRAM"* ● *"FUNCTION_BLOCK"* – *"EXTENDS"*: Input field for a basic function block – *"IMPLEMENTS"*: Input field for an interface ● *"FUNCTION"* – *"Return type"* Input field with current POU name: you can change the name of the POU
"Automatically adapt all references on rename"	☑: Dialog box *"Refactoring"* opens. ☐: Renaming is only effective in the declaration header of the POU.
"Attributes"	The dialog box *"Attribute "* opens for the input of attributes and pragmas.

See also

- *Chapter 1.4.1.8.2.1 "Using the declaration editor" on page 226*
- *Chapter 1.4.1.19.6 "Pragmas" on page 683*
- *Chapter 1.4.1.8.15 "Refactoring" on page 289*

Command 'Move Down'

Symbol: 

Function: This command shifts a variable declaration downwards by one row.

Call: Context menu

Requirement: A row with a variable declaration is selected in the tabular declaration editor.

See also

- *Chapter 1.4.1.8.2.1 "Using the declaration editor" on page 226*

Command 'Move Up'

Symbol: 

Function: This command shifts a variable declaration upwards by one row.

Call: Context menu

Requirement: A row with a variable declaration is selected in the tabular declaration editor.

See also

- *Chapter 1.4.1.8.2.1 "Using the declaration editor" on page 226*

Menu 'Declarations' (Persistence)

Command 'Reorder List and Clean Gaps'

Function: This command cleans the gaps that can result when you make changes to the declaration of persistent variables. The memory requirement is reduced by this cleaning. When the command is executed, CODESYS displays a warning informing the user about the possible loss of data.

Call: Main menu *“Declarations”*, context menu

Requirement: The persistence editor (persistent variable list) is active.

Before cleaning you should consider saving the current values of the persistent variables to a recipe (command *“Save Current Values to Recipe”*). Then you can load the values to the controller again after the next download.

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.20.3.17.3 “Command 'Restore Values from Recipe'” on page 1123*
- *Chapter 1.4.1.20.3.17.2 “Command 'Save Current Values to Recipe'” on page 1123*

Command 'Save Current Values to Recipe'

Function: This command creates a new recipe definition in the recipe manager and stores the current values of the persistent variables in it. You should execute this command before the command *“Reorder List and Clear Gaps”* in order to avoid a possible loss of data. You can subsequently restore the data with the command *“Restore Values from Recipe”*.

Call: Main menu *“Deklarationen”*

Requirement: The application is in online mode and the persistence editor (persistent variable list) is active.

If a list already exists in the recipe manager with the corresponding names when saving a persistent variable list, then the current persistent variables are sorted into the list:

- *New persistent variables are added to the list*
- *Variables, that are not in the list, will be deleted*

Therefore, it is possible to add more recipes to the list in the recipe manager and these will be retained . However, if new variables are added to the list, then these are deleted the next time the command “Save Current Values to Recipe” is executed.

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.20.3.17.1 “Command 'Reorder List and Clean Gaps'” on page 1123*
- *Chapter 1.4.1.20.3.17.3 “Command 'Restore Values from Recipe'” on page 1123*
- *Chapter 1.4.1.20.2.12 “Object 'Persistent variable list'” on page 872*

Command 'Restore Values from Recipe'

Function: This command restores the values of the persistent variables that you have stored in a recipe using the command *“Save Current Values to Recipe”*. You would normally select this command after executing the command *“Reorder List and Clear Gaps”*.

Call: Main menu *“Declarations”*

Requirement: The persistence editor (persistent variable list) is active, the application is in online mode

See also

- *Chapter 1.4.1.8.19 “Data Persistence” on page 301*
- *Chapter 1.4.1.20.3.17.2 “Command 'Save Current Values to Recipe'” on page 1123*
- *Chapter 1.4.1.20.3.17.1 “Command 'Reorder List and Clean Gaps'” on page 1123*
- *Chapter 1.4.1.20.2.12 “Object 'Persistent variable list'” on page 872*
- *Chapter 1.4.1.20.3.17.2 “Command 'Save Current Values to Recipe'” on page 1123*

Command 'Add all instance paths'

Function:

- When you execute the command in the persistence editor, the application is searched for declarations of persistent variables with the `PERSISTENT` keyword which are outside of the persistence editor. For each declaration found, an instance path of this variable is added in the persistence editor.
- When you execute the command in a variable configuration, an instance path is added for each variable with an incomplete address. All function blocks of the application are considered in this case.

Call: Menu bar: *“Declarations”*, right-click.

Requirement

- the persistence editor (global persistent variable list) is active or a variable configuration (global variable list with `VAR_CONFIG` declarations) is opened.
- The application was compiled successfully.

See also

- *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*
- *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537*

Menu 'Device Communication', Gateway

Command 'Add New Gateway'

Function: This command opens the *“Gateway”* dialog where you can define a gateway channel and add it to the current device configuration.

Call: Menu bar: *“Gateway”* in the *“Communication Settings”* dialog of the device editor.

Dialog 'Gateway'

“Name”	Name of the gateway.
“Driver”	Driver type from a drop-down list.
Driver-specific settings, for example: IP address, port	Editable after double-clicking the predefined value. A short description for each parameter is displayed in the lower part of the dialog. Note: You can also specify the address of a DNS domain. This has to begin with `dns:` (example: `dns:MyDynDNSAdress`).

The dialog is also used for later editing of the gateway entries of your project.

See also

- *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*

Command 'Configure the Local Gateway'

Function: The command opens the *“Gateway Configuration”* dialog where you can configure the block drivers for the local gateway. This is an alternative to manually editing the configuration file `Gateway.cfg`.

Call: Context menu when a gateway entry is selected in the device editor in the *“Communication Settings”* dialog.

> **NOTICE!**
> A correct configuration of the gateway requires detailed knowledge. In case you have any doubts, do not change the default configuration settings.

Dialog 'Gateway Configuration'

The configuration tree displayed in the dialog corresponds to the description currently valid configuration file `gateway.cfg`. It displays the parameters with the current settings for the interfaces involved. Changes to the configuration in the dialog, confirmed by clicking *“OK”* result in the direct update of the configuration file.

> *After the gateway configuration file `gateway.cfg` has been changed, the gateway has to be restarted in order for the changes to be applied.*

“Add”	Menu with commands for adding interfaces and settings. The commands are also available in the context menu of the dialog. The selection depends on which entry is selected and which settings have already been added: *“Add Interface”*: Select an interface for communication via the gateway. It is inserted at the top level of the tree. See the table below for the possible block driver interfaces. *“Add Configuration Setting”*: Select a setting for the selected interface. It is inserted below the interface in the tree. To edit the value of the setting, double-click in the *“Setting”* column to open an editing field. See the table below for the possible settings per block driver interface.
“Delete”	Deletes the selected configuration setting
“Up”, *“Down”*	Moves the selected configuration entry one position up or down.

Table 166: Possible block driver interfaces

"COM Port"	Serial port on the device, for example for data exchange according to the RS-232 standard on a COM port intended for this purpose.
	Possible configuration settings: *"Name"*: Symbolic only *"Port"*: Physical serial port which is used for this interface, for example `COM 5` on a Windows computer *"Baudrate"*: `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, `115200` *"Activate auto addressing"*: ☑ (default = ☐) The setting `Local address` is evaluated. Both devices, which communicate via the serial port, will negotiate their addresses independently before they begin exchanging messages. If the addresses of both devices are the same, then they are negotiated again. This setting is useful when the local addresses cannot be set explicitly, for example for physically separated devices. *"Local address"*: Evaluated only when *"Enable auto addressing"* is activated. Default = actual value for port
"Shared Memory":	Shared memory driver
	Possible settings: *"Name"*: Symbolic only *"Forced address"*: Default = -1 (= `no forced address`); example: `42` means that the driver has to use the fixed address defined here and that addresses are assigned freely in the range `0`-`255`. This setting can be useful when more than one shared memory driver is activated in the configuration.
"Ethernet UDP/IP":	Ethernet interface for data exchange according to the "user datagram protocol".
	Possible settings: *"Name"*: Symbolic only *"Port index"*: Port number for the communication. Port indices are in the range `0`–`3`. They are mapped to the following Ethernet port: `1740` to `1743`. *"IP address"*: Default = `127.0.0.1`. This setting can be useful to explicit set an interface when the device has several network interfaces. Example: `127.0.0.1` stands for some local network interface, also known as `localhost`. Every other address (example: `10.27.7.72`) represents a real IP address which has to be available on the device. *"Network mask"*: Default = 255.255.255.0; example: 255.255.252.0. This setting can be useful to explicitly set an interface when there are multiple network interfaces on the device. *"PPP remote address"*: Default = `127.0.0.1`; example: `10.13.42.240`; establishes a logical point-to-point connection between the UDP interface and the node named with the address specified here; has the effect that the UDP interface communicates exclusively with this node and that no broadcasts are sent in the network
"Ethernet TCP/IP":	Ethernet interface for data exchange according to the "Transmission Control Protocol".

	Possible settings: *“Name”*, *“Port”*, *“IP address”*: See Ethernet UDP/IP above. *“Inactivity timeout”*: Default = 0. This setting defines the time span (in seconds) after which the TCP connections are closed when data is no longer exchanged.
“CAN Client”	*“Name”*: Symbolic only A description for the other settings can be found directly in the dialog.
“USB Port”	*“Name”*: Symbolic only A description for the other settings can be found directly in the dialog.

See also:

- *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*

Menu 'Recipes'

Command 'Insert Variable'

Symbol: 

Function: This command inserts a variable into the currently opened recipe definition before the selected position.

Call: Main menu *“Recipes”*.

Requirement: You have opened a recipe definition in the editor and selected the normal view.

CODESYS inserts the default text "NewVariable" in the column *“Variable”*. You must replace this name with the respective variable name. To do this, open the input assistant by clicking  or enter the variable name directly into the table element.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Add a New Recipe'

Symbol: 

Function: This command opens a dialog box for adding a new recipe (new column) to the recipe definition.

Call: Main menu *“Recipes”*.

Requirement: You have opened a recipe definition in the editor.

After choosing the command, a dialog box opens for you to define the name of the new recipe. The dialog box also provides the capability of copying existing recipes into the new recipe.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Remove Recipe'

Symbol: 

Function: This command removes a recipe from the currently opened recipe definition.

Call: Main menu *“Recipes”*.

Requirement: You have selected a field in the recipe column of a recipe definition.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command ‘Load Recipe'

Symbol: 

Function: The command loads a recipe from a file.

Call: Menu bar: *“Recipes”*.

Requirement: You have selected a field in the recipe column of a recipe definition.

This command overwrites the values of the selected recipe of the recipe definition.

If you have selected the option “Recipe Management in the PLC”, please note the following.

If you change recipes in the project by choosing the command “Load Recipe” or “Read Recipe”, then an online change is required when logging in again.

If you want to overwrite only individual recipe variables with new values, then remove the values for the other variables before loading to the recipe file. Entries without value definitions are not read, and therefore updating leaves these variables unchanged on the PLC and in the project.

For values of the data type REAL/LREAL, the hexadecimal value is also written to the recipe file in some cases. This is necessary so that the exact identical value is restored when converting back. In this case, change the decimal value and delete the hexadecimal value.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Save Recipe'

Symbol: 

Function: This command saves the variable values of a recipe to a file.

Call: Main menu *“Recipes”*.

Requirement: You have selected the value of a recipe in the recipe definition.

When you choose this command, CODESYS saves the values of the selected recipe to a file. You can define the format in the settings for the recipe manager in the tab *“Storage”*.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Read Recipe'

Symbol: 

Function: This command reads the variable values of a recipe from the controller.

Call: Main menu *“Recipes”*.

Requirement: The application is in online mode and you have selected the value of a recipe in the recipe definition.

When you choose this command, CODESYS overwrites the values of the selected recipe with the read values from the controller.

If you have selected the option “Recipe Management in the PLC”, please note the following.

If you change recipes in the project by choosing the command “Load Recipe” or “Read Recipe”, then an online change is required when logging in again.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3 “Menu Commands” on page 955*

Command 'Write Recipe'

Symbol: 

Function: This command writes the values of a recipe to the variables in the controller.

Call: Main menu *“Recipes”*.

Requirement: The application is in online mode and you have selected the value of a recipe in the recipe definition.

When you choose this command, CODESYS overwrites the values in the controller with the values of the selected recipe.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Load and Write Recipe'

Symbol: 

Function: This command loads a recipe from a file and writes the values to the variables in the PLC.

Call: Menu bar: *“Recipes”*.

Requirement: The application is in online mode. You have selected the value of a recipe in the recipe definition.

After choosing the command, you are prompted either to write the variable values also to the recipe in the project or only to write them to the PLC. Updating the values in the recipe could require an online change when logging in again.

When you choose this command, CODESYS overwrites the values of the selected recipe of the recipe definition. In addition, these recipe values overwrite the variable values in the PLC.

If you want to overwrite only individual recipe variables with new values, then remove the values for the other variables before loading to the recipe file. Entries without value definitions are not read, and therefore updating leaves these variables unchanged on the PLC and in the project.

For values of the data type REAL/LREAL, the hexadecimal value is also written to the recipe file in some cases. This is necessary so that the exact identical value is restored when converting back. In this case, change the decimal value and delete the hexadecimal value.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Read and Save Recipe'

Symbol: 

Function: This command reads the variable values of a recipe from the controller and saves them to a file.

Call: Main menu *“Recipes”*.

Requirement: The application is in online mode and you have selected the value of a recipe in the recipe definition.

After choosing the command, you are prompted either to read the variable values to the recipe or only to save them. Updating the values in the recipe could require an online change when logging in again.

The values are saved with the default name for recipe files according to the settings for the recipe manager (tab *“Storage”*).

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Remove Variables'

Symbol 

Function: This command removes the selected variables from a *“Recipe Definition”*.

Call: The command is not in any menu by default. You can add it to a menu by using the dialog box from *“Tools ➔ Customize”* (command category *“Recipe”*).

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Command 'Load Recipes from Device'

Symbol: 

Function: This command initiates the synchronization of the recipes from the open recipe definition in the project and the recipes located on the device in the form of recipe files.

Call: Menu bar: *“Recipes”*.

Requirement: The application is in online mode and a recipe definition is open in the editor.

In detail, the synchronization is described as follows:

- The current values for the recipe variables located in the project are overwritten by the values from the recipes on the controller. As a result, there is likely an online change at the next login.
- If recipe variables are defined in the recipe files on the controller, and the recipe variables are missing in the recipe definition of the project, then these variables are ignored when at the time of download. Before that, a message appears for each recipe file regarding the variables in question.
- If recipe variables are missing in the recipe files on the controller, and these recipe variables are included in the recipe definition of the project, then a message appears for each recipe file with the variables in question.
- If more recipes for these variables have been created on the controller, then they are added to the recipe definition in the project.

Command 'Update Structured Variables'

Symbol: 

Function: This command opens the *“Update Structured Variables”* dialog box.

Call: Main menu *“Recipes”*.

In this dialog box, you can update recipe definitions if the declaration of a structured variable or a block has changed. For example, if the dimension of an array is changed, then you can automatically add or remove the entries in the recipe definition.

Table 167: Dialog Box 'Update Structured Variables'

“Remove not existing variables”	☑: Variables are removed from the recipe definition when they no longer exist in the project due to a change to a structured element.
“Update instances of structures and function blocks”	☑: If the declaration of a structure or function block is extended and available in the recipe definition with an instance, then the respective variables are added to the recipe definition.
“Update array dimensions of array instances”	☑: If the dimension of an array is extended and available in the recipe definition with an instance, then the respective variables are added to the recipe definition.
“Update contained global variable lists”	☑: If the declaration of a global variable list is extended and available in the recipe definition with an instance, then the respective variables are added to the recipe definition.
“Update contained programs”	☑: If the declaration of a program is extended and instanced in the recipe definition, then the respective variables are added to the recipe definition.

See also

- *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- *Chapter 1.4.1.20.3.19 “Menu 'Recipes'” on page 1127*

Menu 'Text List'

Command 'Add Language'

Symbol: 

Function: This command adds a further language column to the text list.

Call: Main menu *"Textlist"*, context menu

Requirement: A text list or a global text list is open and active.

In the dialog box *"Enter Language"*, enter a code for the new language, for example *"en-US"*. CODESYS inserts the code as column header.

Command 'Create Global Text List'

Symbol: 

Function: This command creates the global text list in the *"POUs"* view.

Call: *"Visualization"*, context menu.

Requirements: A visualization is open.

See also

- *Chapter 1.4.1.20.2.9 "Object 'GlobalTextList'" on page 871*
- *Chapter 1.4.1.8.8.1 "Managing static text in global text lists" on page 269*

Command 'Export Everything as Text'

Symbol: 

Function: This command exports all the text lists of the project.

Call: Main menu *"Textlist"*, context menu

Requirement

- A text list or a global text list is open and active.
- The visualization does not code the characters of the texts in Unicode.

CODESYS creates a file as plain text in the format `.txt` for each text list. The name of the text list becomes the name of the file. The directory into which the files are exported is set in *"Project ➔ Project Settings ➔ Visualization"*, category *"General"* in *"Text list files"*.

A controller can read and use this format. For example, you can copy the file to a controller and, by means of a configuration in the visualization manager, prevent the text lists from being transmitted again when loading the application.

Command 'Export All Unicode .txt Text List Files'

Symbol: 

Function: This command exports all the text lists of the project.

Call: Main menu *"Textlist"*, context menu

Requirement

- A text list or a global text list is open and active.
- The visualization codes the characters of the texts in Unicode.
  - The option *"Use Unicode strings"* in the visualization manager is activated.
  - The compiler instruction `VISU_USEWSTRING` in the application is set. Check this by selecting the command *"Properties"* in the context menu of the application. Then select the *"Compile"* tab. `VISU_USEWSTRING` must be entered in the input field for *"Compiler defines"*.

CODESYS creates a file as plain text in the format `.txt` for each text list. The name of the text list becomes the name of the file. The directory into which the files are exported is set in *"Project ➔ Project Settings ➔ Visualization"*, category *"General"* in *"Text list files"*.

A controller can read and use this format. For example, you can copy the file to a controller and, by means of a configuration in the visualization manager, prevent the text lists from being transmitted again when loading the application.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*

Command 'Insert Text'

Symbol: 

Function: This command inserts a new line into the text list above the selected line. An input field under *"Standard"* opens, in which you input the source text.

Call: Main menu *"Textlist ➔ Insert Text"*, context menu

Requirement: A text list, not a *"GlobalTextList"*, is open and active. A field in the table is selected.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*

Command 'Import/Export Text Lists'

Symbol: 

Function: This command exports an active text list, imports a file, or matches a text list with a file. The file has the CSV format. The *"Import/Export"* dialog provides options for this.

Call: Menu bar: *"Text List ➔ Import/Export Text Lists"*; context menu

Requirement: A text list or global text list is active.

Dialog 'Import/ Export'

"Select File for Import"	File that CODESYS reads. ... opens the dialog *"Select Text List File"* for you to select a file.
"Select export file"	File that CODESYS writes to. ... opens the dialog *"Select Text List File"* for you to select a file and directory.

Table 168: “Import/Export Type”

“Import”	Requirement: A file is selected in *“Select file for compare or import”*. The file can contain text list entries for both the global text list and text lists. Global text list ● CODESYS reads the file, compares the text list entries for the same source text, and accepts differences in the translations. CODESYS overwrites any translations in the project. Text lists ● CODESYS reads the file, compares the text list entries for the same ID, and accepts differences in the source text and translations into the project. CODESYS overwrites any text list entries in the project. ● If the file contains a new ID, then the text list entry is imported into the text list of the project and the text list is added.
“Import replacement file”	Requirement: A replacement file is selected in *“Select file for compare or import”*. The replacement file contains replacements for the global text list. CODESYS processes the replacement file row by row and performs the specified replacements in the global text list. The structure of the replacement file is described in the section "Managing static text in a global text list".
“Export”	Requirement: The file that CODESYS writes to is selected in *“Select export file”*. CODESYS exports all texts from all text lists of the current project. All languages available in the project are inserted as columns in the export file. The file can be used for the external translation of the language-dependent texts.
“Export text differences only”	Requirement: An import file is selected for the comparison in *“Select file for compare or export”*, and an export file that CODESYS writes to is selected in *“Select export file”*. CODESYS reads the import file and then uses that information to compare the rows of the active text list. CODESYS ignores the rows that match. If rows differ, then CODESYS writes the row to the export file and, if necessary, copies translations from the text list. CODESYS accepts the translations from the import file and overwrites them if necessary.

See also

- *Chapter 1.4.1.8.8 “Managing text in text lists” on page 266*
- *“Updating the global text list with a replacement file” on page 271*

Command 'Remove Language'

Symbol: 

Function: Removes the selected language column from the text list.

Call: Main menu *“Textlist”*, context menu

Requirement: A text list or a global text list is open and active. A field is selected in the column of the language that you wish to remove.

See also

- *Chapter 1.4.1.8.8 “Managing text in text lists” on page 266*

Command 'Rename Language'

Symbol: 

Function: Opens a dialog for specifying a new name for a language that is displayed in the text list as a column heading.

Call: Menu bar: *"Text List"*; context menu.

Requirement: A text list or global text list is active. A field in the language column to be renamed is selected.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*

Command 'Remove Unused Text List Entries'

Symbol: 

Function: This command checks whether a text list entry in the project is used as static text. If not, CODESYS removes it from the text list.

Call: Main menu *"Textlist"*, context menu

Requirement: The *"GlobalTextList"* is open and active. A field in the table is selected.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*

Command 'Check Visualization Text IDs'

Symbol: 

Function: This command checks whether the ID of a text list entry in the project is correct and reports the result.

Call: Main menu *"Textlist"*, context menu

Requirement: The *"GlobalTextList"* is open and active. A field in the table is selected.

If CODESYS finds during checking that the global text list and the static texts of the visualizations do not correspond, this could be because the global text list is or was write protected. The requirement for this is that you have set up a user management system in the project.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*

Command 'Update Visualization Text IDs'

Symbol 

Function: This command updates all inconsistent IDs in a static text list.

Call: Main menu *"Textlist ➔ Paste Text"*, context menu

Requirement: The *"GlobalTextList"* is open and active. A field in the table is selected. The object is write protected.

If CODESYS finds during checking that the global text list and the static texts of the visualizations do not correspond, this could be because the global text list is or was write protected. The requirement for this is that you have set up a user management system in the project.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*
- *Chapter 1.4.1.5.5 "Protecting Objects in the Project by Access Rights" on page 204*

Command 'Add Text List Support'

Symbol: 

Function: This command adds text list support to the selected DUT object (type: enumeration).

Call: Context menu of a standard DUT object (type: enumeration ).

Text list support allows the localization of the enumeration component identifier and the display of the symbolic component value in a text output of a visualization.

See also

- *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835*
- *Chapter 1.4.1.20.3.20.13 “Command 'Remove Text List Support'” on page 1136*

Command 'Remove Text List Support'

Symbol: 

Function: This command removes text list support from the selected enumeration object.

Call: Context menu of an object of an enumeration with text list support ().

Text list support allows the localization of the enumeration component identifier and the display of the symbolic component value in a text output of a visualization.

47293

See also

- *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835*
- *Chapter 1.4.1.20.3.20.12 “Command 'Add Text List Support'” on page 1136*

Menu 'Trace'

Command 'Add Variable'

Function: This command adds a trace variable to the configuration.

Call: Main menu *“Trace”*.

A new variable is displayed in the tree view of the trace configuration. The associated variables configuration appears to the right in *“Variable Settings”*.

See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Chapter 1.4.1.12.3.2 “Creating trace configuration” on page 424*

Command 'AutoFit'

Symbol:

Function: This command scales the y-axis of the trace diagram for optimum display of all graphs, making sure that the y-values fit in the visible region of the diagrams. The command works with both single-channel and multi-channel displays.

Call: Menu bar: *“Trace”*; context menu.

Trace in a single-channel display When all trace variables are displayed in one diagram, the trace is in single-channel display.

Trace in a multi-channel display When the trace variables are displayed in multiple diagrams, the trace is in multi-channel display.

See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Command 'Compress'

Symbol: 

Function: This command compresses the trace graph by zooming into the displayed time range by a fixed percentage.

Call: Main menu *“Trace”*, or context menu.

See also

- *Chapter 1.4.1.20.3.21.18 “Command 'Stretch'” on page 1146*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Command 'Configuration'

Function: This command opens the *“Trace Configuration”* dialog box for enabling the configuration of the data recording.

Call: Main menu *“Trace”*, or context menu.

See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *“Subdialog 'Variable Settings'” on page 1211*

Command 'Cursor'

Symbol: 

Function: This function

- inserts a trace cursor into the trace diagram when **no** trace cursor is available
- inserts a second trace cursor into the trace diagram when **1** trace cursor is available
- removes the trace cursors when **2** trace cursors are available

Call: Menu bar: *"Trace"*; context menu.

A trace cursor is a small black triangle with a vertical black line running parallel to the y-axis.

Trace diagram without trace cursors

In this mode, you can process the trace diagram with the mouse pointer. The x-value that focuses on with the cursor is displayed in the status bar with normal style. Example: *"Time: 1m23s456ms; Value: 1 "*

Trace diagram with one trace cursor

In the status bar and y-value, CODESYS prints the time that was marked by the trace cursor. Example: *"Time: 1m23s456ms "*

Trace diagram with two trace cursors

In the status bar, CODESYS prints the two times and the time interval that are marked by the two trace cursors. Example: *"Time: 1m23s456ms - Time: 1m24s456ms (Δ 1s) "*.

User input in the trace diagram

If one or two trace cursors are available, then you can move them along the x-axis.

Mouse Input	Symbol	Effect
Drag the triangle of a trace cursor to another position.		While the mouse button is pressed, the cursor can be moved without restriction. The current y-value is always displayed in the status bar. When the mouse button is released, the cursor jumps to the nearest measuring point

Keyboard Shortcuts	Effect
[Left arrow] *[Right arrow]*	CODESYS moves the black trace cursor to the next measuring point.
[Shift]+[Left Arrow] *[Shift]+[Right Arrow]*	CODESYS moves the gray trace cursor to the next measuring point.

See also

- *Chapter 1.4.1.12.3.5 "Navigating into trace data" on page 429*

Command 'Download Trace'

Symbol: 

Function: This command transfers the trace configuration on the controller to the associated application, and starts the data recording. The recorded data is transferred back to the development system. The trace diagram shows the current samples and continues.

Call: Menu bar: *"Trace"*; context menu.

Requirement: The command is available when the assigned application is in online mode.

See also

- *Chapter 1.4.1.12.3.3 "Operating the data recording" on page 427*

Command 'Export Symbolic Trace Config'

Function: This command exports a trace configuration to a `traceconfig` file.

Call: Main menu *“Trace”*, or context menu.

Requirement: The origin application includes a symbol configuration that defines the configured trace variables as symbols. Access to the IEC variables where data was recorded is therefore symbolic. Then you can use the trace configuration for various similar applications.

Using the configuration file

You can transfer this file to any runtime system. At runtime, its `CmpTraceMgr` runtime system component can access and perform data recording. The configuration file also includes information about the application context in addition to the configuration data.

The configuration file defines the following context:

- Application name
- Trace name
- Task name

The application that is executed at runtime must fulfill the following conditions:

- The application has the same name as the origin application.
- The trace that is configured in the application has the same as the trace that is configured in the origin application
- The task that is running in the data recording has the same name as the task that is configured in the origin application.

NOTICE!
The configuration is not loaded automatically. You must execute the command explicitly.

You can proceed as follows:

– Access the trace manager programmatically via IEC code by using library interfaces.
– Register the configuration file with the trace manager. Then the trace manager loads the configuration file when the application is started.

For more information about the functionality of the trace manager, refer to "Trace Manager Runtime System Component Description".

Sample configuration file

Configuration file `Trace_Trigger.traceconfig`

```
[key]; [value]
Version; 0x03050000
Name; Application.Trace_Trigger
ApplicationName; Application
ApplicationDataGuid; 00000000-0000-0000-0000-000000000000
IecTaskName; MainTask
Comment;
Trigger.Flags; 5
Trigger.Edge; 2
Trigger.Position; 0
Trigger.UpdatesAfterTrigger; 50
Trigger.Variable.Name; PLC_PRG.B.OUT
Trigger.Variable.AddrFlags; 0x00000101
Trigger.Variable.Class; 0
Trigger.Variable.Size; 1
Trigger.Level;
Condition.Name;
Condition.AddrFlags; 0x00000000
Condition.Class; 0
Condition.Size; 0
EveryNCycles; 1
BufferEntries; 100
Flags; 16
0.Variable; PLC_PRG.S5.OUT
0.Address.AddrFlags; 0x00000101
0.Class; 7
0.Size; 2
0.GraphColor; 4278190335
0.GraphType; 3
0.MinWarningColor; 4278190080
0.MaxWarningColor; 4294901760
0.CriticalLowerLimit; 0
0.CriticalUpperLimit; 0
0.ActivateMinWarning; 0
0.ActivateMaxWarning; 0
0.YAxis; 0
0.Data;
1.Variable; PLC_PRG.B.OUT
1.Address.AddrFlags; 0x00000101
1.Class; 0
1.Size; 1
1.GraphColor; 4278222848
1.GraphType; 1
1.MinWarningColor; 4278190080
1.MaxWarningColor; 4294901760
1.CriticalLowerLimit; 0
1.CriticalUpperLimit; 0
1.ActivateMinWarning; 0
1.ActivateMaxWarning; 0
1.YAxis; 0
1.Data;
```

See also

- *Chapter 1.4.1.12.3.6 “Managing trace” on page 430*

Command 'Load Trace'

Function: This command makes it possible to load a file, which contains the configuration and data, and was saved to the file system of the development system. The *“Load Trace”* dialog box opens.

Call: Main menu *“Trace”*, or context menu.

Dialog box *“Load Trace”*

“File name”	Name of the file that is loaded
“File type”	File format ● `*.trace`: *“Trace file”* that includes the trace configuration ● `*.csv`: Text file in CSV format that includes a trace configuration

See also

- *Chapter 1.4.1.20.3.21.15 “Command 'Save Trace'” on page 1145*
- *Chapter 1.4.1.12.3.6 “Managing trace” on page 430*

Command 'Mouse Zooming'

Symbol: (command disabled), (command enabled)

Function: This command enables and disables mouse zooming in the trace diagram.

Call: Menu bar: *“Trace”*; context menu.

User input in the trace diagram

If the command is enabled, then you can stretch a box with the mouse. When you release the mouse button, the display zooms in on the box and the data is enlarged.

See also

- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

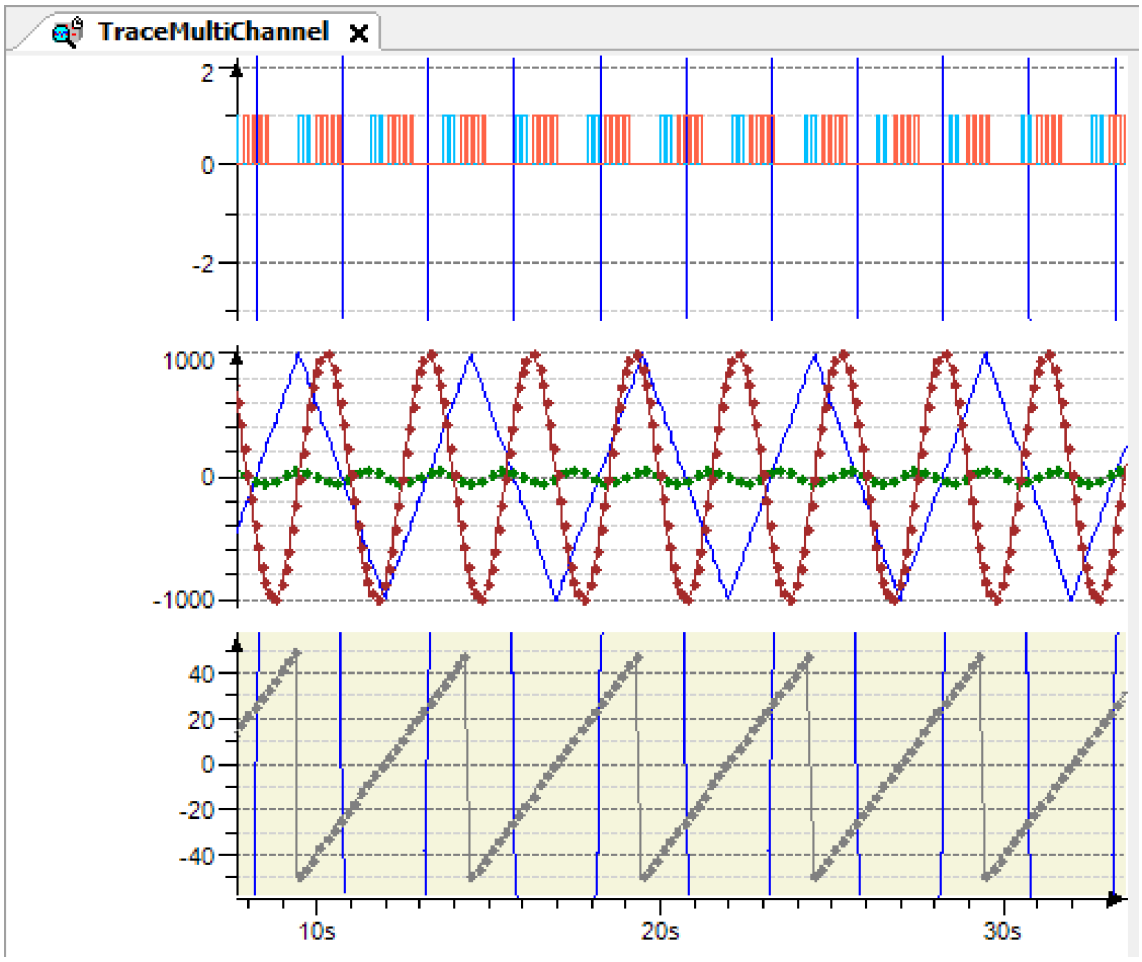
Command 'Convert to Multi-Channel'

Function: This command switches the display in the trace editor from single-channel to multi-channel.

Call: Menu bar: *“Trace”*; context menu.

Multi-channel display

Multi-channel display means that the trace variables are displayed in multiple diagrams.



See also

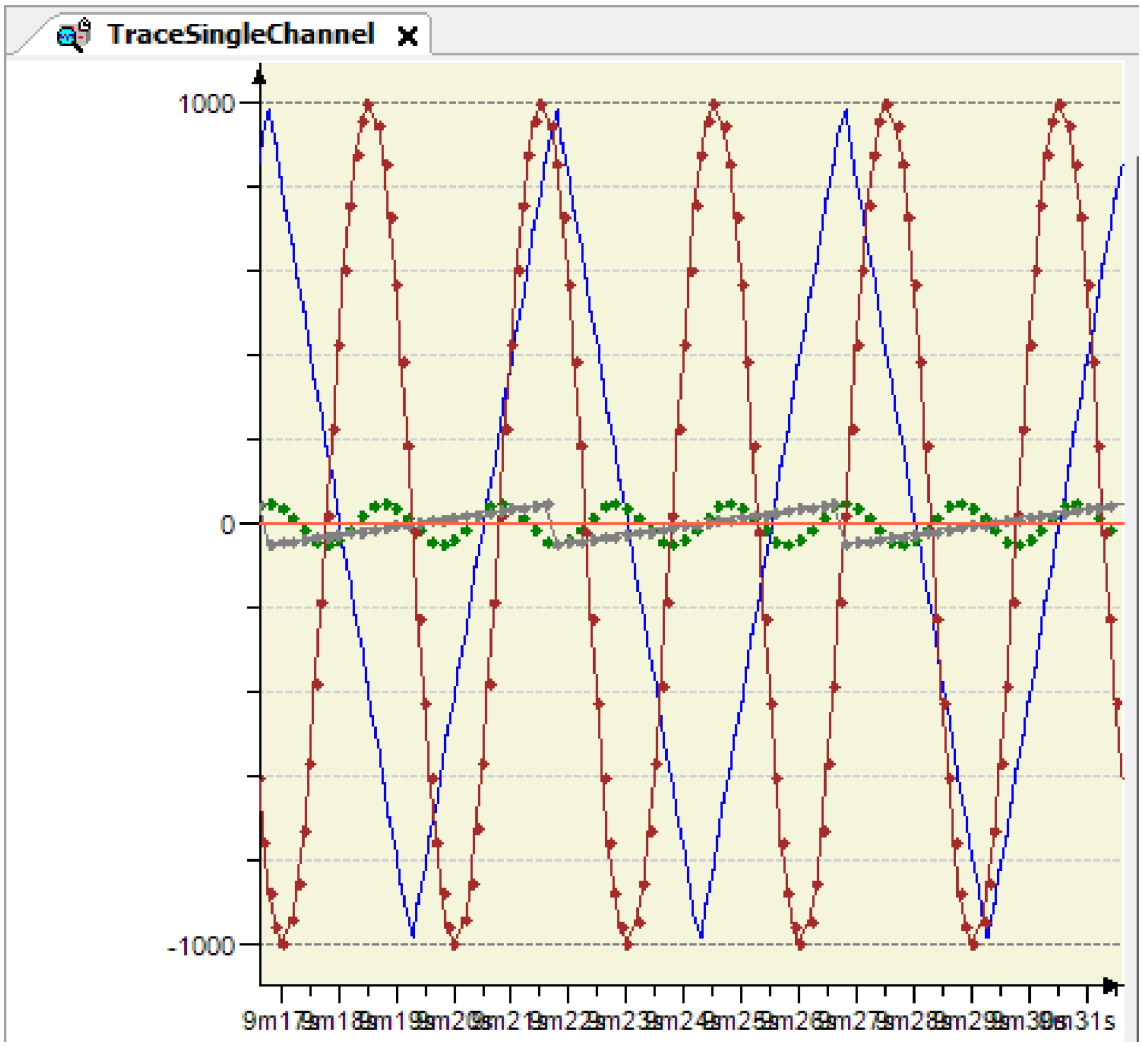
- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Command 'Convert to Single-Channel'

Function: This command switches the display in the trace editor from multi-channel to single-channel.

Call: Menu bar: *“Trace”*; context menu.

Single-channel display If a trace is displayed as single-channel, then all trace variables are included in one diagram.



See also

- *Chapter 1.4.1.20.4.15.2 “Dialog 'Trace Configuration'” on page 1209*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Command 'Online List'

Function: This command opens the *“Online List”* dialog. If the trace editor of a DeviceTrace object is active, then all traces that are running on the controller are displayed in a tree view. If the trace editor of an application-specific trace object is active, then only this trace is displayed if it is running on the controller.

Call: Menu bar: *“Trace”*; context menu of the trace editor.

Requirement: The runtime system uses the `CmpTraceMgr` components. An application belonging to the device is in online mode.

NOTICE!

Closing the DeviceTrace editor terminates the connection to the controller.

Please note that the connections to the controller is also terminated when the last open *"DeviceTrace"* editor is closed. In order for device traces to be displayed again in the project, you have to reload them into the *"DeviceTrace"* objects.

At this time, closing the editor is also the recommended procedure for deliberately terminating the connection to the controller. Logging out is not enough for this.

Dialog 'Online List'

"Delete from runtime"	Stops and removes the selected trace from the running application.
"Upload"	This command is visible when a DeviceTrace is loaded in the trace editor. A DeviceTrace is a trace that runs on the controller: In the device tree, it can be represented with a DeviceTrace object directly below a device. When you execute this command, the trace that is selected in the tree view is loaded from the runtime system into the trace editor. Any existing configuration in the project is overwritten. For example, the device can provide traces for data of the processor load (`cpuload`, `plcload`), which then you can track in the trace editor in CODESYS. An individual *"DeviceTrace"* object is necessary in the device tree for each trace of the device that should be displayed in the project.

See also

- *Chapter 1.4.1.12.3.4 "Accessing All Traces on the Controller" on page 428*
- *Chapter 1.4.1.20.2.29 "Object 'DeviceTrace'" on page 948*
- *Chapter 1.4.1.20.2.28 "Object 'Trace'" on page 945*
- *Chapter 1.4.1.20.3.21.19 " Command 'Upload Trace'" on page 1146*

Command 'Reset Trigger'

Symbol: 

Function: This command resets the trace configuration after a triggered data recording. Then the application can record new data and react to a trigger again.

Call: Main menu *"Trace"*, or context menu.

Requirement: After triggering, the complete data is in the buffer of the development system.

See also

- *Chapter 1.4.1.12.3.3 "Operating the data recording" on page 427*

Command 'Reset View'

Symbol: 

Function: This command resets the trace diagram to the default view.

Call: Main menu *"Trace"*, or context menu.

Requirement: The display in the trace diagram has been changed by zooming, scrolling, or *“AutoFit”*.

See also

- *Chapter 1.4.1.20.3.21.2 “Command 'AutoFit'” on page 1137*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Command 'Save Trace'

Function: This command saves the data to a file on the development system. Depending on the file format, the configuration may also be saved. The *“Save Trace”* dialog box opens.

Call: Main menu *“Trace”*, or right-click.

Dialog Box 'Save Trace'

“File name”	Name and location of the trace file
“File type”	File format ● `*.trace`: *“Trace file”* contains the data and configuration. You can run the *“Load Trace”* command to load the file to the trace editor when offline. ● `*.txt`: *“Text file”* contains the recorded data. You can load this file type and edit it with tools that support CSV format. It cannot be loaded to the trace editor when offline because the trace editor cannot read this format. ● `*.trace.csv` *“Trace CSV file (data only)”* contains the recorded data. Address information is provided for each trace variable. The created file can be read in the runtime system. The data is imported but the trace cannot be started because the variable addresses are not saved.

See also

- *Chapter 1.4.1.20.3.21.8 “Command 'Load Trace'” on page 1141*
- *Chapter 1.4.1.12.3.6 “Managing trace” on page 430*

Command 'Start Trace'

Symbol: 

Function: This command starts the data recording on the controller when it is stopped.

Call: Main menu *“Trace”*, or context menu.

Requirement: The assigned application on the runtime system is running and a trace configuration is loaded.

See also

- *Chapter 1.4.1.12.3.3 “Operating the data recording” on page 427*

Command 'Stop Trace'

Symbol: 

Function: This command stops the data recording of a trace.

Call: Main menu *“Trace”*, or context menu.

Requirement: The assigned application on the runtime system is running and executes a trace.

See also

- *Chapter 1.4.1.12.3.3 “Operating the data recording” on page 427*

Command 'Stretch'

Symbol: 

Function: This command stretches the trace graph by zooming out of the displayed time range by a fixed percentage.

Call: Main menu *“Trace”*, or context menu.

See also

- *Chapter 1.4.1.20.3.21.3 “Command 'Compress'” on page 1137*
- *Chapter 1.4.1.12.3.5 “Navigating into trace data” on page 429*

Command 'Upload Trace'

Function: This command establishes the connection to the PLC device, if not already connected. Then it opens the *“Online List”* dialog listing the traces running on the controller. Then the selected trace is loaded to the trace editor by means of the *“Upload”* command in the dialog.

Call: Menu bar: *“Trace”*; context menu of the trace editor.

Requirement: The editor of a *“DeviceTrace”* object is open. The runtime system uses the `CmpTraceMgr` components (trace manager). At least one application in the runtime system in running. The communication settings for the PLC are configured correctly in the CODESYS project.

> **NOTICE!**
>
> **Closing the DeviceTrace editor terminates the connection to the controller.**
>
> Please note that the connections to the controller is also terminated when the last open *“DeviceTrace”* editor is closed. In order for device traces to be displayed again in the project, you have to reload them into the *“DeviceTrace”* objects.
>
> At this time, closing the editor is also the recommended procedure for deliberately terminating the connection to the controller. Logging out is not enough for this.

See also

- *“Dialog 'Online List'” on page 1144*
- *Chapter 1.4.1.20.2.29 “Object 'DeviceTrace'” on page 948*
- *Chapter 1.4.1.12.3.4 “Accessing All Traces on the Controller” on page 428*

Command 'Statistics'

Function: This command opens the *“Trace Statistics”* dialog box, which shows statistics about each trace variable.

Call: Main menu *“Trace”*, or right-click.

Requirement: The trace editor contains samples.

Dialog Box 'Trace Statistics'

The analyzed time range and duration are shown in the caption.

The table contains one line per signal.

“Signal”	Name pf the trace variable (for example, `PLC_PRG.S1`)
“Index”	0-based index of the signal order
“N”	Number of measurements for the calculations
“Min”	Smallest value
“Avg”	Average
“Median”	Middle value when the values are ordered by size
“RMS”	Root mean square
“StdDev”	Standard deviation
“Max”	Largest value
“Integral”	Integral
“Min ⊿t [s]”	Smallest change of time intervals for successive values
“Avg ⊿t [s]”	Average change of time intervals for successive values
“Median ⊿t [s]”	Median change of time intervals for successive values
“StdDev ⊿t [s]”	Standard deviation of change of time intervals for successive values
“Max ⊿t [s]”	Largest change of time intervals for successive values

Click a column head in the table.	CODESYS sorts the table by that column, changing the order from ascending to descending and back. Default: The table is sorted ascending by the *“Index”* column. The signals are then sorted in the same order as in the signal tree.
Click in the line.	The line is selected. You can select or clear other lines by pressing *[Shift]+ [arrow]* up or down.
[Ctrl]+[C]	CODESYS copies the selected lines as text to the clipboard. The values of the individual columns are tab-separated, and the lines are delimited with the control character *[CR]* or *[LF]*. Requirement: At least one line is selected.

See also

- ↳ *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*

Other

Command 'Add Watch'

Symbol: 

Function: This command adds the variable of the current location of the cursor to a watchlist for the purpose of online monitoring.

Call: Right-click a variable in an editor when the application is in online mode.

This command adds the variable to the currently opened watchlist. If a watchlist is not open, then the variable is added to the *“Watch 1”* list and that view opens.

See also

- *Chapter 1.4.1.12.1.2 “Using watch lists” on page 416*
- *Chapter 1.4.1.12.1 “Monitoring of Values” on page 409*

Command 'Implement Interfaces'

Function: This command updates the implemented interfaces for a function block.

Call: Context menu of the selected function block (FB) in the device tree.

Requirement: The function block implements an interface that you have modified. For example, an additional method was added to the interface.

In object-oriented programming, if you derive a function block (FB) from a base function block, which implements one or more interfaces, for the purpose of inheritance, then the following applies:

When you execute the “Implement Interfaces” command for the derived FB, all interface methods and interface attributes of the base FB are accepted into the derived FB in the form of stubs (without implementation). Then you are responsible for making sure that an "empty" method/attribute in the derived FB does not conflict with an implemented one in the base FB. The following actions are taken to support you in this case: If there es a base implementation for a method/attribute, then CODESYS adds a pragma attribute {error..} in the first line of the affected derived interface method or interface attribute that will generate the error message. If there is no base implementation for the method/attribute, then there is a pragma attribute entry for a warning. After editing the block, you must remove the error pragma attribute entry explicitly.

See also

- *Chapter 1.4.1.20.2.18.4 “Object 'Interface'” on page 888*
- *Chapter 1.4.1.8.22.2 “Implementing interfaces” on page 312*

Command 'Limit Results to Current Declaration'

Function: When multiple declarations have been found, this command collapses the display in the cross-reference list. It shows only the results for the declaration that you selected explicitly in the list.

Call: Right-click.

Requirement: The cross-reference list is active. Multiple declarations for the searched symbol are listed as cross-references.

See also

- *Chapter 1.4.1.8.13.1 “Using the cross-reference list to find occurrences” on page 285*

1.4.1.20.4 Dialogs

The dialogs of the CODESYS user interface bascally are described on the help pages for the CODESYS menu commands or CODESYS objects. The help book *“Dialogs”* contains only descriptions of dialogs, which

- appear only after multi-step calls after a certain menu command call or within an object editor,
- or which are not placed on a help page for a command or for an object because of their complexity (multiple subdialogs).

Dialog 'Import Assistant'

Function: The dialog allows for the transfer of CODESYS options and package installations from an older CODESYS installation that was found in the local computer.

Call: The dialog opens when a recently installed CODESYS version is started for the first time and an older version is installed on the computer.

“Program settings”	☑: The user-specific CODESYS options are transferred from the older installation to the new installation.
“Packages”	☑: The packages installed with the older CODESYS version are transferred to the Package Manager of the new version. See the list of discovered package installations with the *“Name”*, *“Version”*, and *“Installation date”*.
“Import”	The program settings and/or options are transferred to the current CODESYS version.
“Skip”	The program settings and/or options are not transferred to the current CODESYS version.

See also

- *Chapter 1.4.1.1.1 “Setting CODESYS options” on page 180*

Dialog 'Library Reference Conversion'

Function: The dialog defines how references to libraries that are no longer available are to be handled. Note: The undefined library references can be found in the Global Library Manager located in the *“POUs”* view.

Call: When opening a CoDeSys V2.3 project in V3, the dialog opens when the converter detects a library which cannot be used anymore in the current CODESYS version.

> *A CoDeSys V2.3 project can be converted into a CODESYS V3 project only if the CODESYS V2.3 Converter package is installed in CODESYS V3. The package is available in the CODESYS Store.*

Table 169: “What do you want to do?”

“Convert and install the library as well.”	The converter also converts the library file into the new format. It remains referenced in the project. It is installed automatically in the library repository in the "Other" category. If the library does not provide the necessary project information for an installation, then the *“Enter Project Information”* dialog opens for the information to be added.
“Use the following library that has already been installed”	The previously used library is replaced by another library. The *“Browse”* button opens a dialog for selecting from the local library repository.
“Ignore the library. The reference will not appear in the converted project”	The library reference is removed from the project.

“Remember this mapping for all future occurrences of that library reference”	The settings made here in the dialog are also used for future project conversions.

See also

- *Chapter 1.4.1.2.2 “Opening a V2.3 project” on page 187*
- *Chapter 1.4.1.20.3.8.5 “Command 'Library Repository'” on page 1061*

Dialog 'Select Function Block'

Function: The dialog is used for selecting a function block for I/O mapping. The function block should be mapped to the I/O channel selected in the *“<device name> I/O Mapping”* tab or to the object selected in the *“<device name> IEC Objects”* tab.

Call:

- Tab *“<device name> I/O Mapping”*, command button *“Add FB for I/O channel”*
- Tab *“<device name> IEC Objects”*, command button *“Add”*

The dialog provides all function blocks from the active application and the libraries included in the project which fulfill the following:

- The function block has the `{attribute 'io_function_block'}` attribute.
- The function block contains input or output parameters that match the channel type (input, output, data type) and has the `{attribute 'io_function_block_mapping'}` attribute.

When a function block is selected that provides multiple matching parameters, only the first one is mapped automatically to the channel. The others can only be assigned manually in the *“<device name> I/O Mapping”* tab.

After the function block is assigned, the parameter of the function block instance is entered in the *“Variable”* column of the mapping table. Then the path is composed as follows:

`<application name>.<device channel name>_<FB name>_<continuous FB instance number>. <FB parameter name>`

Example: `App1.Out_4_Int_myScale_Output_Int_1.iOutput` for the parameter `iOutput` of the first inserted instance of the function block `myScale_Output`.

“Find”	Input field for searching for function block names
“Type”	Function blocks in the tree structure that match the channel type. Nodes: application, library name(s)
“Documentation”	Shows the available documentation for the library selected in the tree or the library block.

See also

- *Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*
- *Chapter 1.4.1.19.6.2.22 “Attribute 'io_function_block', 'io_function_block_mapping'” on page 707*

Dialog 'Device Conversion'

Function: The dialog defines how references to devices that are no longer available are to be handled.

Call: When opening a CoDeSys V2.3 project in V3, the dialog opens when the converter detects a device reference which cannot be used anymore.

A CoDeSys V2.3 project can be converted into a CODESYS V3 project only if the CODESYS V2.3 Converter package is installed in CODESYS V3. The package is available in the CODESYS Store.

Table 170: “What do you want to do?”

“Use the following device that has already been installed”	CODESYS replaces the previously used device in the device tree with another device. The *“Browse”* button opens a dialog for selecting from the local device repository.
“Ignore the device. All device specific objects will not be available in the new project”	The device entry with all objects inserted below it is removed from the device tree.

“Remember this mapping for all future occurrences of that device”	The settings made here in the dialog are saved in the CODESYS Options, in the *“CODESYS V2.3 Converter”* category. As a result, they are also valid for future project conversions.

See also

- *Chapter 1.4.1.2.2 “Opening a V2.3 project” on page 187*
- *Chapter 1.4.1.20.3.8.8 “Command 'Device Repository'” on page 1067*

Dialog 'Breakpoint Properties'

Function: The dialog is used to display or change the properties of the selected breakpoint in the *“Breakpoints”* view.

Call:

- *"Breakpoints"* view, *"Properties"* button
- *"Breakpoints"* view, *"New"* button, *"New Breakpoint"* command or *"New Data Breakpoint"* command

Requirement: An entry is selected in the list of breakpoints.

The dialog is identical to the *"New Breakpoint"* dialog which is opened in the *"Debug"* menu by means of the respective commands. Therefore, see the description in the help for the *"New Breakpoint"* dialog.

See also

- *Chapter 1.4.1.20.4.8 "Dialog 'New Breakpoint'" on page 1154*

Dialog 'Permissions'

Function: The permissions of user groups are defined here with which they can execute specific actions on specific objects in the project.

Call: Menu bar: *"Project ➔ User Management"*.

Every change made in the dialog is applied immediately.

Actions

All possible actions on objects of the projects are listed in *"Actions"*. The actions are divided into four categories and assignments to all current objects of the project are listed below each action. For each "action->object" assignment, you can define the permission for each existing user group.

Action categories:

"Commands"	Actions regarding the execution of commands
"Users, groups and permissions"	Actions regarding the configuration of user accounts, user groups, and their permissions
"Object types"	Actions regarding the creation of object types
"Project objects"	Actions regarding the viewing, modification, removal, and child-object handling of objects of the project

Actions in detail:

"Execute"	Execute a menu command
"Create"	Create a new object in the project
"Add or remove children"	Add or remove a child object below an existing object
"Modify"	Modify an object in the editor or modification of user, group, and permission settings in the corresponding editor/dialog
"Remove"	Delete or remove an object
"View"	Open the view of an object in the editor

	Possible target of an action. This can be specific objects of the project, or the user, group, and permission configuration.

Permissions

All defined user groups (except the "Owner" group) are listed in *"Permissions"* with a toolbar for configuring the permissions of a group.

Granted	The action, which is selected in the actions view, on the selected target(s) is **granted** for the selected group.
Denied	The action, which is selected in the *"Actions"* view, on the selected target(s) is **denied** for the selected group.
	The permission that executes the actions, which are selected in the *"Actions"* view, on the selected targets has not been defined explicitly. However, the actions are **granted by default**; for example, because the corresponding permissions have been granted to the parent object. Example: The group has the permission for the object "myplc". As a result, it also has the permission by default for the object "myplc.pb_1".
	The action, which is expanded in the actions view, on the selected targets has not been denied explicitly. However, it is denied by default; for example, because it has been denied to the parent object.
No symbol	There are currently multiple actions selected in the Actions view for which the group does not have the same permission.

Toolbar:

"Grant"	The selected action on the selected target object is granted explicitly for the selected group.
"Deny"	The selected action on the selected target object is denied explicitly for the selected group.
"Clear"	The permission for the selected action on the selected target object is reset to the default value for the selected group.

"Export/Import"	Opens a menu with the commands ● *"Export all permissions"* ● *"Export selected permissions"* ● *"Import permissions"*
"Export all permissions"	Exports all actions and their configured access permissions of the current project to a user-specific file of data type `*.perms`. To do this, the *"Export Permissions"* dialog opens for you to specify a file name and to select a location in the file directory. The default file type is `Permissions (*.perms)`.
"Export selected permissions"	Exports all selected actions and their configured access permissions of the current project to a user-specific file of data type `*.perms`. To do this, the *"Export Permissions"* dialog opens for you to specify a file name and to select a location in the file directory. The default file type is `Permissions (*.perms)`.
"Import permissions"	The contents of a `*.perms` file is merged with the actions and permissions of the current project. Groups that are part of the file but not part of the project are ignored. The actions and permissions are aligned by name. To do this, the *"Import Permissions"* opens for you to select the `*.perms` file from the file system.

See also

- *Chapter 1.4.1.5.5 "Protecting Objects in the Project by Access Rights" on page 204*

Dialog Box 'Prepare Value'

Function: This dialog box is used for preparing a value for a forced variable. CODESYS executes the prepared action with the next forcing.

CODESYS opens the dialog box in the following situations:

- When clicking in the field *“Prepared value”* of a forced variable in the declaration section
- When clicking in the inline monitoring field of a forced variable
- When clicking in the field *“Prepared value”* of a forced variable in the monitoring window

“Prepare a new value for the next write or force operation”	Value that CODESYS writes to the variable with the next force operation
“Remove a preparation with a value”	CODESYS deletes the prepared value.
“Release the force, without modifying the value”	CODESYS retains the forced value and ends forcing. CODESYS marks the variable `<Unforce>`.
“Release the force and restore the variable to the value it had before forcing it”	CODESYS resets the forced value and ends forcing. The variable is marked with `<Unforce and restore>`.

See also

- *Chapter 1.4.1.20.3.7.16 “Command 'Force Values'” on page 1053*

Dialog 'New Breakpoint'

Function: In the dialog, you define the settings for a new breakpoint or data breakpoint. It is identical to the *“Breakpoint Properties”* dialog which is used in the *“Breakpoints”* view.

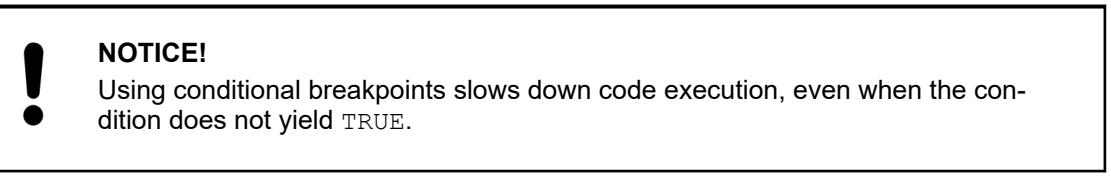
Call:

- *“Debug ➔ New Breakpoint”*
- *“Debug ➔ New Data Breakpoint”*

Requirement: The application is in online mode.

Tab 'Condition'

The dialog defines the requirements for which program processing should halt at a breakpoint.



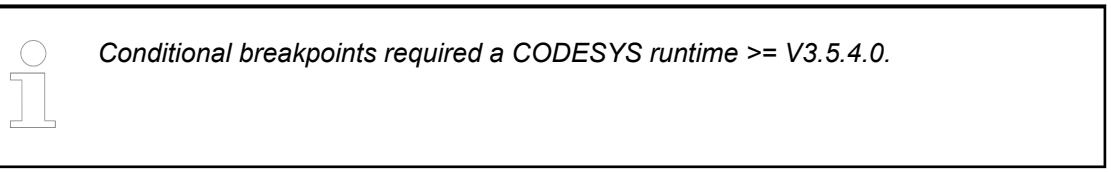


Table 171: “Tasks”

“Break only when the breakpoint is hit in one of the following tasks”	☑: CODESYS evaluates the breakpoint only if it is reached by specific tasks. The required tasks must be activated. For example, you can define a single debug task and also prevent other tasks that use the same block from being affected when debugging.

Table 172: “Hit Count”

“Hit Count”	*“Break always”*: The program always halts at this breakpoint. Alternative: The program halts at the breakpoint when the breakpoint has been hit as often as defined in the following (type in the required hit count or select it from the number list): ● *“Break when the hit count is equal to”* ● *“Break when the hit count is a multiple of ”* ● *“Break when the hit count is greater than or equal to”*

Table 173: “Condition”

“Break, when true”	☑: CODESYS evaluates the specified condition and halts the program at the breakpoint only when the result yields `TRUE`. You can define a condition as a valid Boolean expression. Examples: `x>100`, `x[y]=z`, `a AND b`, `boolVar`.

Tab 'Data' Requirement: This is used for the properties of a data breakpoint.

The function of data breakpoints depends on the target system. Currently, data breakpoints are possible only with the CODESYS Control Win V3.

On the tab, the variable or memory address is specified for which the data breakpoint is set or will be set.	
“Break execution when the value of the variable or address changes”	● Input of a qualified variable name ● …: Selection of a variable in the *“Input Assistant”* dialog, in the *“Watch Variables”* category Examples: variable: `PLC_PRG.fb_DoSth.dwVariable`; address: `16#12A`, `0x12A`, `129`
“Size”	Number of bytes of the specified variable or memory address above which should be monitored for changes. When a new variable or memory address is specified, a value that matches the data type or memory is set automatically at first. Note: The *“Size”* and quantity depend on the target system. For the CODESYS Control Win V3, a maximum of four data breakpoints with a maximum size of 8 bytes can be defined. Example: `4` for data type `DWORD` Example: `2` for data type `DWORD`: Only the two first bytes of the variable are monitored.

Tab 'Execution Point Settings' Here. an existing breakpoint or data breakpoint can be converted into an execution point.

"Execution point (execution does not stop at breakpoint)"	☑: The breakpoint becomes an execution point. Processing does not halt at this point and the given code is executed. ● Execution point of a breakpoint: activated: ●; deactivated: ○ ● Execution point of a data breakpoint: activated: ●; deactivated: ○
"Execute the following code"	Code that is executed when the execution point is reached. Looping structures (For, While) and IF or CASE expressions are not possible.
"Print a message in the device log"	This option is available only when you select the *"Enable logging in breakpoints"* option in *"Project Settings ➔ Compile Options"*. CODESYS can print variables with the placeholder {variable name} in the message text.

Tab 'Location' Requirement: The command *"New breakpoint"* was selected.

"POU"	POU of the active application where the breakpoint is placed.
"Position"	Position of the breakpoint in the POU. Entry as row and column numbers (text editor) or as network or element numbers.
"Instances"	In the case of function blocks, you have to define whether the breakpoint should be set in the implementation or in an instance. ☑ CODESYS sets the breakpoint in the instance. For this option, select *"Instance Path"*. ☐ CODESYS sets the breakpoint in the implementation.

"Enable breakpoint immediately"	☑: The breakpoint is activated. ☐: The breakpoint is not activated. To activate later, click the button in the *"Breakpoints"* view.

See also

- *Chapter 1.4.1.20.4.5 "Dialog 'Breakpoint Properties'" on page 1151*
- *Chapter 1.4.1.11.2 "Using Breakpoints" on page 395*

Dialog 'Monitoring Range'

Function: This dialog restricts the range of array elements whose values are displayed during monitoring.

Call: Click in the column field *"Data Type"* that belongs to the array variable.

Requirement: A POU is in online mode and is being monitored. In addition, a variable of the POU has the data type *"ARRAY"*.

"Valid range"	The validity range of the array elements that are monitored. Example of a three-dimensional array: `[1..10][-3..3][-10..10]`
"Maximum number of array elements"	Number of elements of the array variables Example: `1470`
When you edit one of the settings *"Start"*, *"End"*, or *"Scroll range of 1000 elements"*, both of the other settings are adapted automatically.	

“Start”	Index of the first array element whose value is displayed.
“End”	Index of the last array element whose value is displayed.
“Scroll range of 1000 elements”	Scrollbar for selecting a range from the set of array elements.

See also

- *Chapter 1.4.1.12.1.1 “Calling of monitoring in programming objects ” on page 410*

Dialog 'Properties'

This dialog box is for the configuration of the properties of an object in CODESYS. In addition, depending on the object, it contains different tabs that each handle a category of properties.

Call: Menu *“View”*, context menu of the object in the *“Devices”*, *“POUs”* or *“Modules”* view.

Dialog Box 'Properties' - 'Common'

Function: This dialog box shows common information about the selected object.

Call: Main menu *“View ➔ Properties”*, or context menu of the object (*“Common”*).

Requirement: An object is selected in the device tree or POUs view.

Table 174

“Name ”	Object name as shown in the device tree or POUs view
“Object type ”	Type of object (for example, POU, application, or interface)
“Open with ”	Type of editor to display or edit the object

Dialog 'Properties' - 'Boot Application'

Function: The settings on this tab define when and how a boot application is created from the application.

Requirement: The device supports the settings.

Call: Select the application object; context menu: *“Properties”*; menu bar: *“View ➔ Properties”*, *“Boot Application”* category

“Create implicit boot application on download”	A boot application is created automatically when downloading the application.
“Create implicit boot application on Online Change”	A boot application is created automatically when for an online change.
“Remind boot application on project close”	Before closing the project, CODESYS prompts to create the boot application.
“Verify boot application after creation”	After the boot application is created, an independent service checks whether or not the boot application has been created correctly.

Regardless of the presets defined here, you are always able to create a boot application explicitly when you login.

See also

- *Chapter 1.4.1.10.6 “Generating boot applications” on page 391*
- *Chapter 1.4.1.20.3.6.6 “Command 'Online Change'” on page 1033*

Dialog 'Properties' - 'Encryption'

Function: The dialog contains the properties of the application for encryption. If the CODESYS Security Agent is installed, then you can start a wizard for the encryption of downloads, online changes, and boot applications.

Call:

- Menu bar: *“View ➔ Properties”*
- Context menu of an application object

Table 175: “Encryption Technology”

If the *“Enforce encryption of downloads, online changes and boot applications”* option is selected in the *“Security Screen”* view in the *“Security level”* group, then the encryption technology is set to *“Encryption with certificates”* and cannot be changed in this dialog.	
“No Encryption”	
“Simple Encryption”	You can download the boot application to the controller only when the defined dongle (license key) is connected to the computer. The dongle is provided by ABB AG or by the respective hardware manufacturer. The firmcode is displayed. Type in the delivered product code.

"Encryption with license management"	You can download the boot application to the controller only after you have specified the product code and firmcode, and the respective dongle is connected to both the development computer and the controller. You receive the codes from the vendor that manages the licenses.
"Encryption with certificates"	You can download the boot application to the controller only when a valid certificate exists for it. The *"Certificates"* group is enabled. See the description below. The option is already selected if the *"Enforce signing of downloads, online changes and boot applications"* option is selected on the *"User"* tab of the *"Security Screen"* view.

Table 176: "Certificates"

Note: If the *"Enforce encryption of downloads, online changes and boot applications"* option is selected in the *"Security Screen"* view in the *"Security level"* group, then the encryption technology is set to *"Encryption with certificates"* and cannot be changed in the *"Properties"* dialog.	
	The *"Certificate Selection"* dialog opens. Here you can select previously installed certificates of devices for which the encryption of download, online change, and boot application is enabled. The list can contain several entries if several devices are authorized to run this application.
"Digitally sign application code"	The application is signed with a digital signature. The certificate for the digital signature is specified in the *"Security Screen"* view on the *"User"* tab.
Area for the display of the selected certificates with corresponding information	Information per certificate: ● *"Issued for"* ● *"Issued by"* ● *"Valid from"* ● *"Valid until"* ● *"Thumbprint"*
"Encryption Wizard"	This button is available only if the CODESYS Security Agent is installed. It starts the wizard with the same name. See the help for CODESYS Security Agent in this case.

See also

- *Chapter 1.4.1.8.17 "Encrypting an application" on page 294*
- *Chapter 1.4.1.20.3.3.18 "Command 'Security Screen'" on page 995*
- Help about CODESYS Security Agent

Dialog 'Properties' - 'Build'

Symbol: 

Function: The dialog contains options for building (build operation) the object.

Call: Menu bar: *"View* ➔ *Properties"*; context menu of the object in the device tree

Name	Description
"Exclude from build"	☑: This object and recursively its child objects are not considered for the next compile process. The object entry is displayed in green fonts in the *"Devices"* or *"POUs"* view.
"External implementation" *"(Late link in the runtime system) "*	☑: CODESYS does not generate any code for this object when compiling the project. The object is linked as soon as the project is running on the target system, provided it is available there (for example, in a library). The object name is postfixed with `(EXT)` in *"Devices"* or *"POUs"* view.

Name	Description
“Enable system call”	☑: A system call (runtime system) for functions is possible. Background: As opposed to CoDeSys V2.3, the ADR operator in V3 can be used with function names, program names, function block names, and method names. It replaces the INSTANCE_OF operator. BUT: It is not possible to call function pointers from within CODESYS.
“Link always”	☑: The object is marked by the compiler and therefore always included in the compile information. This means that it is always compiled and downloaded to the PLC. Note: The pragma `{attribute 'linkalways'}` can also be used to instruct the compiler to always include an object.
“Compiler defines”	Here you can specify defines or conditions for compiling the object (conditional compile). You can also type in the expression `expr`, which is used in these kinds of pragmas. Multiple entries are possible as a comma-separated list (see `{define}` statements). Example: `hello, test:='1'`
“Additional compiler definitions from the device description”	
“Defined in device”	List of compiler definitions that originate from the device description. These compiler definitions are used in the build if they are not listed in the *“Ignored definitions”* field.
“Ignored definitions”	List of compiler definitions from the device description that are not used in the build.
>	Copies the selected compiler definition from the *“Defined in device”* field to the *“Ignored definitions”* field.
<	Moves the selected compiler definition from the *“Ignored definitions”* field to the *“Defined in device”* field. The compiler definition is used in the build.

See also

- *Chapter 1.4.1.19.6.3 “Conditional Pragmas” on page 732*
- *Chapter 1.4.1.19.6.2.24 “Attribute 'linkalways'” on page 708*

Dialog 'Properties' – 'Build' (C-integration)

Function: In this dialog, you configure the build environment and the necessary data for the integration of the C development environment.

Call: Main menu *“View”*, context menu of the object *“C Code Module”*

Requirement: The object *“C Code Module”* is selected in the device tree.

> **NOTICE!**
> The dialog in this form is valid only for CODESYS Control Win V3 and Visual Studio. For other environments, the dialog can look different or may not even be available at all.

“Visual Studio location”	Installation path of Visual Studio on the hard disk You can also select the path with the input assistant or search for it with the magnifying glass.
“Windows SDK location”	Installation path of Windows SDK on the hard disk You can also select the path with the input assistant or search for it with the magnifying glass.
“Temporary build folder Location”	Path on the hard disk for the temporary build files

See also

- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Dialog 'Properties' - 'Access Control'

Function: The dialog defines the access rights of user groups for objects.

Call: Main menu *“View ➔ Properties”*, context menu of an object in the view *“Device”* or *“POUs”*.

Requirement: An object is selected in the view *“Device”* or in the view *“POUs”*.

“Groups, actions and permissions”	A table which displays the following user groups access rights on objects: ● *“View”* ● *“Modify”* ● *“Remove”* ● *“Add/remove children”* Perform a double click on the access right symbol to open the drop down list with the available rights.

See also

- *Chapter 1.4.1.5.5 “Protecting Objects in the Project by Access Rights” on page 204*
- *Chapter 1.4.1.20.4.6 “Dialog 'Permissions'” on page 1152*

Dialog 'Properties' - 'External file'

Function: The dialog is used to view and edit the properties of the external file. The properties were defined when the object was created. Changed properties are saved by pressing the *“OK”* button.

Call: Menu bar: *“View ➔ Properties”*; context menu of the object

Requirement: The object of the external file is selected in the *“Devices”* view or the *“POUs”* view.

Table 177

“What do you want to do with the external file?”	
“Remember the link”	The file is available in the project only as long as it exists in the defined file path.

“Remember the link and embed into project”	CODESYS saves an internal copy of the file in the project, as well as the link to the defined file path. The update option selected below applies as long as the external file exists there. Otherwise CODESYS uses the version saved in the project.
“Embed into project”	CODESYS saves only one copy of the file in the project. There is no longer a link to the external file.

Table 178: “When the external file changes, then”

“Reload the file automatically”	If the external file changes, then CODESYS updates the file in the project.
“Prompt whether to reload the file”	If the external file changes, then a dialog prompt opens whether CODESYS should also update the file in the project.
“Do nothing”	The file remains unchanged in the project, even if the external file changes.

Table 179: “Linked File”

Requirement: Either the *“Remember the link”* option or the *“Remember the link and embed into project”* option is selected.	
The following information about the linked file is displayed: *“Name”*, *“Location”*, *“Size”*, *“Changed”*.	
“Display File Properties”	Clicking this button opens the default *“Properties of <file name>”* dialog, which you can also open in the Windows file system by right-clicking the file.

Table 180: “Embedded file”

Requirement: Either the *“Remember the link and embed into project”* option or the *“Embed into project”* option is selected.	
Display of the following information about the embedded file: *“Size”* and *“Changed”*	
“Update the embedded file”	☑: If the external file that was added to the project is changed in the specified file path, then CODESYS updates the embedded file in the project.

See also

- *Chapter 1.4.1.20.2.7 “Object 'External File'” on page 838*

Dialog Box 'Properties' - 'Bitmap'

Function: The dialog is for assigning a bitmap file to the object. The image will be used in the graphic view of the Library Manager and in the Toolbox view of the FBD/LD/IL editor.

Call: Main menu *“View ➔ Properties”*, context menu of the object

Requirement: The object is selected in the view *“Devices”* or in the view *“POUs”*

“Render pixels of this color translparently: ”	The selected color will be displayed transparently.

Dialog 'Properties - Application Build Options'

Function: The dialog includes settings that CODESYS uses for creating a boot application for the controller.

Call: Menu bar: *“View ➔ Properties”*; context menu of an application object

“Download application info”	This feature requires compiler version >=3.5.0.0, runtime version >= 3.5.0.0. The information about the application contents is also downloaded to the PLC. We recommend that you keep this option enabled because it allows for a difference check between the current application and the application on the PLC. This compares the number of blocks, data, and memory locations. To get the information about the differences, click *“Details”* in the *“Applications”* tab of the device editor. This is also in the message view that opens when you are downloading an application to the PLC when it is different from the one already on the PLC.
“Stop parent application in case of exception”	Available for applications with a parent application.
“Dynamic memory settings”	Memory is allocated dynamically for the application, for example when using the operator `__NEW`. In this case, define the *“Maximum size of memory (bytes)”*. Caution: The entire memory is not available for creating objects dynamically. Instead, the system always uses part of it for management information.

See also

- *Chapter 1.4.1.10.4 “Generating Application Code” on page 389*
- *Chapter 1.4.1.10.6 “Generating boot applications” on page 391*
- *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*

Dialog 'Properties' - 'Target memory settings'

Function: The dialog allows for changing the memory settings of the target device.

Call: Menu bar: *“View ➔ Properties”*; context menu of the application

Requirement: The application is selected in the *“Devices”* view.

“Override target memory settings”	☑: The memory settings stored in the device description are overridden by the values specified in *“Input size”*, *“Output size”*, and *“Memory size”*. Note: If the memory settings of the target device are changed, then it is no longer possible to log in to an existing application on the target device, nor is it possible to perform an online change.
“Input size” *“Output size”* *“Memory size”*	Input fields for the memory sizes used to override the values `"memory-layout\\input-size"`, `"memory-layout\\output-size"`, and `"memory-layout\\memory-size"` stored in the device description. Requirement: The *“Override target memory settings”* option is selected.

See also

- *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*

Dialog 'Properties' - 'Network Variables'

Symbol:

Function: In this dialog, you define network properties for the variable list that is selected in the device tree. Furthermore, any variables in it that are declared as network variables are also available.

Call: *“Context menu of variable list in device tree ➔ Properties”*, *“Network Variables”* tab

“Network type”	UDP
“Task”	Task of the current application that controls the variables to be sent. CODESYS always sends the variables at the end of a task cycle.
“List identifier”	Used to identify the network variable list. Must be unique
“Pack variables”	The size of the packages (telegrams) that are transmitted depends on the network type. In the case of *“UDP”*, a package is 256 bytes. ☑: CODESYS bundles the variables for sending in packages in order to reduce as much as possible the number of packages to send. In the case of variables of type array or structured data types, this can lead to the splitting of the variables into multiple telegrams. As a result, data inconsistencies are possible within these variables, even if the variable size is smaller than the package size. ☐: CODESYS generates one package per variable.
“Transmit checksum”	☑: A checksum is provided for each variable package. The receiver checks the checksum to make sure that the variable definitions match from the sender and receiver. A package with non-matching checksums is not accepted.
“Acknowledgement”	☑: CODESYS sends an acknowledgement message for each received data package. If the sender does not receive a confirmation before it sends again, then an error is written to the diagnostic structure. Note: For the `NetVarUdp` library version 3.5.7.0 and later, a receiver channel is no longer assigned when confirmed transfer is not selected. In this way, network variable exchange is also possible between two controllers on one hardware device .
“Cyclic transmission”, *“Interval”*	CODESYS sends the variables within the defined interval. Example for time definition: "T#70ms".
“Transmit on change”, *“Minimum gap”*	☑: CODESYS sends the variables only if their values have changed. You can use "minimum gap" to define the least amount of time between two transmissions.
“Transmit on event”, *“Variable”*	☑: CODESYS sends the variables as soon as the defined variable yields `TRUE`.
“Settings”	Protocol-specific settings; possible entries depend on the network library. *“Port”*: Number of the port that CODESYS uses for data exchange with other network units. The *“Default value”* is "1202". You can change the current value in the *“Value”* field at any time. Select the field, press the *[Space Bar]*, and type the value. **Caution**: The other nodes in the network must define the same port. If more than one UDP connection is defined in the project, then the port numbers in all configurations are adapted to this value. *“Broadcast Adr.”*: The *“Default value”* is 255.255.255.255, which means that data exchange will take place with all network units. You can change the current value in the *“Value”*: select the field, press the *[space bar]*, and type the address or address range of a subnetwork (for example, `197.200.100.255` when communication should be with all nodes that have an IP address in the range 197.200.100.x.

See also

- *Chapter 1.4.1.9.3 “Network Variables” on page 360*
- *Chapter 1.4.1.9.3.1 “Configuring a Network Variable Exchange” on page 361*
- *Chapter 1.4.1.20.2.17 “Object 'Network Variable List (Receiver)'” on page 880*
- *Chapter 1.4.1.20.2.16 “Object 'Network Variable List (Sender)'” on page 880*

Dialog 'Properties' - 'Network Settings'

If the device supports the network functionality, then the current network settings for a GNVL (global network variable list) can be displayed and changed in the *"Properties"* dialog of the object. These are the settings that were used when adding the GNVL in the *"Add Network Variable List (Receiver)"* dialog.

See also

- *"Dialog Box 'Add Network Variable List (Receiver)'" on page 880*
- *Chapter 1.4.1.9.3.1 "Configuring a Network Variable Exchange" on page 361*

Dialog 'Properties' - 'CFC Execution Order'

Function: The tab switches the mode of the execution order for CFC objects.

Call: Context menu: *"Properties"* of a CFC object in the *"Devices"* view or *"POUs"* view

Tab 'CFC Execution Order'

"Execution order"	In the CFC editor, you position the elements and therefore also the networks freely. Two modes are available to prevent the execution order in the CFC POU from being undefined.
"Auto Data Flow Mode"	In this mode, the execution order is determined automatically by data flow, or in case of ambiguity, by network topology. The POUs and the outputs are numbered internally. The networks are executed from top to bottom and left to right. Advantage: The automatically defined execution order is optimized by time and by cycle. You do not need any information about the internally managed execution order during the development process. The following commands are provided afterwards in the *"CFC ➔ Execution Order"* menu: ● *"Display Execution Order"* ● *"Set Start of Feedback"* The elements in the CFC editor are displayed without markers and without numbering. It is not possible to change the execution order manually. For networks with feedback, you can also set a starting point.
"Explicit Execution Order Mode"	In this mode, you can define the execution order explicitly. To do this, the elements are displayed in the CFC editor with markers and numbering, and menu commands are provided for defining the order. The following commands are provided in the *"CFC ➔ Execution Order"* menu: ● *"Send to Front"* ● *"Send to Back"* ● *"Move Up"* ● *"Move Down"* ● *"Set Execution Order"* ● *"Order by Data Flow"* ● *"Order by Topology"* Note: Up to CODESYS V3.5 SP1, this was the usual behavior of CFC POUs. Pay attention that it is your responsibility to adapt the execution order and assess the consequences and impacts. This is another reason why the execution order is always displayed.
"Apply to All CFCs"	Changes the mode for all other CFC objects in the project to the mode selected in the list

See also

- *Chapter 1.4.1.8.3.2.1 “Automatic Execution Order by Data Flow” on page 242*
- *Chapter 1.4.1.20.3.12.10 “Command 'Set Start of Feedback'” on page 1092*
- *Chapter 1.4.1.20.3.12.11 “Command 'Send to Front'” on page 1092*
- *Chapter 1.4.1.20.3.12.12 “Command 'Send to Back'” on page 1093*
- *Chapter 1.4.1.20.3.12.13 “Command 'Move Up'” on page 1093*
- *Chapter 1.4.1.20.3.12.14 “Command 'Move Down'” on page 1094*
- *Chapter 1.4.1.20.3.12.16 “Command 'Order by Data Flow'” on page 1095*
- *Chapter 1.4.1.20.3.12.17 “Command 'Order by Topology'” on page 1095*

Dialog 'Properties' - 'SFC Settings'

Function: The dialog defines the default settings for all POUs used in the project, which are programmed in SFC.

Call: Main menu *“View ➔ Properties”*, context menu of a SFC POU in the view *“Device”* or *“POUs”*.

Flag

List of all possible SFC flags	*“Use”* ☑: The SFC flag is activated and will be considered in the program execution. *“Declare”* ☑: The SFC flag is declared automatically. If *“Declare”* is activated, but *“Use”* is not activated, the variable will be declared but the flag has no effect in the program execution. Hint: If you have manually declared a SFC variable you have to disable the declaration of this flag in the *“SFC Settings”*. Otherwise the automatically generated flag will overwrite the manually declared flag. Hint: A automatically declared flag variable is only visible in the online mode in the declaration part of the SFC editor.
“Use defaults”	☑: The settings of this dialog overwrites the *“SFC settings”* of the single POUs.

See also

- *Chapter 1.4.1.19.1.4.6 “SFC Flags” on page 481*

Build

Table 181: “Code generation”

“Calculate active transitions only”	☑: CODESYS generates code for the currently active transition only.

Dialog 'Properties' – 'Link to File'

Function: The dialog defines the link of an external file with the contents of the global variable list (GVL). You can either export the GVL to an external file or import it from an external file.

Call: Menu bar: *“View ➔ Properties”*; context menu of an object of type *“Global Variable List”*

“File name”	Input field of the file path
“Export before compile ”	☑: Before each compile of the project (for example with *[F11]*), CODESYS saves a file with the extension `gvl` in the path, which is specified in the *“File name”* field.
“Import before compile ”	☑: The export file which is specified in the *“File name”* field is read automatically before each project compile. Therefore you can import a GVL which was exported from another project, for example to set up a communication by means of network variables.

See also

- *Chapter 1.4.1.20.2.10 “Object 'GVL' - Global Variable List” on page 871*
- *Chapter 1.4.1.9.3.1 “Configuring a Network Variable Exchange” on page 361*

Dialog 'Properties' - 'Cam'

Function: Use this dialog to define the global variables of the cam.

Table 182: “Dimensions”

“Master start/end position”	The start and end positions of the master define the range of the master values and therefore the scale of the horizontal axis of the cam. The default settings are given in angular degrees with 0 and 360 as limiting values.
“Slave start/end position”	The associated slave positions are determined by the graph type that is defined for the cam. However, the segment depicted by the curves (this is also the scale of the vertical axis) can be defined by the slave start and end positions given here.

Table 183: “Period”

These settings affect the work in the cam editor and cam table. Depending on these parameters, the slave start point is adjusted automatically when the end point is changed, as well as the other way around. This adjustment optimizes the period transition to be as smooth and jerk-free as possible.	
“Smooth transition”	☑: The values for position, velocity, and acceleration are adjusted automatically.
“Slave period”	Indicates when the slave period is repeated mechanically. The slave position at the start and end of the master period may then be in an interval of a whole number multiple of this value. This value is effective only if the *“Smooth transition”* check box is selected.

Table 184: “Continuity Requirements”

Activation of these options for the continuity of the curve does not have any effect when editing the cam. It does, however, prompt a continuity check, which reports any violations to the message view (CAM). It is not possible to edit jumps in the position curve. The default setting also requires the continuity of velocity and acceleration. You can clear these options, for example in the special case of a curve that consists of only linear segments. However, this will lead to kinks in the position curve. By default, the jerk (3rd derivative) is not tested for jumps.	
“Position”	☑: The entire curve is tested for jumps.
“Velocity”	
“Acceleration”	
“Jerk”	

Table 185: “Compile Format”

When compiling, `MC_CAM_REF` structure variables are generated. A cam is described according to the following options:	
“polynomial (XYVA)”	Polynomial description of the individual points consisting of the master position, slave position, slave velocity, and slave acceleration.
“one dimensional point array”	1D table of slave positions
“two dimensional point array”	2D table of composite master/slave positions
“Elements”	Number of elements in the arrays. This array has already been created in SM3_Basic for the default cases *“128”* and *“256”*. If you type in another value, you must create the structure in your application (see the following example).

Example of an array with 720 elements

```
TYPE SMC_CAMTable_LREAL_720_2 :
STRUCT
    Table: ARRAY[0..719] OF ARRAY[0..1] OF LREAL;
    fEditorMasterMin, fEditorMasterMax: REAL;
    fEditorSlaveMin, fEditorSlaveMax: REAL;
    fTableMasterMin, fTableMasterMax: REAL;
    fTableSlaveMin, fTableSlaveMax: REAL;
END_STRUCT
END_TYPE
```

Dialog 'Properties' - 'Image Pool'

Function: The dialog allows for setting the basic properties of the selected image pool.

Call: *“View* ➔ *Properties”* of an *“Image Pool”* object type; context menu of an *“Image Pool”* object type.

“Download only used images”	☑: Instead of loading all images from the image pool, CODESYS loads only the images that are actually used in the application on the PLC.
“Download by visualization”	☑: The image pool is downloaded with the visualization to the controller.
“Internal”	☑: CODESYS does not provide the image pool in the *“ToolBox”* view. You cannot drag these images to the visualization.

Table 186: “Symbol library settings”

“Mark library as symbol library”	Marks the image pool as a symbol library for use in a visualization. The symbol library receives the key `VisuSymbolLibrary = TRUE` as file property in the project information. The `VisuElements` library is inserted automatically as a placeholder library in the *“POUs”* pool of the Library Manager. Requirement: A library project is open. CODESYS displays symbol libraries that are installed in the repository in the *“Project Settings”* (*“Visualization”* category, *“Symbol Libraries”* tab).
“Textlist for symbol translation”	Select the text list from the drop-down list that contains the translated texts for the image pool.

See also

- *Chapter 1.4.1.20.2.13 “Object 'Image Pool'” on page 873*
- *Chapter 1.4.1.8.9 “Using image pools” on page 274*
- *Chapter 1.4.1.20.4.11.9 “Dialog 'Project Settings' - 'Visualization'” on page 1180*

Dialog 'Properties' - 'TextList'

Function: The dialog allows for setting the basic properties of the selected text list.

Call: *“View ➔ Properties”* of an *“Text List”* object type; context menu of an *“Image List”* object type.

“Download by visualization”	☑: The text list is downloaded with the visualization to the controller.
“Internal”	☑: The text list can be used only in a library. It is not available in an ordinary CODESYS project.

See also

- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

Dialog 'Properties' - 'Options'

Options (Controller)

Function: This dialog provides the settings for monitoring an login for objects of type device. The availability of the options depends on the device description.

Call: Context menu of the device, or main menu *“View ➔ Properties”*, if the device is selected.

“Monitoring interval (ms)”	Interval of the monitoring (10 ms - 1000 ms)

Table 187: “Interactive Login Mode”

This mode is used to prevent an accidentally login to a different controller.	
“None”	No interaction with the user during login. Corresponds to the behavior of previous versions.
“Enter ID”	During login CODESYS asks to enter an ID. The ID is stored in the controller. Without a valid ID no login is possible. When login a second time, CODESYS does not ask again for the ID if the computer name, the user name, the device name and the device address have not changed. The information is saved in the project options.
“Press key”	During login a dialog prompts and requests the user to press a key on the controller. The timeout for this action is saved in the device description.
“Wink (= blink an LED)”	During login a led blinks on the connected controller.

Table 188: “Symbol Configuration”

“Access variables in sync with IEC tasks”	☐: Default setting, consistent access is not permitted ☑: Consistent access is permitted The setting only will take effect when all applications and boot applications are re-downloaded to the controller. Note: If the option is activated, then the jitter for all IEC applications may increase on this device! The consistent access can disturb the real-time capability.

Siehe also

- *Chapter 1.4.1.20.2.8 “Object 'Device' and Generic Device Editor” on page 839*
- *Chapter 1.4.1.20.2.25 “Object 'Symbol Configuration'” on page 927*

Dialog 'Properties' - 'Monitoring'

Function: The tab contains options for the monitoring of transitions in SFC.

Call: Select transition object, click *“Properties”*; menu bar: *“View ➔ Properties”*.

“Enable monitoring”	☑: An implicit variable is created for the transition, which is then always given the current property value when the application calls the Transition method. The value stored last in this variable is displayed in the monitoring.
“Monitoring using call”	☑: The transition to be monitored is read by directly calling the transition. Note: When you activate this option, you have to consider possible side effects. These kinds of side effects can occur if additional operations are implemented in the transition.

See also

- *Chapter 1.4.1.20.2.18.10 “Object 'Transition'” on page 903*
- *Chapter 1.4.1.19.6.2.25 “Attribute 'monitoring'” on page 709*

Dialog 'Project Settings'

Symbol: 

Function: The object contains the basic configuration of the project. In the *“Project Settings”* dialogs the configuration can be adjusted.

Call: Double click on the *“Project Settings”* object in the device tree, or main menu *“Project ➔ Project Settings”*.

CODESYS saves the project settings directly in the project. If a project is transferred to another system for example the *“Project Settings”* object is transferred as well without the need of a project archive.

The project settings are valid project wide. Dependent on the installed packages the dialogs provide settings for several categories, as for example *“SFC”* or *“User and Groups”*.

See also

- *Chapter 1.4.1.20.3.8.4 “Command 'Package Manager'” on page 1059*

Dialog 'Project Settings' - 'SFC'

Symbol: 

Function: This dialog is used for configuring the settings of SFC objects. The properties of each new SFC object automatically have the configured settings.

Call: Menu bar: *“Project ➔ Project Settings”* (*“SFC”*).

Requirement: A project is open.

Tab 'Flags'

Implicitly generated variables for checking and monitoring the processing in an SFC diagram	
“Active”	☑: The corresponding variable is used.
“Declare”	☑: The corresponding variable is created automatically. Otherwise, you have to declare the variable explicitly if you intend to use it (*“Use”* is selected).

“Apply to all”	In this dialog, CODESYS applies changes to existing SFC objects. CODESYS selects the *“Use defaults”* check box in the properties of the SFC POUs.

NOTICE!
Automatically declared variables are visible in the declaration part of the SFC editor only in online mode.

Tab 'Build'

Table 189: “Code Generation”

“Calculate active transitions only”	☑: CODESYS generates code only for currently active transitions.

Table 190: “SFC Library”

This part of the dialog is available only for compiler versions < 3.4.1.0.	
“Company ” *“Title”* *“Version”*	Defines the SFC library that CODESYS uses by default.
“Namespace”	Enables unique references to libraries. Required when various versions of the library are available on the system. Please make sure that there are no discrepancies between the namespace defined in the library manager and the namespace defined for the individual object. The `SfcIec.library` data is used for the default settings that CODESYS provides with the default profile.

Each SFC block stores the information via the library version that applied when you added the block. This can cause you to use multiple library versions within the same project. In order to prevent this, you are prevented from defining specific versions of `IecSfc.library` (as of compiler version 3.4.1.0). The library version, which you use for all SFC blocks in the project, is defined with a placeholder. CODESYS resolves the placeholder depending on the compiler version in use. The allocation of the library version to the compiler version is defined in the library profile.

See also

- *Chapter 1.4.1.19.1.4.6 “SFC Flags” on page 481*
- *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*

Dialog 'Project Settings' - 'Users and Groups'

Symbol: 

Function: This dialog is for the configuration of the user management for the current project.

Call: Menu *“Project ➔ Project Settings”*, category *“Users and Groups”*

Tab 'User'

Displays the users and their memberships in groups	
“Add”	Opens the *“Add User”* dialog.
“Edit”	Opens the *“Edit User”* dialog.
“Delete”	An error message appears if you attempt to delete the last user of a group, since a group must have at least one member.

Table 191: “Add User / Edit User”

Input fields for setting up a new user account or changing an existing one	
“Active”	☑ : You may use the user account, default ☐: The user cannot log in. If he attempts to login again with incorrect login data, this can result in automatic deactivation of the account; see below: Settings.
“Memberships”	List of all user groups that you have defined in addition to the group *“Everyone”* (to which each new user automatically belongs). ☑<group name> : the new user belongs to the group.

Table 192: “Export/Import”

“Export Users and Groups”	The command opens the standard dialog for saving a file in the local file system. You can store the users and groups definitions of the project in a file `*.users` in xml format.
“Import Users and Groups”	Export users and groups opens the standard dialog for browsing the local file system for a file. Search for a file with extension `*.users` in order to import the users and groups definiitions, stored in this file, into the project.

Tab 'Groups'

Display of the groups and their members. A group can also be a member of a group.	
“Add”	Opens the *“Add Group”* dialog.
“Edit”	Opens the *“Edit Group”* dialog.
“Delete”	If you delete a group, the user accounts of the members remain unchanged. You cannot delete the groups *“Everyone”* and *“Owner”*.

On button *“Export/Import”* please see above the *“User”* paragraph.

Tab 'Settings'

Display of the groups and their members in a tree structure. A group can also be a member of a group.	
"Maximum number of authentication trials"	☑ (standard) : If the user has attempted to login with an incorrect password the number of times specified here, the user account is deactivated. ☐: The number of the unsuccessful attempts is unlimited
"Automatic logoff after time of inactivity"	☑: The user is automatically logged out if CODESYS does not register any user actions by mouse or keyboard during the time period (minutes) specified here.

See also

- *Chapter 1.4.1.5 "Protecting and Saving Projects" on page 197*
- *Chapter 1.4.1.20.3.4.28 "Command 'User management' – 'Log in User'" on page 1016*

Dialog Box 'Project Settings' - 'Compileoptions'

Symbol:

Function: This dialog box is for configuring the compiler options.

Call: Main menu *"Project ➔ Project Settings"* (*"Compileoptions"* category).

Requirement: A project is open.

Table 193: "Compilerversion"

"Fix Version"	Defines the compiler version that CODESYS uses when compiling and downloading for compile (for example, *"3.5.6.0"* for version 3.5 SP6).

Table 194: "Settings"

"Allow unicode characters for identifiers"	Cleared by default because using Unicode characters in identifier names is not permitted in the IEC standard. May be required for some foreign languages (for example, Asian languages).
"Replace constants"	☑ (default): CODESYS loads the value directly for every scalar constant (not for strings, arrays, and structures). In online mode, CODESYS marks the constants with a symbol that is prepended to the value in the declaration editor or monitoring view. In this case, access is not possible, for example by means of an `ADR` operator, forcing, and writing. ☐: Access to constants is possible, but it prolongs the computation time.
"Enable logging in breakpoints"	For breakpoints that are defined as execution points, you can create a message text in the *"Execution point settings"* dialog box. CODESYS prints this text to the device log when the application halts at the execution point.

Table 195: "Compiler Warnings"

"Maximum number of warnings"	Refers to the warnings that CODESYS prints to the messages view. You define the selection of displayed compiler warnings in the *"Project Settings"* dialog box in the *"Compiler Warnings"* category.

See also

- *Chapter 1.4.1.20.4.11.4 "Dialog Box 'Project Settings' - 'Compiler Warnings'" on page 1173*
- *Chapter 1.4.1.20.2.8.8 "Tab 'Log'" on page 848*

Dialog Box 'Project Settings' - 'Compiler Warnings'

Symbol:

Function: This dialog box is used for selecting the compiler warnings that CODESYS displays in the messages view during a compile process.

Call: Call: Main menu *“Project ➔ Project Settings”* (*“Compiler Warnings”* category).

Requirement: A project is open.

> *You define the maximum number of listed warnings in the “Compileoptions” dialog box.*

See also

- *Chapter 1.4.1.20.4.11.3 “Dialog Box 'Project Settings' - 'Compileoptions'” on page 1173*
- *Chapter 1.4.1.20.3.5.4 “Command 'Build'” on page 1022*

Dialog 'Project Settings' – 'Source Download'

Symbol: 

Function: This dialog defines the compilation and the storage of the source code as a source-code download archive on one or more controllers.

Call: *“Project ➔ Project settings”* menu, *“Download source code”* category

A source-code download archive is a project archive with the name `Archiv.prj`.

> *The settings affect the command “Online ➔ Load source code to connected controller”. These settings do not affect the command “File ➔ Load source code to controller”.*

Table 196: “Destination device”

Defines the location of the project archive.	
“<name of controller>”	Selected controller. CODESYS loads the project archive to this controller. Requirement: the project contains several controllers.
“<all devices in project>”	CODESYS loads the project archive to all controllers in the project.

Table 197: “Content”

Defines the contents of the project archive.	
“Use compact download”	☑: The project archive contains only that device in the project that contains the active application. ☐: The project archive contains all the devices in the project

“Additional Files”	Opens the *“Additional files”* dialog where you can select additional files for downloading.
	Not all types of additional files are available for each project. *“Download information files”* - Project information files *“Library profile”* - Includes the applied profile *“Project information”* - Includes the project information *“Referenced devices”* - Includes all device descriptions of third party devices into the archive *“Referenced libraries”* - Includes all referenced libraries into the archive *“Referenced visualisation styles”* - Includes the used styles *“Visualisation profile”* - Includes the used profile The most important types *“Referenced devices”* and *“Referenced libraries”* should always be included, if the archive shall be usable by Automation Builder installations without availability of the required devices or libraries.

Table 198: “Timing”

Defines the time at which CODESYS creates a project archive.	
“Implicitly at program download and online change”	Each time an application is loaded or an online change is made, CODESYS additionally loads the project archive to the target device(s) with no further prompt.
“Implicitly at creating boot project”	Each time a boot application is created, CODESYS additionally loads the project archive to the target device(s) with no further prompt.
“Implicitly at creating boot project, download and online change”	Each time a boot application is created, an application is loaded or an online change is made, CODESYS additionally loads the project archive to the target device(s) with no further prompt.
“Prompt at program download and online change”	Each time an application is loaded or an online change is made, CODESYS opens a prompt, where you can select whether CODESYS should load the project archive to the controller.
“Only on demand”	A prompt opens only if the command *“Online ➔ Load source code to connected controller”* is called. There you can select whether CODESYS should load the project archive to the controller.

See also

- ↬ *Chapter 1.4.1.10.7 “Downloading source code to and from the PLC” on page 393*
- ↬ *Chapter 1.4.1.20.3.6.7 “Command 'Source Download to Connected Device'” on page 1035*
- ↬ *Chapter 1.4.1.20.3.1.11 “Command 'Source Download'” on page 963*

Dialog 'Project Settings' - 'Page Setup'

Symbol: 

Function: This dialog defines the layout for the print version of the project contents. This layout is used for the printout of the project information by clicking *“File ➔ Print”* and the printout of the project documentation by clicking *“Project ➔ Document”*.

Call: Main menu *“Project ➔ Project Settings”* (*“Page Setup”*)

You can change settings the following:

- *“Paper”*
- *“Margins”*

- *“Header and Footer”*
- *“Document”*
- *“Title Page”*

Table 199: “Edit Header, Edit Footer”

The headers and footers are structured in table style. You can configure rows and columns, and add text and images to the resulting cells.	
“Row spanning”	Number of rows that CODESYS should merge into a single column.
“Column spanning”	Number of columns that CODESYS should merge into a single row.
	Opens the list of available placeholders for the *“Text”* field. When printing the page, CODESYS provides the placeholders with the current values.

See also

- *Chapter 1.4.1.20.3.1.14 “Command 'Page Setup'” on page 964*
- *Chapter 1.4.1.20.3.4.19 “Command 'Document' ” on page 1009*
- *Chapter 1.4.1.20.3.1.12 “Command 'Print'” on page 963*

Dialog 'Project Settings' - 'Security'

Symbol: 

Function: this dialog is for the configuration of the project protection by a password, a dongle, or a certificate.

Call: Menu bar: *“Project ➔ Project Settings”* (category *“Security”*).

> **NOTICE!**
> If the encryption password is lost you can no longer open the project. You can also no longer restore it.

“No protection”	: The project file is not protected from unauthorized access and data manipulation. Note: We strongly recommend that you use security functionality. : The *“Password”*, *“Dongle”*, and *“Certificates”* options cannot be selected.
“Integrity check”	When you create a new project, this option is enabled by default. : The project file is stored in a proprietary format and its integrity is checked each time the project is loaded. The file may be incompatible with older versions of the development system. Please note that the project file is not encrypted. To better protect your data, activate one of the encryption functions.
“Encryption”	: The *“Password”*, *“Dongle”*, and *“Certificates”* encryption functions can be selected.
“Password”	Entering, changing and confirming the encryption password. If you save the project with these settings you must enter the password later in order to open the project again, even if it is to be loaded as a library reference.
“Dongle”	Requirement: you have connected the CODESYS security key (dongle) to the computer. *“Add”*: The dialog *“Add Registered Dongle”* opens.

“Registered dongles”	Drop-down list of the registered dongles.
“Certificates”	Certificates are used for the encryption of contents of the open project file. Requirement: The certificates for all users who share the project must be installed in the local memory. : The *“Certificate selection”* dialog opens.

Table 200: Adding a registered dongle

“Dongle”	Drop-down list of all connected dongles.
“Update”	CODESYS refreshes the drop-down list.
“Flash”	The LEDs of the currently selected dongle flash for two seconds (if it supports this function).

The dongle must be connected to the computer when CODESYS loads the project, even if it is loaded as a library reference.

See also

- *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*
- *Chapter 1.4.1.5.2 “Assigning Passwords” on page 202*
- *Chapter 1.4.1.5.3 “Protecting Projects Using a Dongle” on page 203*
- *Chapter 1.4.1.5.7 “Encrypting Projects with Certificates” on page 207*
- *Chapter 1.4.1.20.4.18 “Dialog 'Certificate Selection'” on page 1215*

Dialog 'Project Settings' - 'Static Analysis Light'

Symbol: §

Function: This dialog activates the tests that the light version of CODESYS Static Analysis performs each time code is generated.

Call: Menu bar: *“Project ➔ Project Settings”* (*“Static Analysis Light”* category).

You can exclude lines of code from the static code analysis by marking the code with the pragma `{analysis ...}` or the pragma `{attribute 'analysis' := '...'}`.

Additional compile tests

“SA0033: Unused variables”	Finds variables that are declared, but not used within the compiled program code. For GVL variables: If there are multiple applications in one project, then only the objects under the currently active application are affected. If there is only one application, then the objects in the `POUs` view are also affected.
“SA0028: Overlapping memory areas”	Detects the locations where two or more variables reserve the same storage space. For example, this occurs for the following declarations: `var1 AT %QB21: INT` and `var2 AT %QD5: DWORD`. In this case, both variables use byte 21, which means that the memory range of the variables overlap.
“SA0006: Write access from multiple tasks”	Detects variables that are written by more than one task.

SA0004 *“Multiple write access on output”*	Detects outputs that are written to more than one location. Note: No error is reported when an output variable (`VAR_IN_OUT`) is written in different branches of `IF` and `CASE` statements. Note: A pragma cannot deactivate this rule.
“SA0027: Multiple use of identifiers”	Detects multiple uses of a name/identifier for a variable or an object (POU) within the scope of a project. The following cases are detected: ● The name of an enumeration constant is the same as in another enumeration in the application or used in an included library. ● The name of a variable is the same as an object in the application or an included library. ● The name of a variable is the same as for an enumeration constant in and enumeration in the application or an included library. ● The name of an object is the same as another object in the application. ● The name of a variable is the same as the name of a method. ● The name of an object is the same as the name of a superordinate object ("parent object").
“SA0167: Temporary function block instances”	The test detects function block instances that are declared as temporary variables. This concerns instances that are declared in a method or in a function or as `VAR_TEMP`, and therefore are reinitialized in each processing cycle and for each POU call.

Examples

SA0003: Empty statements

```
;
(* Comment *);
iVar;
```

SA0006: Concurrent access

```
FUNCTION_BLOCK ADD_FB
g_iTemp1 := g_iTemp1 + INT#1;

PROGRAM PLC_PRG  //controlled by MainTask
g_iTemp1 := g_iTemp1 + INT#2;
g_xTemp2 := g_iTemp1 > INT#10;

PROGRAM PLC_PRG_1  //controlled by SubTask
g_iTemp1 := g_iTemp1 - INT#3;
g_xTemp2 := g_iTemp1 < INT#-10;
```

SA0004 **Multiple write access on output**

```
VAR_GLOBAL
    g_xVar AT %QX0.0 : BOOL ;
    g_iTest AT %QW0 : INT ;
END_VAR

PROGRAM PLC_PRG
IF iCondition < INT#0 THEN
    g_xVar := TRUE;
    g_iTest := INT#12;
END_IF

CASE iCondition OF
    INT#1:
        g_xVar := FALSE;
    INT#2:
        g_iTest := INT#11;
    ELSE
        g_xVar := TRUE;
        g_iTest := INT#9;
END_CASE
```

SA0006: Write access from multiple tasks

```
FUNCTION_BLOCK ADD_FB
g_iTemp1 := g_iTemp1 + INT#1;

PROGRAM PLC_PRG  // Controlled by MainTask
g_iTemp1 := g_iTemp1 + INT#2;
g_xTemp2 := g_iTemp1 > INT#10;

PROGRAM PLC_PRG_1  //Controlled by SubTask
g_iTemp1 := g_iTemp1 - INT#3;
g_xTemp2 := g_iTemp1 < INT#-10;
```

SA0027: Multiple use of name

```
PROGRAM PLC_PRG
VAR
ton : INT;     // error SA0027
END_VAR
```

SA0029: Different notation in implementation and declaration

The `PLC_PRG` POU and a `fnc` function POU are in the device tree.

```
PROGRAM PLC_PRG
VAR
    iVar:INT;
    _123test_var_: INT;
END_VAR
    ivar := iVar + 1; // notation different to that in the
declaration part ->   SA0029
    _123TEST_var_ :=     _123test_var_INT; // notation different to
that in the declaration part ->   SA0029
 Fnc(); // notation different to that in the devices tree ->
SA0029
END_VAR
```

SA0167: Temporary function block instances

```
PROGRAM PLC_PRG
VAR
END_VAR
VAR_TEMP
    yafb: AFB;
END_VAR

FUNCTION Fun : INT
VAR_INPUT
END_VAR
VAR
    funafb: AFB;
END_VAR

METHOD METH: INT
VAR_INPUT
END_VAR
VAR
    methafb: AFB;
END_VAR
```

See also

- *Chapter 1.4.1.8.12.2 “Analyzing code statically” on page 283*

Dialog 'Project Settings' - 'Visualization'

Symbol: 

Function: The dialog is used to configure the project-wide settings for objects of type *“Visualization”*.

Call: Menu bar: *“Project* ➔ *Project Settings”*, *“Visualization”* category

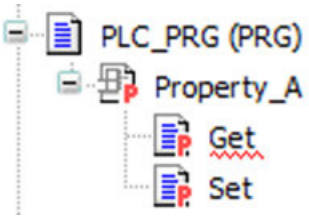
Requirement: A project is open.

Tab 'General'

Table 201: "Visualization Directories"

"Text list files"	Directory which contains text lists that are available in the project to configure texts for different languages. CODESYS uses the directory, for example to import or export text lists. After clicking ..., the *"Select Directory"* dialog opens which allows for the selection of a directory in the file system.
"Image files"	Directory which contains image files that are available in the project. Multiple folders are separated with a semicolon. CODESYS uses the directory, for example to import or export image files. After clicking ..., the *"Select Directory"* dialog opens which allows for the selection of a directory in the file system.

Table 202: "Advanced"

"Activate property handling in all element properties"	☑: You can also configure a visualization element with a property in those of its properties in which you select an IEC variable. Then CODESYS creates additional code for the property handling when a visualization is compiled. Requirement: Its IEC code contains at least an object of type *"Interface property"* (a property).  Requirement: *"Visible"* is selected.
"Enable implicit checks for visualization POUs"	☑: The implicit check is also performed for visualization POUs. As a result, additional code is generated, which increases memory usage. When memory is limited, this option should be disabled.

See also

- *Object 'Property'*

Tab 'Symbol Libraries'

Table 203: "Visualization Symbol Libraries"

"Symbol libraries"	List of all installed symbol libraries (example: `VisuSymbols`)
"Assigned"	☑: Symbol library is selected in the project and CODESYS makes it available in the *"Visualization ToolBox"* view of a visualization. ☐: Symbol library is installed in the library repository, but CODESYS does **not** make it available in the *"Visualization ToolBox"* view of a visualization.

See also

- *CODESYS Visualization*
- *Dialog 'Add Visualization'*

Dialog 'Project Settings' - 'Visualization Profile'

Symbol:

Function: The dialog enables the setting of the visualization profile.

Call: Menu *“Project ➔ Project Settings”*, category *“Visualization Profile”*

Requirement: A project is open.

Table 204: “Visualization Profile”

“Certain profile”	Profile that CODESYS uses in the project and that determines the visualization elements that are available in the project. The selection list contains all the profiles installed so far.

Dialog 'Project Environment'

Function: You use this dialog for checking the actuality of the software and of the files, which are included in the project. CODESYS checks for example the selected compiler and finds out if there is a newer version. In such a case you can update the affected components.

Call: Main menu*“Project”*

Dialog 'Project Environment' – 'Library Versions'

Function: This dialog displays the libraries of the opened project for which newer versions are available.

Call: Main menu *“Project ➔ Project Environment”*, tab *“Library Version ”*

This dialog opens automatically when you open a project containing outdated libraries.

Table 205

The list shows the name of the outdated library with version, the currently available version and the planned action.	
“Action”	Double-click inside the field to select the desired actions.
“Check for updates when loading this project”	☑: Checking takes place each time the project is opened. ☐: Checking takes place once only.
“Set all to newest”	CODESYS uses the newest available version of the library.
“OK”	CODESYS performs the selected action(s).

Dialog 'Project Environment' - 'Compiler Version'

Function: This dialog shows the current compiler version of the project and provides the capability of updating.

Call: Main menu *“Project ➔ Project Environment”* (*“Compiler Version”* tab).

Table 206

"Current compiler version in project"	Shows the set compiler version for the open project.
"Recommended, newest version"	Shows the latest version.
"Action"	● *"Do not update"*: The compiler version of the project remains the same. ● *"Update to x.x.x.x"*: The selected compiler version is set for the project.
"Check for updates when loading this project"	☑: CODESYS checks for new versions each time the project is opened. If there is a new version, then the respective update dialog opens automatically. ☐: The compiler version is not checked. The update dialogs do not open automatically.
"Set all to newest"	The compiler version is set to the latest version.

Dialog 'Project Environment' - 'Device Versions'

Function: This dialog shows the devices of the open project in which there are new versions available.

Call: Main menu *"Project ➔ Project Environment"* (*"Device Versions"* tab)

This dialog opens automatically when you open a project that contains an outdated device.

Table 207

Names of the outdated devices and their versions, as well as the current version and the planned action.	
"Action"	Double-click in the field to select the required actions.
"Check for updates when loading this project"	☑: The check is performed when the project is opened. ☐: The check is performed one time only.
"Set all to newest"	CODESYS uses the latest library version.
"OK"	CODESYS executes the selected actions.

Dialog 'Project Environment' – 'Visualization Profile'

Function: This dialog shows the current visualization profile of the project. The profile can be updated here.

Call: Menu bar: *"Project ➔ Project Environment"* (*"Visualization Profile"* tab).

"Current visualization profile in the project"	The set visualization profile of the open project.
"Recommended, newest profile"	The newest version.
"Action"	● *"Do not update"*: The visualization profile of the project remains unchanged. ● *"Update to x.x.x.x"*: CODESYS updates the project to the selected visualization profile.
"Check for updates when loading this project"	☑: CODESYS checks for new profiles each time the project is opened. If there is a new version, then the respective update dialog opens automatically. ☐: Not test of the profile when opening the project. The update dialogs do not open automatically.
"Set all to newest"	CODESYS updates the .

See also

- Help about visualization

Dialog 'Project Environment' – 'Visualization Styles'

Function: This dialog shows the current visualization style of the project and provides the capability of updating it.

Call: Menu bar: *"Project ➔ Project Environment"* (*"Visualization Styles"* tab).

Table 208: "Newer versions are available for the following visualization styles currently in use"

"Visualization style"	Version of the set visualization style in the open project
"Current"	Current version of the visualization style (example: `3.5.6.0`)
"Recommended"	Recommended version of the visualization style (example: `3.5.7.0`)
"Action"	● *"Do not update"*: The visualization style of the project remains unchanged. ● *"Update to x.x.x.x"*: CODESYS updates the project to the version of the selected visualization style.
"Check for updates when loading this project"	☑: CODESYS checks for new versions each time the project is opened. If there is a new version, then the respective update dialog opens automatically. ☐: The version is not checked. The update dialogs do not open automatically.
"Set all to newest"	CODESYS updates the version.

See also

- Help for visualization, section "Visualization style"

Dialog 'Project Environment' – 'C Code Modules'

Function: This dialog lists all C-code modules and their C-code files that have changed in the source directory on the disk. You can update individual C-code modules here.

Call: Menu *"Project ➔ Project Environment"*, tab *"C Code modules"*

Table 209: "The sources of the following projects changed"

"Project"	Display of the C-code module with its changed C-code files in the project.
"Action"	Selection option for the C-code module of the *"Project"* field A double-click on the field displays all selection options: ● *"Update"* ● *"Do not update"* For each C-code file this indicates what action is executed if you select the action *"Update"* for the corresponding C-code module (*"Project"*).
"Delete IEC interfaces"	Deletes the created IEC interface if the headers in the project have changed. In this case you must create the IEC interface again.
"Check for updates when loading this project"	☑: checking takes place each time the project is opened.
"Set all to 'newest'"	CODESYS refreshes all C-code modules.
"OK"	CODESYS executes the selected actions in the project.

See also

- *Chapter 1.4.1.20.3.5.15 “Command 'Create IEC Interface'” on page 1026*
- *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*

Dialog 'Project Environment' – 'Visualization Symbols'

Function: The dialog lists installed symbol libraries and allows for you to assign symbol libraries to a project.

Call: Menu bar: *“Project ➔ Project Environment”*, *“Visualization Symbols”* tab

Requirement: The open project contains a visualization and has been saved with a compiler version < `3.5.7.0`. CODESYS recognizes symbol libraries in compiler version `3.5.7.0` and higher.

“Symbol library”	List of all installed symbol libraries
“Active”	☑: Symbol library is selected for the project. CODESYS provides its symbols in the *“Visualization Toolbox”* view. ☐: Symbol library has been previously installed only in the library repository.

See also

- Help for visualization, "Using the symbol library in the visualization" chapter

Dialog 'Options'

Function: You use the dialog box for selecting the CODESYS options. With these options you configure the appearance and the behavior of the user interface. CODESYS saves the current configuration as standard settings in the local system.

Call: Main menu *“Tools ➔ Options”*

Dialog 'Options' - 'Automation Builder'

Symbol:

Function: This dialog is for the configuration of the settings for the Automation Builder.

Call: menu *“Tools ➔ Options”*, category *“Automation Builder”*

Tab 'DeviceTree'

“Show warning message on delete objects”	☑: A warning appears whether the selected object should really be deleted from the project.
“Show device type”	☑: In the project tree the device type is displayed in brackets
“Show device tag”	☑: Show device tag
“Close Add / Update object dialog after single transaction”	☑: The "Add/Update Object" dialog is closed after a single transaction.
“Display all versions”	☐: Some devices are present in several versions. If the check mark is set, then all devices in all versions are displayed. If the checkbox is not set (default), then only the latest version is displayed.

Tab 'Project'

"Check integrity on open project"	☑: The integrity of an open project is automatically checked in the background. *Chapter 1.4.1.20.3.4.10 "Command 'Check integrity'" on page 1006*
"Check configuration on the fly directly on modify"	☑: The configuration can be checked directly when changing.
"Incremental update of configuration data"	☑: Performs an incremental update of configuration data.
"Activate legacy version of CSV signal export / import"	☐: If this checkbox is set, the old version of the CSV signal export/import is activated.

Tab 'Editors' Max parallel opened editors allowed 25 (max. 99).

"I/O-mapping"	
"Use tree based I/O mapping dialog"	☐: The view of the I/O mapping dialog is defined, here use tree based I/O mapping dialog.
"Use list based I/O mapping dialog"	☐: The view of the I/O mapping dialog is defined, here use list based I/O mapping dialog.
"Use both I/O mapping dialog"	☑: The view of the I/O mapping dialog is defined, here use both based I/O mapping dialog.

Tab 'General' ☐: *"Participate in ABB usability improvement program"* (Function not yet active.)

Dialog 'Options' - 'C Compiler'

Symbol:

Function: This dialog is for the configuration of the settings for the *"C Compiler"*.

Call: menu *"Tools* ➔ *Options"*, category *"C Compiler"*

Tab 'GCC 4.7.3'

"Path to Compiler executable"	Path to the file location.	
	"...": Opens the file manager to search for the file location.	
	"Reset": Resets the input.	
"Environment Variables"	*"New.."*: A new input window opens.	*"Variable name:"* Enter new variables.
		"Variable value:" Enter new variables.
	"Edit...": A new input window opens.	*"Variable name:"* Edit new variables.
		"Variable value:" Edit new variables.
	"Delete": Deletes the entries.	
	"Reset": Deletes the entries.	
"Include path"	*"New.."*: A new input window opens.	*"Path:"* Enter a new path.
	"Edit...": A new input window opens.	*"Path:"* Edit include path.
	C:\Program Files (x86)\ABB\AutomationBuilder\CCodeToolchain\FWAPI\2.11	

	“Delete”: Deletes the entries.
	“Reset”: Deletes the entries.

Tab 'GCC 4.7.3 PM595-4ETH'

“Path to Compiler executable”	Path to the file location.	
	“...”: Opens the file manager to search for the file location.	
	“Reset”: Resets the input.	
“Environment Variables”	*“New..”*: A new input window opens.	*“Variable name:”* Enter new variables.
		“Variable value:” Enter new variables.
	“Edit...”: A new input window opens.	*“Variable name:”* Edit new variables.
		“Variable value:” Edit new variables.
	“Delete”: Deletes the entries.	
	“Reset”: Deletes the entries.	
“Include path”	*“New..”*: A new input window opens.	*“Path:”* Enter a new path.
	“Edit...”: A new input window opens.	*“Path:”* Edit include path.
	C:\Program Files (x86)\ABB\AutomationBuilder\CCodeToolchain\FWAPI\2.11	
	“Delete”: Deletes the entries.	
	“Reset”: Deletes the entries.	

Tab 'GCC ++ 4.7.3 PM595-4ETH'

“Path to Compiler executable”	Path to the file location.	
	“...”: Opens the file manager to search for the file location.	
	“Reset”: Resets the input.	
“Environment Variables”	*“New..”*: A new input window opens.	*“Variable name:”* Enter new variables.
		“Variable value:” Enter new variables.
	“Edit...”: A new input window opens.	*“Variable name:”* Edit new variables.
		“Variable value:” Edit new variables.
	“Delete”: Deletes the entries.	
	“Reset”: Deletes the entries.	
“Include path”	*“New..”*: A new input window opens.	*“Path:”* Enter a new path.
	“Edit...”: A new input window opens.	*“Path:”* Edit include path.
	C:\Program Files (x86)\ABB\AutomationBuilder\CCodeToolchain\FWAPI\2.11	
	“Delete”: Deletes the entries.	
	“Reset”: Deletes the entries.	

Tab 'External diff tool'

"Path to external diff tool"	*"..."*: Opens the file manager to search for the file location.

Dialog 'Options' - 'CFC Editor'

Symbol: 

Function: This dialog is for the configuration of the settings for editing and printing in the CFC editor.

Call: menu *"Tools ➔ Options"*, category *"CFC Editor"*

Tab 'General'

"Enable AutoConnect"	☑: If you drag a CFC element onto the work area of the editor and insert it, CODESYS automatically connects together unconnected pins that 'touch' one another. Make sure that you do not create unwanted connections when shifting elements!
"Prepare values in implementation part"	☑: In online mode you can also prepare variable values for writing and forcing in the implementation part of the CFC module. In addition, CODESYS displays the values you have just prepared in the inline monitoring box of the variable in angle brackets.

Tab 'View'

"Display grid points"	☑: Grid points at which you can position the elements are visible in the editor.
"Show box icon"	☑: Existing function blocks that are linked with a bitmap are displayed by CODESYS in the CFC editor as symbols. Requirement: You have either created the link for a function block or a function in the object properties or loaded it via a library.
"Edit Line Colors"	Opens the *"Edit Line Colors"* dialog for the definition of the colors of the connecting lines, depending on the data type applied. The lines appear in offline and online mode in these colors, unless CODESYS paints over them with the thick black and blue lines used to display the boolean data flow. ● *"Add Type:"* Adds a data type to the list. ● *"Delete Type"*
"Font"	Display of the font and button for changing the font.

See also

- ↳ *Chapter 1.4.1.20.4.10.8 "Dialog Box 'Properties' - 'Bitmap'" on page 1162*

Tab 'Print'

Setting the *"Layout Options"*	
"Fit method"	*"Page"* or *"Poster"*
"Scale"	Possible values: 20 % - 200 %

See also

- ↳ *Chapter 1.4.1.8.3.2.2 " Programming in the CFC editor" on page 246*
- ↳ *Chapter 1.4.1.19.1 "Programming Languages and Editors" on page 460*

Dialog 'Options' – 'Declaration Editor'

Symbol:

Function: This dialog is for the configuration of the display settings for the declaration editor.

Call: Main menu *“Tools ➔ Options”*, category *“Declaration Editor”*

“Textual only”	Textual view of the declaration editor
“Tabular only”	Tabular view of the declaration editor
“Switchable between textual and tabular”	The declaration editor offers two buttons for switching between the textual and tabular views: : textual view : tabular view The following option defines the view that appears by default when opening a programming object: ● *“Always textual”* ● *“Always tabular”* ● *“Remember recent setting (per object)”* ● *“'Remember recent setting (global)”*

See also

- *Chapter 1.4.1.8.2.1 “Using the declaration editor” on page 226*

Dialog 'Options' – 'Device Description Download'

Symbol:

Function: This dialog is for the configuration of addresses of download servers for device descriptions.

Call: Menu *“Tools ➔ Options”*, category *“Download the Device Descriptions”*.

See also

- *Chapter 1.4.1.17 “Managing devices” on page 452*

Table 210: “Download server”

List of download servers containing device descriptions. By default `'https://store.codesys.com/CODESYSDevs'` is entered as the download server. If you select the button *“Download Missing Device Descriptions”* in the *“Device Repository”* dialog, CODESYS uses the servers entered here and uses the set login data for the proxy server.	
Double-click on *“(Enter new download server here...)”*	An input field opens in which you can enter the URL address of a server.
[Del]	Deletes the selected download server.

See also

- *Chapter 1.4.1.20.3.8.8 “Command 'Device Repository'” on page 1067*
- *Chapter 1.4.1.20.4.13.20 “Dialog 'Options' - 'Proxy Settings'” on page 1198*

Dialog 'Options' - 'Device Editor'

Symbol:

Function: This dialog includes settings for displaying the device editor.

Call: Menu bar: *“Tools ➔ Options”*; category: *“Device Editor”*.

Tab 'View'

“Show generic device configuration views”	☑: This tab with the list of device parameters is available in the device editors of parameterizable devices.
“Create cross references for IEC addresses (clean necessary) ”	☑: CODESYS creates the cross-references for unmapped I/Os.
“Communication page”	• *“Classic mode”*: The *“Communication”* tab of the device editors appears as a split window with the left side showing the current configured gateway channels in a tree structure and the right side showing the associated data and information. • *“Simple mode”*: The *“Communication”* tab appears as described in the corresponding section in the help. Additional modes may also be available from customer-specific extensions.
“Show implicit files for application download on the editor of a PLC”	☑: The tab for synchronized files is available in the device editors. Synchronized files are downloaded to the PLC at the time of application download. These can be external files that were added to the application, or implicit files such as a source code archive.
“Show access rights page”	☑: The *“Access Rights”* tab is available in the device editors. Note: Depending on the device, the device description may overwrite this setting.

See also

- *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*
- *Chapter 1.4.1.20.2.8.6 “Tab 'Synchronized Files'” on page 847*

Dialog 'Options' - 'Diagnosis'

Symbol: 

Function: This dialog is for the *“Diagnosis”* setting and views.

Call: menu *“Tools ➔ Options”*, category *“Diagnosis”*

Table 211: 'Diagnosis view'

“Enable subtree diagnosis”	☑: The subtree diagnosis is switched on.
“Enable debug columns”	☐: Debug columns are enabled.

Dialog 'Options' - 'External tools'

Symbol: 

Function: This dialog is for setting of *“External tools”*.

Call: menu *“Tools ➔ Options”*, category *“External tools”*

Tool	Version
“Panel builder”	*“Default”*

	“Custom”: Opens the file manager to search for the file location. Sometimes it is required to used dedicated versions of these tools (qualified versions, versions supporting more legacy types, …)
“Drive composer pro”	*“Default”*
	“Custom”: Opens the file manager to search for the file location. Sometimes it is required to used dedicated versions of these tools (qualified versions, versions supporting more legacy types, …)

“Restore defaults”	Resets the custom settings to default.

The modified settings will be valid after restart of Automation Builder.

Dialog 'Options' - 'FBD, LD, and IL'

Symbol: 

Function: This dialog is used for configuring the display options for the FBD/LD/IL editor.

Call: *“Tools ➔ Options”* (category *“FBD, LD, and IL”*.

Tab 'General'

Table 212: “View”

“Show network title”	The network title is displayed in the upper left corner of the network.
“Show network comment”	The network comment is displayed in the upper left corner of the network. When the network title is also shown in CODESYS, the comment is shown in the line below.
“Show box icon”	The block symbol is displayed in the block element in the FBD and LD editor. The standard operators also have symbols.
“Show operand comment”	CODESYS shows the comment that you indicated for a variable in the implementation part. The operand comment refers to the local occurrence of the variable only, as opposed to the symbol comment. This comment is truncated automatically depending on available space. You can limit the comment to a defined width by activating the option *“Fixed size for operand fields”*.
“Show symbol comment”	The comment that you indicated for a variable or symbol in the declaration is displayed in CODESYS above the variable name. You can also assign a local operand comment in addition to or instead of the symbol comment.
“Show symbol address”	If an address is assigned to a symbol (variable), then this address is displayed above the variable name.
“Show network separators”	A separator is displayed between the individual networks.

Table 213: “Behavior”

“Placeholder for new operands”	The operand field of pins for the new function block is left blank (instead of "???").
“Empty operands for function block pins”	Adds blank operands instead of `???`.

Table 214: “Font”

Click the input field to open the *“Font”* dialog.

“Fixed size for operand fields”	☑: *“Edit operand sizes”* can be enabled.
“Edit operand sizes”	The *“Operand Sizes”* dialog opens for setting the number of characters and lines.

Tab 'FBD'

Table 215: “View”

“Networks with line breaks”	☑: Display of the network with line breaks so that so that CODESYS can show as many blocks as possible in the current width of the window.
“Connect boxes with straight line”	☑: The length of the lines between the elements are fixed and short.

Table 216: “Behavior”

“Default network content”	Drop-down list: Contents of a new network
“After insertion select”	Drop-down list: Element that CODESYS selects after inserting a new network

Tab 'LD'

Table 217: “View”

“Networks with line breaks”	☑: Display of the network with line breaks so that so that CODESYS can show as many blocks as possible in the current width of the window.

Table 218: “Behavior”

“Default network content”	Drop-down list: Contents of a new network
“After insertion select”	Drop-down list: Element that CODESYS selects after inserting a new network

Tab 'IL'

Table 219: “View”

“Enable IL”	The IL implementation language is available in the development system.

Table 220: “Behavior”

“Default network content”	Drop-down list: Contents of a new network
“After insertion select”	Drop-down list: Element that CODESYS selects after inserting a new network

Tab 'Print'

Table 221: “Layout Options”

“Fit method”	Drop-down list for resizing.
“Avoid cutting of elements”	Elements that do not fit on the page are printed on the next page.
“Mark connections on adjacent pages”	Enabled for selection when *“Avoid cutting of elements”* is selected.

See also

- *Chapter 1.4.1.19.1 “Programming Languages and Editors” on page 460*

Dialog 'Options' - 'Help'

Symbol:

Function: This dialog defines whether CODESYS Online Help or CODESYS Offline Help opens when help is called.

Call: Menu bar: *“Tools ➔ Options”*; category: *“Help”*.

“Use CODESYS Online Help, if available”	• ☑ CODESYS Online Help opens when CODESYS Help is called. This is the default setting. • ☐ CODESYS Offline Help opens when CODESYS Help is called.

See also

- *“Using CODESYS help” on page 176*

Dialog 'Options' - 'Help'

Symbol:

Function: This dialog activates the online help if available.

Call: menu *“Tools ➔ Options”*, category *“Help”*

☐: *“Use Online Help if available”*.

Dialog 'Options' - 'IEC 60870-5-104'

Symbol:

Function: In this dialog you can set this notation of the *“Address format”*.

Call: menu *“Tools ➔ Options”*, category *“IEC 60870-5-104”*

Tab 'Adress format'

☑ 1.2.3 (separated by dots)
☐ 1-2-3 (separated by hyphens)
☐ 66051 (decimal number, big endian --> 0x10203)
☐ 197121 (decimal number, little endian --> 0x30201)

Dialog 'Options' – 'International Settings'

Symbol:

Function: This dialog is for the setting of the language in the user interface and in the help.

Call: Menu bar: *“Tools ➔ Options”*, category *“International Settings”*.

Dialog 'Options' – 'Libraries'

Symbol:

Function: This dialog helps you to manage the mappings of library references that CODESYS uses during the conversion of an old project. If you have not yet stored any mapping for a certain library, you must redefine the mapping each time when opening an old project in which this library is integrated.

Call: Menu bar: *“Tools ➔ Options”*; category: *“Libraries”*.

A mapping defines what a library reference looks like following the conversion of the project to the current format. There are three possibilities:

- You retain the reference. This means that CODESYS similarly converts the library into the current format (*.library) and installs it in the local library repository.
- You replace a reference with another reference. This means that one of the installed libraries replaces the library that was integrated until now.
- You delete the reference. This means that the converted project no longer integrates the library.

CODESYS applies all the listed mappings to the library references of an old project the next time it is converted. Hence, you must repeat the mapping definition if the same library is integrated again in a project that is to be converted. You can enter a new mapping in the last line.	
“Source Library”	Path of the library that is integrated in the project before the conversion. A double-click an entry makes the field editable and the button for the input assistance appears.
“Target Library”	Name and location of the library that is to be integrated in the project after the conversion. A double-click an entry opens the dialog *“Set target system library”*.

Table 222: “Set target system library”

“Scan”	The *“Select Library”* dialog opens. You can select a library from the library repository here. The dialog corresponds to the dialog in the library repository.
“Ignore”	When CODESYS converts the project, CODESYS always removes the existing source library from the project.

Dialog 'Options' – 'Library Download'

Symbol

Function: This dialog is for the setting of download servers.

Call: menu *“Tools ➔ Options”*, *“Library Download”* category

If you click on the button *“Download Missing Libraries”* in the library manager, CODESYS browses these download servers for libraries marked as missing in the library manager and uses the set login details for the proxy server.	
“Download servers”	URL of a server containing library files
Double-click on *“(Enter new download server here)”*	An input field opens in which you can enter the URL address of a server.

See also

- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*
- *Chapter 1.4.1.20.4.13.20 “Dialog 'Options' - 'Proxy Settings'” on page 1198*

Dialog 'Options' – 'Load and Save'

Symbol: 

Function: The dialog contains settings for the behavior of CODESYS when loading and saving a project.

Call: Menu bar: *“Tools* ➔ *Options”*, *“Load and Save”* category

“Create backup files”	☑: Each time the project is saved, CODESYS also saves the project as the file `<project name>.project` in addition to the file `<project name>.backup`. You can rename the backup file and open it in the programming system.
“Automatically save every … minutes”	☑: CODESYS automatically saves the project at the specified time intervals in a file `<project name>.autosave`, which you can reload following non-regular closing of the programming system. CODESYS deletes the `.autosave` file whenever the project is closed or saved regularly. CODESYS retains the `.autosave` file in the case of an irregular termination. When you open a project for which there is an associated autosave file, the *“Auto Save Backup”* dialog opens. In this dialog you select whether the `.autosave` file or the version of the project last saved by the user should be opened.
“Save before build”	CODESYS saves the project automatically before each build operation.
“Create project recovery information”	Requirement: The *“No protection”* option is selected in the project settings in the *“Security”* category. This means that the project is not protected against unauthorized access and data manipulation, and there is no integrity check when the project is loaded. ☑: If a project crashes during editing, then the next time the project is opened, a prompt is displayed asking whether or not you want to restore the unsaved data and create a new project file. If you click *“Yes”*, then another dialog opens. In this dialog, you can select whether you want to open the restored project or open the project comparison. This project comparison displays the differences between the last saved project and the restored project. Note: The project restore records every change on the hard disk when the change is made. If a power failure or hard disk error occurs on the hard disk during this operation, then the last change may be lost.
“Advanced Settings”	The *“Advanced Settings”* dialog opens.

"At startup"	List box for the startup screen of CODESYS: ● *"Show start page"*: The start page of CODESYS is shown. ● *"Load last loaded project"* ● *"Show "Open Project" dialog"* ● *"Show "New Project" dialog"* ● *"Show empty environment"*
"News page"	URL that is opened by means of the command *"Help ➔ CODESYS CODESYS Homepage"*. By default, this page is `http://www.codesys.com/startpage`.

Table 223: Dialog "Advanced Settings"

"Project compression"	
"Level"	Requirement: The *"No protection"* option is selected in the project settings in the *"Security"* category. This means that the project is not protected against unauthorized access and data manipulation, and there is no integrity check when the project is loaded. List box for the compression level that is used when saving the project. ● *"Least compression - best speed (recommended)"* ● *"Medium compression - medium speed"* ● *"Most compression - worst speed"*
"Load Behavior"	
☑	Libraries and compilation information are loaded in the background while you edit the project.

See also

- *Chapter 1.4.1.5 "Protecting and Saving Projects" on page 197*

Dialog 'Options' - 'Message View'

Symbol:

Function: In this dialog the number of messages can be determined.

Call: menu *"Tools ➔ Options"*, category *"Message View"*

"Maximum numer of messages" xxx

Default, 500, max. 9999, min. 20

Dialog 'Options' - 'Monitoring'

Symbol:

Function: This dialog includes settings for displaying the variable values in monitoring.

Call: Menu bar: *"Tools ➔ Options"*; category: *"Monitoring"*.

Table 224: “Display Mode for Integer Variables”

“Binary”	The value of the variable is displayed in the corresponding format in online mode. This option corresponds to the setting of the command *“Debug ➔ Display Mode”*.
“Decimal”	
“Hexadecimal”	

Table 225: “Floating Point Variables”

“Number of displayed digits”	Decimal places that are represented in online mode when REAL values are displayed. Note: The settings apply to the watch list, the monitoring of the declaration editor and the trace editor. The configuration for inline monitoring of the editor is set in the text editor options.

See also

- *“Tab 'Monitoring'” on page 1205*

Dialog 'Options' - 'PLCopenXML'

Symbol: 

Function:This dialog contains settings for the behavior of CODESYS when exporting or importing PLCopenXM.

Call: Main menu *“Tools ➔ Options”*, category *“PLCopenXML”*

Table 226: “PLCopenXML Export Settings”

“Additionally export declarations as plain text”	By default, CODESYS splits the declaration parts in accordance with the PLCopenXML scheme into individual variables and thus loses the formatting and some comment information. ☑: Formatting and comments are retained. CODESYS additionally writes the plain text of the exported declaration part into the PLCopenXML file and thus extends the PLCopenXML scheme.
“Export Folder Structure”	☑: CODESYS also exports the folders if they contain one of the selected objects. That is a CODESYS-specific extension to the PLCopenXML scheme.

Table 227: “PLCopenXML Import Settings”

“Import folder structure”	☑: If the import file contains information about the folder structure of the objects, CODESYS also imports this structure. ☐: CODESYS imports objects without structure.

See also

- *Chapter 1.4.1.3.1 “Exporting and importing projects” on page 193*
- *Chapter 1.4.1.20.3.4.26 “Command 'Export PLCopenXML'” on page 1015*
- *Chapter 1.4.1.20.3.4.27 “Command 'Import PLCopenXML'” on page 1015*

Dialog 'Options' - 'Proxy Settings'

Symbol: 

Function: You use this dialog for storing the authentication data for the proxy server which is currently used for accessing the internet from CODESYS.

Call: Main menu *“Tools ➔ Options”*, category *“Proxy Settings”*

Requirement: Internet access of the network via proxy server

“Enter proxy credentials”	A double click opens the input request for the user name and the password for the proxy server. CODESYS uses the access data for the establishment of the connection to the download server for libraries and the device description, for the establishment of the connection to the CODESYS Store and for the command *“View ➔ Start Page”*. Requirement: If the internet access of your computer or of the network takes place via a proxy server, then the button is available.

- *Chapter 1.4.1.20.2.14 “Object 'Library Manager'” on page 874*
- *Chapter 1.4.1.20.3.3.20 “Command 'Start Page'” on page 999*

Dialog 'Options' - 'Refactoring'

Symbol: 

Function: The dialog is used for defining the operations in the project for which the automatic refactoring is suggested. The refactoring functionality helps you in your improvement endeavors.

Call: Menu bar: *“Tools ➔ Options”*, *“Refactoring”* category

'Suggest Refactoring for the Following Operations'

“Auto-declare”	When you change the name of a variable in a declaration by calling AutoDeclare (*[Shift]+[F2]*), the activated option *“Apply changes by means of refactoring”* appears. Then the *“Refactoring”* dialog opens and you can change the variable throughout the project. ● *“On adding or removing variables, or on changing the scope”* ☑: You delete the names in the *“Declare Variable”* dialog and click *“OK”* to close the dialog. Then the *“Refactoring”* dialog opens for removing the variable throughout the project. ● *“On renaming variables”* ☑: You specify the names in the *“Declare Variable”* dialog and click *“OK”* to close the dialog. Then the *“Refactoring”* dialog opens for renaming the variable throughout the project. See the chapter: "Refactoring", "Changing a variable declaration and applying refactoring automatically".
“Unit conversion editor”	*“ On renaming of unit conversions”* ● ☑: When you change the name of a conversion in the unit conversion editor, you are prompted whether CODESYS should perform "Automatic Refactoring" when renaming.
“Mapping editor”	*“On renaming variables”* ● ☑: When you change a variable name in the device editor (*“I/O Mapping”* tab), you are prompted whether CODESYS should perform "Automatic Refactoring" when renaming.

“Navigator”	*“On renaming objects”* ● ☑: When you change the name of an object in the device tree or in the POUs view, you are prompted whether CODESYS should perform "Automatic Refactoring" when renaming.
“Tabular declaration editor”	*“On renaming variables”* ● ☑: When you change the name of a variable in the tabular declaration editor, you are prompted whether CODESYS should perform "Automatic Refactoring" when renaming.

See also

- *Chapter 1.4.1.8.15 “Refactoring” on page 289*
- *Chapter 1.4.1.8 “Programming of Applications” on page 222*
- *Chapter 1.4.1.20.3.2.40 “Command 'Refactoring' - 'Rename <...>'” on page 980*
- *Chapter 1.4.1.20.3.2.32 “Command 'Auto Declare'” on page 975*
- *Chapter 1.4.1.20.2.33 “Object 'Unit Conversion'” on page 952*

Dialog 'Options' - 'SFC Editor'

Symbol: 

Function: This dialog is used for configuring the settings for the SFC editor.

Call: Menu bar: *“Tools ➔ Options”* (*“SFC Editor”* category).

See also

- *Chapter 1.4.1.8.3.4.1 “Programming in SFC” on page 255*
- *Chapter 1.4.1.20.3.11 “Menu 'SFC'” on page 1079*
- *Chapter 1.4.1.19.1.4.1 “SFC editor” on page 476*

Tab 'Layout'

Table 228: “Elements”

This defines the dimensions of the SFC elements: step, action, qualifier, property. The values are given in matrix units, where one matrix unit equals the font size that you set in the text editor options (text area / font). The settings are always active immediately in all open SFC editor views.	
“Step height”	Possible values: 1-100
“Step width”	Possible values: 2-100
“Action width”	Possible values: 2-100
“Qualifier width”	Possible values: 2-100
“Property width”	Possible values: 2-100

Table 229: “Font”

The example text shows the current font. Click it to change the font.

Table 230: “Step Actions”

“Default insertion method”	• *“Copy reference”*: The reference to the action objects that call the step are also copied when the step is copied. The copied step and new step call the same action. • *“Duplicate implementation”*: The reference to the action objects that call the step are linked to this step. When copying the step element, new action objects are created for the new step, and the implementation is duplicated. • *“Always ask”*: When inserting a step action, you are always prompted whether the actions of a step element should be duplicated when it is copied, or whether the reference to the existing action should be applied. Note: If a step already contains an embedded action, then new inserted actions of this step are also embedded. Likewise, new inserted actions are not embedded when the step already contains a non-embedded action. In these cases, you are no longer prompted for a duplication mode.

Table 231: “Embedded Objects”

“Show action and transition objects in the navigator”	☑: Action and transition objects that are embedded in the SFC box by a step are displayed in the *“Devices”* or *“POUs”* tree view.

Tab 'View'

Table 232: “Property Visibility”

List of element properties for the categories *“Common”* and *“Specific”* with definitions of the display options.	
“Property”	Defines the element properties displayed next to the element in the SFC diagram.
“Value”	☑: Display of the property value.
“With Name”	☑: Display of the property value including name.

Table 233: “Online”

“Show step time”	☑: In online mode, CODESYS displays the step time to the right of the steps.

Dialog 'Options' - 'SmartCoding'

Symbol: 

Function: This dialog is for configuring the settings for easier coding.

Call: Menu bar: *“Tools* ➔ *Options”*, *“SmartCoding”* category

“Declare unknown variables automatically (AutoDeclare)”	☑: The *“Declare Variable”* dialog opens when you type an undeclared identifier into an implementation language editor and then click away from the input line. In order for the AutoDeclare function to be available in the ST editor as well, the *“Enable for ST editor”* option also has to be selected.
“Enable for ST editor”	Requirement: The *“Declare unknown variables automatically (AutoDeclare)”* option is selected. ☑: The AutoDeclare function is also available in the ST editor. ☐: The AutoDeclare function is not available in the ST editor.

“Show all instance variables in Input Assistant”	☑: The "List components" function also lets you select the local variables of a function block instance. ☐: The "List components" function lets select only the input variables and output variables of a function block instance.
“Show symbols from system libraries in Input Assistant”	System libraries are inserted in the library manager automatically and displayed in light gray. ☑: Symbols, such as global variables, data types, and function blocks, are offered in the Input Assistant. ☐: The symbols of the system libraries are not available in the Input Assistant.
“List components after typing a dot (.)”	☑: Activates the "List components" function. When you type a dot (.) at a location where CODESYS expects an identifier, a list box appears with possible code.
“List components immediately when typing”	Requirement: The *“List components after typing a dot (.)”* check box is selected. ☑: While you type code, a list box appears with possible identifiers and operators.
“Insert with namespace”	☑: CODESYS adds the namespace before the identifier.
“Convert keywords to uppercase automatically (AutoFormat)”	☑: CODESYS displays all keywords in uppercase.
“Automatically list selection in cross-reference view”	☑: The cross-reference list automatically shows the references of variables, POUs, and DUTs that are currently selected or where the cursor is waiting.
“Underline errors in the editor”	☑: Incorrect or unknown program code is underlined.
“Highlight symbols”	☑: All occurrences of a symbol where the cursor is positioned are highlighted in color within the editor. In this way, cross-references within the editor are quickly detected.
“Max. degree of parallelism”	List box for the number of parallel threads that can be used for the precompile processing. CODESYS detects the displayed number of threads from the number of CPU cores. This default number should be changed only in exceptional cases.

See also

- *Chapter 1.4.1.19.1 “Programming Languages and Editors” on page 460*
- *“Smart tag functions” on page 263*
- *Chapter 1.4.1.8.13.1 “Using the cross-reference list to find occurrences” on page 285*
- *Chapter 1.4.1.19.1.3.1 “ST Editor” on page 463*

Dialog 'Options' - 'Startup settings'

Symbol:

Function: In this dialog the *“Version profile”* and the *“License”* are set.

Call: menu *“Tools* ➔ *Options”*, category *“Startup settings”*

“Version profil:”	Automation Builder 2.5
	☐ *“Display selected dialog at each start”*: Refers to the version of the AB to be displayed

“License:”	☑ *“Default: take any available license”*
	☐ *“Use only local license”*
	☐ *“Display license selection dialog if shared licenses are available”*

The modified settings will be valid after restart of the Automation Builder.

Dialog 'Options' - 'Text Editor'

Symbol: 

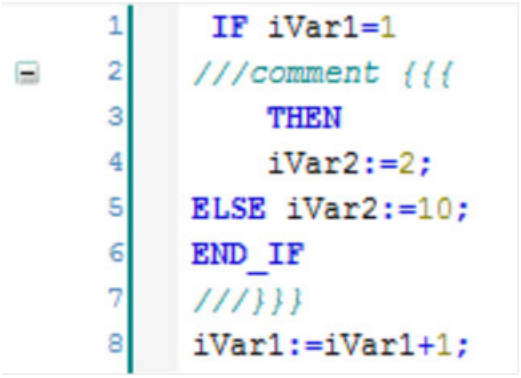
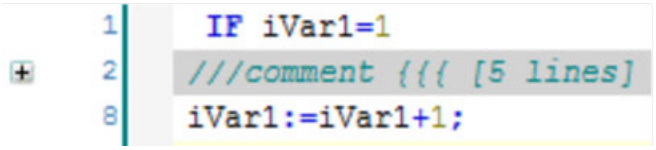
Function: The dialog contains settings for displaying and working in a text editor.

Call: Menu bar: *“Tools* ➔ *Options”*, *“Text Editor”* category

Tab 'Theme' On this tab, you set the desired theme in the interface design of the ST editor.

“Theme”	Color theme for the text editor. The selected theme is shown in the *“Preview”* window. The available color schemes are stored in the installation directory in the `Themes` folder.

Tab 'Editing'

"Number of undos"	Maximum number of editing steps that you can apply the *"Edit ➔ Undo"* command to.
"Folding"	Defines the structuring of the code by indentation. When you select an indentation, you can expand or collapse the indentation section by means of a plus and minus sign in front of the first line of each section. ● *"Indent"*: CODESYS combines all lines that are indented in relation to the preceding line into one indentation unit. ● *"Explicit"*: You mark the code segment explicitly with comments that should be combined in one indentation unit: a comment with three opening braces "{{{" has to be before the segment, and a comment with three closing braces "}}}" has to be after the segment. The comments can contain additional text. Example:  
"Word wrap"	● *"Soft"*: The line break occurs at the edge of the editor window when 0 is specified for *"Wrap margin"*. ● *"Hard"*: The line break occurs after the number of characters specified for *"Wrap margin"*.
" Tab width"	Number of characters
"Keep tabs"	☑: CODESYS does not break up the space you have inserted with the *[Tab]* key into individual spaces afterwards.
"Indent width"	If you have selected *"Smart"* or *"Smart with code completion"* for the *"AutoIndent"* option, then CODESYS inserts the number of spaces at the beginning of the line.
"AutoIndent"	● *"None"* ● *"Block"*: A new line automatically applies the indentation of the previous line. ● *"Smart"*: Lines that follow a line which contains a keyword (for example, `VAR`) indent automatically by the specified `Indent width`. ● *"Smart with code completion"*: Indentation as in the case of the *"Intelligent "* option, but CODESYS also inserts the closing keyword (for example, `END_VAR`).

Tab 'Text Area'

"Highlight current line"	☑: The line where the cursor is located is highlighted.
"Matching brackets"	☑: When the cursor is positioned before or after a bracket within a line of code, the corresponding closing or opening bracket is marked by a frame.

"End of line markers"	☑: The end of each editor line is marked by a small dash after the last character (including spaces) of the line.
"Wrap guide"	☑: When a soft or hard line break is activated, the defined line break position is displayed with a vertical line.
"Font"	Clicking the field opens the default dialog for configuring the font.

Tab 'Margin'

Settings for the left margin of the text editor window, which is separated from the input area by a vertical line:	
"Line numbering"	☑: The declaration and implementation parts of the editor are numbered on the left, each beginning with 1.
"Highlight current line"	☑: The line number of the line where the cursor is located is highlighted.
"Show bracket scope"	☑: Brackets include the lines between the keywords that open and close a construct (for example, `IF` and `END_IF`). When the option is enabled and the cursor is positioned before, after, or in one of the keywords of a construct, the bracket area is displayed with a square bracket in the margin.
"Mouse Actions"	You can assign one of the following actions to each of the specified mouse actions or mouse-keyboard combinations. CODESYS performs the selected action when you move the mouse to the plus or minus sign in front of the header of a bracketed area: ● *"None"*: The mouse action does not trigger an action. ● *"Select fold"*: CODESYS selects all lines of the bracketed area. ● *"Toggle fold"*: CODESYS opens or closes the bracketed area, or if there are nested brackets, the first level of the bracketed area. ● *"Toggle fold fully"*: CODESYS opens or closes all levels of a nested bracketed area.

Tab 'Monitoring'

Settings for displaying the monitoring fields	
"Enable inline monitoring"	☑: Display of the monitoring fields behind the variables in online mode
"Number of displayed digits"	Number of comma places in the monitoring field
"String length"	Maximum length of string variable values in the monitoring field

See also

- *Chapter 1.4.1.8.3.3.1 "Programming structured text (ST)" on page 254*

Dialog 'Customize'

The dialog contains the tabs to configure the user interface.

You can reset the CODESYS settings to default by use of the *"Reset"* button.

Dialog 'Customize' - 'Menu'

Function: With this dialog, you define the structure and contents of the user interface.

Call: Main menu *"Tools ➔ Customize"* (*"Menu"*).

When you click *"OK"* to close the dialog, the changes are visible in the menu bar of the CODESYS user interface.

Table 234: "Menu"

Display of currently defined menus, submenus, and included commands. In CODESYS, a menu or submenu caption is identified by the caption symbol ( ). The layout from top to bottom corresponds to the layout displayed later in the CODESYS menu.	
"Add Command"	Enabled when a command is selected. Adds a command above the selected command. Opens the *"Add Command"* dialog. Use the *"Add Command"* dialog for selecting one or more commands. Left part: List of categories. Right part: List of commands in the selected category.
"Add Separator"	Adds a separator above the selected command.
"Add Popup Menu"	Adds a popup menu above the selected menu, submenu, or command. Opens the *"Add Popup Menu"* dialog.
"Edit Popup Menu"	Opens the *"Edit Popup Menu"* dialog.
"Reset"	Resets the default settings of the entire menu.
"Load"	Loads the settings from a stored file (<file name>.opt.menu).

Table 235: "Add Popup Menu"

In CODESYS, a new menu is shown in the menu bar only when the menu contains at least one command.	
"Default text"	Select this check box when localization is available.
"Localized Texts"	List: Languages and localized texts.
"Add Language"	Opens a drop-down list of available languages. In CODESYS, the selected language is displayed in the area *"Localized Texts"*. Use the *"Text"* column for typing the localized texts.

See also

- *Chapter 1.4.1.1.2.1 "Customizing menus" on page 180*
- *Chapter 1.4.1.20.4.14.3 "Dialog 'Customize' - 'Toolbars'" on page 1207*

Dialog 'Customize' - 'Command Icons'

Function: This dialog defines the icons of the menu commands.

Call: Menu bar: *"Tools ➔ Customize"* (*"Command Icons"*).

Table 236: "Command icon"

"Assign"	Opens a dialog for selecting the new icon (*.ico).
"Remove"	Removes the user-defined icon. The default icon is active again.
"Reset"	Resets all default settings of the command icons.
"Load"	Loads the settings from a stored file (<file name>.opt.keyb).
"Save"	Saves the current settings to a file (<file name>.opt.keyb).

See also

- *Chapter 1.4.1.1.2.3 “Customize command icon” on page 183*

Dialog 'Customize' - 'Toolbars'

Function: Use this dialog for generating new toolbars or customizing existing toolbars.

Call: Main menu *“Tools ➔ Customize”* (*“Toolbars”*).

When you click *“OK”* to close the dialog, the changes are visible in the menu bar of the CODESYS user interface.

Table 237: “Toolbars”

Display of currently defined toolbars. In CODESYS, the associated commands are listed below each toolbar in the order they will appear in the toolbar. Double-clicking a toolbar in the list switches to editing mode.	
“Add Toolbar”	Enabled when a toolbar is selected. In CODESYS, this adds a toolbar above the selected toolbar and places the cursor in the name field of the new toolbar.
“Add Command”	Enabled when you select a command or blank command entry below a toolbar. Adds a command above the selected command. Opens the *“Add Command”* dialog. Use the *“Add Command”* dialog to select one or more commands. Left part: List of categories. Right part: List of commands in the selected category.
“Add Separator”	Adds a separator above the selected command.
“Hide”	Hide the selected toolbar from the user interface.
“Show”	Shows the selected hidden toolbar in the CODESYS user interface.
“Reset”	Resets the default settings of the toolbars.
“Load”	Loads the settings from a stored file (<file name>.opt.tbar).

See also

- *Chapter 1.4.1.1.2.2 “Customizing toolbars” on page 182*
- *Chapter 1.4.1.20.4.14.1 “Dialog 'Customize' - 'Menu'” on page 1206*

Dialog Box 'Customize' - 'Keyboard'

Function: This dialog box is used for defining keyboard shortcuts (quick access keys or keyboard combinations) for commands.

Call: Main menu *“Tools ➔ Customize”* (*“Keyboard”*).

Table 238: “Keyboard”

“Shortcuts for selected command”	Keyboard shortcuts for the selected command The drop-down list can include more than one keyboard shortcut for the command.
“Press shortcut keys”	Input field for the keyboard shortcut of the selected field. Permitted combinations include *[Ctrl]*, *[Alt]*, *[Shift]*, and other keys. You clicking *“Assign”* to assign a recorded keyboard shortcut to a selected command.
“Shortcut keys currently used by”	Command assigned to the currently defined keyboard shortcut

“Reset”	Resets the default settings of the keyboard shortcuts.
“Load”	Loads the settings from a stored file (<file name>.opt.keyb).

Dialog 'Trace Configuration'

Dialog 'Advanced Trace Settings'

Function: This dialog provides extended settings for recording data.

Call: *“Advanced”* button in *“Trace Configuration ”* dialog, *“Record Settings”* subdialog

Requirement: The trace editor is open and active. The dialog *“Trace configuration”* is open and the top node of the trace record tree is selected so that the subdialog *“Record settings”* is available.

For the calculation of the values, you have to select a task in the “Trace Configuration” dialog.

The buffer size is defined as "number of samples". CODESYS calculates the time intervals that corresponds to this number and displays the result in normal fonts on the right outside the table (for example, *“1h1m1s1ms”*). The calculation is possible only with the help of the task configuration settings and when the task cycle time is known.		
“Measurement in every nth cycle”	Data recording in every n task cycle Preset: `1`; then the application performs the data recording in each task cycle.	Scanning interval of the data recording Example: `100ms`
“Recommended runtime buffer size (samples)”	Requirement: *“Override runtime buffer size”* is deactivated. The maximum number of samples that CODESYS calculates and recommends, which the application stores at runtime per trace variable. CODESYS calculates the number in the task cycle time from the value in *“Measure in every n-th cycle”* and the value in Measure in every n-th cycle.	Maximum length of the time interval during which the application collects data on the runtime system. Example: `2s`
“Override runtime buffer size”	Maximum number of samples per trace variable that saves the application per trace variable in runtime mode. Example: `100` Value range: starting at `10` ☑: The application uses this value, not the value calculated by CODESYS from *“Recommended runtime buffer size (samples)”*.	Maximum length of the time interval during which the application collects data on the runtime system. Example: `6s`
“Trace editor buffer size per variable (samples)”	Number of values that can be stored per variable in the trace editor. Example: `10000`	The maximum time period for the display in the trace editor results from the maximum number and the scanning interval of the data recording. You can scroll back a maximum of this time in the trace editor.

See also

- *Chapter 1.4.1.20.2.28 “Object 'Trace'” on page 945*
- *Chapter 1.4.1.12.3.2 “Creating trace configuration” on page 424*

Dialog 'Trace Configuration'

Symbol:

Function: The dialog includes the trace configuration for the data recording.

Call

- *“Trace ➔ Configuration”*; context menu
- Link *“Configuration”* in the trace editor
- Link *“Add Variable”* in the trace editor

Requirement: The editor of a trace object is open and active.

See also

- *Chapter 1.4.1.12.3.2 “Creating trace configuration” on page 424*
- *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*
- *Chapter 1.4.1.19.6.2.25 “Attribute 'monitoring'” on page 709*

Tree view 'Trace Record' The tree view lists the variables that are traced and allows for access to the variable settings.

Selected trace name	The *“Record Settings”* subdialog is displayed on the right.
Selected trace variable	The *“Variable Settings”* subdialog is displayed on the right.

Table 239: Context menu commands

+ *“Add Variable”*	Adds a new trace variable. The *“Variable Settings”* subdialog opens on the right and it is partially configured. Select a variable in the input field of the *“Variable”* setting to trace its value curve.
“Assign to Diagram”	Lists the diagrams (in the submenu on the right) where the selected variable is not currently displayed. Select a diagram to display the variable there. The command is available when a variable is selected in the tree view. Hint: When the command is deactivated, the variable is already displayed in all diagrams.
“Enabled”	Selected by default Disabled variables are displayed as disabled. They are neither displayed nor recorded.

Tree view 'Presentation (Diagrams)' The tree view lists the diagrams that are displayed in the trace editor and allows for access to their display mode.

Selected node *“Time axis”*	The *“Display Mode”* subdialog for the time axis is displayed on the right. You can specify the time axis display. See below.
Selected diagram name	The settings for the coordinate system of the diagram and a preview are displayed on the right. See below.
Selected node *“Y-axis”*	The *“Display Mode”* subdialog is displayed on the right. You can specify the axis display. See below.

“Show variables”	
Selected trace variable	The *“Variable Settings”* subdialog is displayed on the right. You can configure the trace variable. See below. Note: These are the same settings that can be accessed in the *“Trace Record”* tree view.

Table 240: Context menu commands

+ *“Add Diagram”*	Adds a new diagram below and displays it in the tree view *“Presentation (Diagrams)”*.
+ *“Add New Variable”*	Adds a new trace variable. The *“Variable Settings”* subdialog opens on the right and it is partially configured. Select a variable in the input field of the *“Variable”* setting to trace its value curve. Specify its display. In addition, the variable is assigned to the selected diagram.
“Add Existing Variable”	Lists all trace variables (in the submenu on the right) where the selected diagram is not currently displayed. Select a variable in order to display it in the selected diagram. Hint: When the command is deactivated, all trace variables are already displayed in the selected diagram.

Subdialog 'Record Settings' Requirement: The top node is selected in the *“Trace Record”* tree view.

“Enable trigger”	☑: Triggering is enabled. The trace data is buffered at runtime only when a trigger signal has been sent. You determine how the trigger signal is sent in the *“Trigger variable”*, *“Trigger parameter”*, *“Trigger edge”*, *“Post-trigger (Samples)”*, and *“Trigger level”* settings. ☐: Continuous display of current records
“Trigger variable”	Signal that is used as a trigger. A complete instance path is required. A valid trigger signal is an IEC variable, a property, a reference, a pointer, an array element of the application, or an expression. Allowed types are all IEC-based types **except** `STRING`, `WSTRING`, and `ARRAY`. Enumerations are allowed when the base type is not `STRING`, `WSTRING`, or `ARRAY` The contents of a pointer are not a valid signal. When the runtime system uses the `CmpTraceMgr` component, a property that is linked to the `'monitoring'` attribute can then be recorded as a variable.
“Trigger parameter”	System parameter that is used as a trigger The *“Input Assistant”* dialog lists all valid system parameters in the *“Parameters”* category of the *“Categories”* tab.
▾	Allows the selection of *“Trigger variable”* or *“Trigger parameter”*

"Trigger edge"	Defined the edge detection for triggering: ● *"positive"* – For Boolean trigger variables, triggering occurs when the values changes from `FALSE` to `TRUE`. – For analog trigger variables, triggering occurs when the value as defined in *"Trigger level"* is reached from below. ● *"negative"* – For Boolean trigger variables, triggering occurs when the values changes from `TRUE` to `FALSE`. – For analog trigger variables, triggering occurs when the value as defined in *"Trigger level"* is reached from above. ● *"both"* – For Boolean trigger variables, triggering occurs when the values changes. – For analog trigger variables, triggering occurs when the value as defined in *"Trigger level"* is reached.
"Post trigger (samples)"	Number of records per trace variable that are buffered after triggering. Default: `50`; value range: `0` to (2^{32} `- 1`)
"Trigger level"	Value that is reached to start the triggering
"Task"	Task in which the data is recorded.
"Recording condition"	At runtime, the application checks the recording condition. If it is fulfilled, then the trace data is buffered. Record condition for data recording with `CmpTraceMgr` runtime system component: ● As an expression that includes only permitted operators and operands. Allowed operators that can also be nested: (logical) `AND`, `NOT`,`OR`, comparison operators `<`, `<=`, `>`, `>=`, `=`, `<>`. Allowed operands: Variables that are valid for trace. ● As a variable. Allowed type: `BOOL`, bit access, property. The condition is fulfilled for `TRUE` or `1`. The contents of a pointer are not permitted. Recording condition for a data recording with IEC code. ● As an expression that returns a Boolean value.
"Comment"	Comment (for example, from the recording condition)
"Resolution"	Unit of measure for the time stamp that is recorded per data set ● *"ms"*: Time stamp (in milliseconds). ● *"µs"*: Time stamp (in microseconds) for a task cycle time of 1 ms or less
"Automatic restart"	☑: Persistently saves the trace configuration and the last contents of the RTS buffer to the target device. After the device has been restarted, the trace is started automatically if the trigger has not occurred yet.
"Advanced"	Opens the *"Advanced Trace Settings"* dialog.

See also

- *Chapter 1.4.1.20.4.15.1 "Dialog 'Advanced Trace Settings'" on page 1208*

Subdialog 'Variable Settings'

Requirement: A trace variable is selected in the *"Trace Record"* or *"Display (Diagrams)"* tree view.

"Variable"	Valid variable Variable; value recorded with full instance path. Valid: ● IEC variable ● Property ● Reference ● Contents of the pointer ● Array element Allowed data type ● IEC-based type **except** `STRING`, `WSTRING`, or `ARRAY` ● Enumeration when the base type is not `STRING`, `WSTRING`, or `ARRAY` When the runtime system uses the `CmpTraceMgr` component, a property that is linked to the `'monitoring'` attribute can then be recorded as a variable.
"Parameter"	Parameter whose data is recorded. Requirement: Runtime system with `CmpTraceMgr` component The *"Input Assistant"* dialog lists all valid system parameters in the *"Parameters"* category of the *"Categories"* tab.
	Allows toggling between *"Variable"* and *"Parameter"*
"Color"	Color of the variable in the trace diagram
"Line type"	Display as line chart ● *"Line"*: Values are linked to form a line. ● *"Step"*: Values are linked in the form of steps ● *"None"*: Values are not linked
"Point type"	Display as scatter chart ● *"Dot"*: Value is displayed as a dot ● *"Cross"*: Value is displayed as a cross. ● *"None"*: value is not displayed
"Activate minimum warning"	☑: Warning when less than the lower limit
"Critical lower limit"	If the value of the trace variable falls below the limit, the variable is displayed in the warning color.
"Color"	Warning color on falling below the limit
"Activate maximum warning"	☑: Warning when exceeding the upper limit
"Critical upper limit"	If the value of the trace variable exceeds the upper limit, the variable is displayed in the warning color.
"Color"	Warning color on exceeding the limit

Subdialog 'Display Mode' Requirement: An axis is selected in the tree view *"Presentation (Diagrams)"*

"Display Mode"	Scaling ● *"Auto"*: Automatically scaled time axis ● *"Fixed length"*: Time axis segment with a constant *"Length"* ● *"Fixed"* Time axis segment from *"Minimum"* to *"Maximum"*
"Minimum"	Literal, variable (integer data type), or constant variable (integer data type). It contains the initial value of the segment. Requirement: The *"Display Mode"* is *"Fixed"*. Examples: `20,PLC_PRG.iLimit_Min, GVL.c_iLimit_Min` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Min : INT := 20`
"Maximum"	Literal, variable (integer data type), or constant variable (integer data type). It contains the end value of the segment. Requirement: The *"Display Mode"* is *"Fixed"*. Examples: `80,PLC_PRG.iLimit_Max, GVL.c_iLimit_Max` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Max : INT := 80`
"Length"	Constant segment length; the initial value is adapted automatically.
"Grid"	☑: Diagram with grid line in the X-direction. Select the grid line color from the list box of colors.

Table 241: "Tick marks"

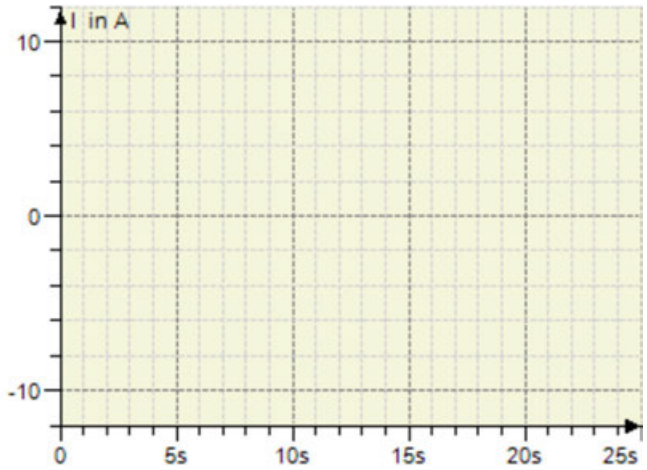
"Fixed spacing"	☑: Display of tick marks with *"Distance"* and *"Subdivisions"*.
"Distance"	Distance between tick marks
"Subdivisions"	Number of subdivisions between two tick marks

"Font"	Font for the time axis.

Link *"Preview"*	Displays the preview of the diagram.

Diagram preview

Requirement: A diagram is selected in the tree view *"Presentation (Diagrams)"*



“Background color”	Background color of the diagram. ▾ opens the list box of colors.
“Background color on selection”	Background color of the selected diagram.

Link *“Add Variable”*	Adds a new trace variable (in the *“Trace Record”* tree view).
Link *“Delete Variable”*	Deletes the selected trace variable (in the *“Trace Record”* tree view).
Link *“Add Diagram”*	Adds a new diagram (in the *“Display”* tree view).
Link *“Delete Diagram”*	Deletes the selected diagram (in the *“Display”* tree view).
Link *“Reset Display Settings”*	Resets the display settings of either the selected diagram or Y-axis to the default values.

“OK”	Accepts the configuration changes and saves the trace configuration.

Dialog Box 'Trend storage'

Function: This dialog box includes the configuration for buffering the trend data of a trend recording.

Call: *“Trend Storage”* button in the editor of a trend recording.

“Maximum number of variables”	Maximum number of trend variables that can be managed in the database. If you increase this value afterwards, then will CODESYS perform a download and reconfigure the database.
“Store every N milliseconds”	Time interval (in ms) when the `CmpTraceMgr` runtime system component buffers the recorded data before storing it persistently in the database. The application calculates internally the number of task cycles from the time interval. The duration of a task cycle is defined in the task configuration. A high value results in better runtime performance. The disadvantage is the increased risk of losing data if the controller crashes or shuts down. A low value reduces this risk. The disadvantage is the slower control over a trend visualization with large amounts of data.
“Limit”	: Limit the recording ● *“No Limit”*: Unlimited number of data records (not recommended) ● *“Maximum number of records”*: Maximum number of data records that are stored in the database. A data record consists of time stamp and the values of the trend variables at this time. ● *“Maximum storage size”*: Maximum size of the trend storage. The application calculates internally the number of data records. Clicking the "down" symbol (▾) of the drop-down list will set the units to kilobytes (KB), megabytes (MB), or gigabytes (GB).

See also

- *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*
- *Chapter 1.4.1.12.4.1 “Getting started with trend recording” on page 431*

Dialog Box 'Advanced Trend Settings'

Function: This dialog box provides more settings for configuring trend recording.

Call: Click *“Advanced”* in the editor of a *“TrendRecording”* object.

“Measure in every n-th cycle”	Frequency that the runtime system records data, depending on the number of processed task cycles. Select a value from the drop-down list or type a value into the input field. Using the settings from the task configuration, CODESYS calculates the time interval according to the frequency. Therefore, the calculation is possible only if at least the task cycle time is set. The result is shown on the right of the input field in normal syntax (for example, *“1h1m1s1ms”*). Default: `1` means that data is recorded in each task cycle.
“Additional Runtime Buffer for”	Length of the time interval when the runtime system can record more data (for example, `1000 ms`). If a delay occurs when writing data in the runtime system component, then there is a risk of data loss due to overwriting. In this case, the runtime system uses the addition buffer.

See also

- *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*
- *Chapter 1.4.1.12.4 “Data Recording with Trend” on page 430*
- *Chapter 1.4.1.12.4.2 “Configuring trend recording” on page 432*

Dialog 'Certificate Selection'

Symbol:

Function: This dialog is used for selecting the certificates for encryption, decryption, and digital signatures.

Call:

- *“Security Screen”* view, *“User”* tab
- Main menu: *“View ➔ Properties”*, *“Encryption”* tab when the *“Application”* is selected in the device tree.
- Main menu: *“Project ➔ Project Settings”*, category *“Security”*

Dialog 'Certificate selection'

The purpose of the certificate that is selected in the dialog depends on the call location: ● Call location: *"Security screen"* view, *"User"* tab – Certificate for digital signatures – Certificate for the decryption of project files – Certificate for encrypted communication ● Call: *"View ➔ Properties"* of the application – *"Certificates of devices that share the encrypted download and the boot application"* ● Call location: *"Project ➔ Project settings"*, category *"Security"* – *"Certificate for project encryption"*	
Listing of the selected certificates in a table	The following properties are displayed for each selected X.509 certificate: ● *"Created for"* ● *"Created by "* ● *"Valid as of"* ● *"Valid until"* ● *"Thumbprint"*: SHA1 fingerprint Double-clicking an entry opens the *"Certificate"* dialog with the *"General"* tab, *"Details"* tab, and *"Certification Path"* tab. In that tab, you will find a reference to Windows help with more information about the dialog.
	Adds the selected available certificate to the list of selected certificates.
	Deletes the certificate selected in the list.
"Available certificates in the local Windows Certificate Store"	Double-clicking an entry opens the *"Certificate"* dialog with the *"General"* tab, *"Details"* tab, and *"Certification Path"* tab. In that tab, you will find a reference to Windows help with more information about the dialog.
Certificate icons	●  ●  : Certificate with private key ●  : Untrusted certificate

See also

- *Chapter 1.4.1.20.3.3.18 "Command 'Security Screen'" on page 995*

1.4.2 Fieldbus Support

1.4.2.1 Device Diagnosis

CODESYS provides general and fieldbus-specific function blocks for performing a diagnosis on the connected devices.

General diagnosis

You can perform a diagnosis on devices regardless of the fieldbus. The function blocks from the `CAA Device Diagnosis` library are provided for this purpose.

Before you can work with these function blocks, you have to select the *"Enable diagnosis for devices"* option in the PLC settings. This causes CODESYS to create instances of the diagnosis functions blocks automatically. These function blocks can be used for your diagnosis.

Work exclusively with the automatically generated instances of the diagnosis function blocks. Do not create your own instances.

See also

- *Chapter 1.4.1.20.2.8.9 “Tab 'PLC Settings'” on page 850*
- *Library CAA DeviceDiagnosis*

Bus-specific diagnosis

For bus-specific diagnosis options, see the diagnosis chapters of the individual fieldbuses.

1.4.2.2 Fieldbus Devices and I/O Drivers

The technical basis for each fieldbus device, which is configured in the device tree, is the CODESYS I/O driver.

The I/O driver is the link between the fieldbus stack, the IEC application, and the CODESYS IDE. The driver configures the fieldbus stack from the data of the device configuration. It shows the diagnosis, provides an API for the IEC application, and is responsible for the I/O mapping (see chapter "I/O Mapping").

This chapter provides a brief overview of the basic functionality of CODESYS I/O driver devices, without discussing the details of specific bus systems. In addition, some recommendations for the configuration are provided.

Bus cycle task

The bus cycle task is the IEC task in whose context the I/O driver is executed. Some I/O drivers use multiple tasks: usually one real-time critical task (with high priority), which is used for the transfer of I/O data, and another task with low priority for tasks such as evaluating diagnostics and executing acyclic services of the bus system.

With real-time critical bus systems, it has to be ensured that no operations are executed in the context of this bus task that would interrupt the bus clock due to the execution time.

The bus task can be configured in the I/O mapping dialog of the I/O driver device. Note that the settings of the parent device are inherited by default. If this device is the PLC, then its PLC setting applies in the bus cycle task.

NOTICE!

If this above setting is not set, then the task with the shortest cycle time is used. In this way, a non-real-time I/O driver can be executed unintentionally in the task context of a real-time critical driver, thus interrupting its communication. To diagnose these communication problems, it is recommended to check the task monitoring.

See also

- *Table 64 ““Bus Cycle Options”” on page 851*

I/O mapping

An essential function of an CODESYS I/O driver is to update the I/O mapping. This means the mapping of the I/O data of the bus system to variables of the IEC application (and vice versa).

The input/output data is mapped cyclically by copy and conversion operations in both directions from the internal memory image of the bus system to IEC variables assigned to %I and %Q addresses.

For the I/O driver, there is no internal difference whether symbolic names or "direct" access to the %I and %Q addresses are used for this I/O mapping. For the maintainability of the application, it is recommended to always use descriptive variable names (example: variable "TemperatureReactor" instead of "%IW117" access).

The updating of the I/O mapping can be set with *“Always update variables”* (globally in the *“PLC Settings”* or individually for each device in the I/O mapping dialog):

- Disabled:
  Only I/O data used in the application is mapped.
  This may improve performance by avoiding the copy operations, but may cause confusion if the I/O data in the I/O mapping dialog is not updated (the values are then grayed out). This setting is recommended for an application whose development has been completed.
- Enabled 1:
  All data is updated.
- Enabled 2:
  Caution: For productive use in special cases only.
  As a result, inconsistent I/O data may occur, because the bus cycle task reads/writes this data while the application code uses it in other tasks.
  See „Consistency of I/O data“.

See also

- *Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*

Consistency of I/O data

The CODESYS programming system allows the IEC application to use multiple tasks executed in parallel (for visualization, field buses, or other POUs). The application code can access I/O data from the context of these tasks via the mapped IEC variables. By accessing the same data from different tasks, inconsistent or corrupt data could occur (for example, due to interrupted write access).

The I/O driver ensures data consistency by providing each task executing a task cycle with a consistent mapping – a snapshot, so to speak – of all I/O data used.

So a code like in the following example cannot cause problems: (Note "DIV by ZERO")

```
IF(inputData <> 0) THEN                 // inputData is mapped to %I
    x := y / inputData;                 // This will never result in
DIV_BY_ZERO Exception
END_IF                                  // inputData is not updated by
bus cycle during execution of POU
```

NOTICE!
With the *“Always update variables”* option set to *“Enabled 2 – always in bus cycle task”*, this mechanism is overridden. Accordingly, the application code has to take this into account.

Services

In addition to the basic functionality, some I/O drivers provide services that can be called from the CODESYS IDE, such as the device scan function or the setting of device addresses.

General recommendations

Settings:

- *“PLC Settings”*: ☑ I/O updates in stop:
  The bus cycle continues even when the application is stopped, for example when the application is on a debug breakpoint. In this way, communication with the field devices is maintained and can be continued immediately without interruption.
- *“PLC Settings”*: *“Always update variables”* is set to *“Enabled 1 – use bus cycle task if not used in any task”*:
  During the development of the application, it is useful to see the values of all I/O data.

Task configuration:

- Especially for real-time critical fieldbus systems such as Profinet, EtherCAT, or CAN, which depend on maintaining an exact send/receive clock, it is recommended to use a separate bus cycle task with high priority. For less real-time-critical tasks (for example, visualization) a significantly lower priority should be selected than for the bus cycle task.
- In order to achieve maximum I/O throughput with as little offset as possible, separate POUs can be executed in the bus task of the fieldbus system. However, these then have to meet the real-time requirements: for example, no file access or blocking socket functions may be executed, but for example only the calculation of the output data.

Multiple I/O drivers and tasks (trouble-shooting)

If consistent access to I/O data from multiple tasks and possibly across multiple I/O driver instances has to be synchronized, then undesired reciprocal interference between the bus and application task may occur under certain circumstances.

This is the case, for example, when the general system load is high or when the I/O data of the real-time critical fieldbus system is used together with I/O data of a slow and blocking local bus system in the same task.

In case of unexpected interference of the communication, with the particularly real-time-critical fieldbuses (EtherCAT, Profinet, CAN), the task monitoring should therefore first be examined for very large jitter or outliers in the cycle time (maximum value compared to average value). The task list provides detailed information about the use of I/O data in different tasks.

It may be possible to avoid using I/O data from different bus systems in one and the same task or to reduce the number of I/O tasks.

See also

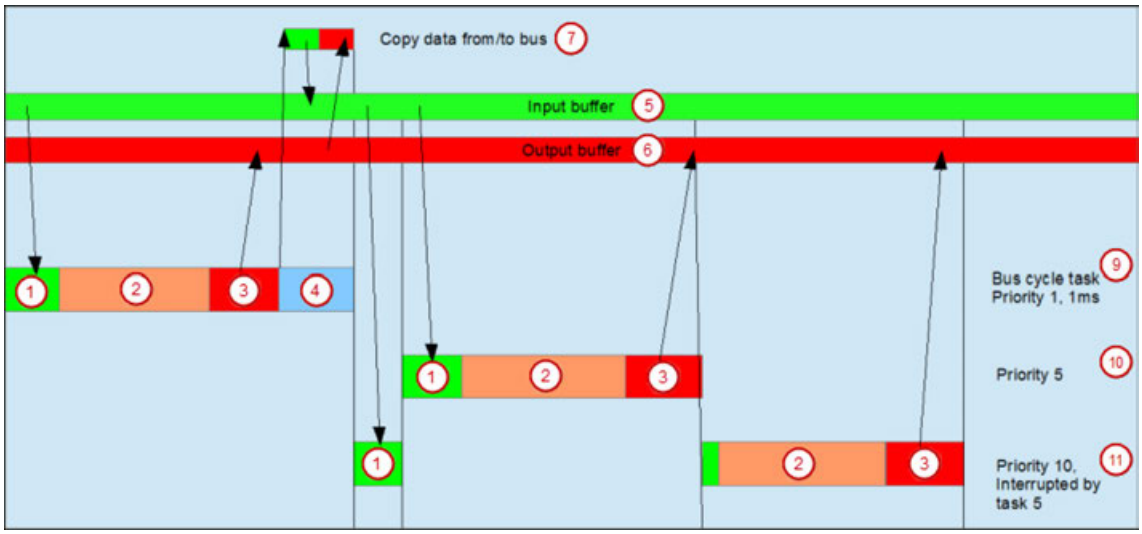
- *Chapter 1.4.1.20.2.8.17 “Tab 'Task deployment'” on page 869*

1.4.2.3 Bus Cycle Task

Generally, for each IEC task, the used input data is read at the start of each task (1) and the written output data is transferred to the I/O driver at the end of the task (3). The implementation in the I/O driver is decisive for additional transfer of the I/O data. It is responsible for the time frame and time point that the actual transfer to the corresponding bus system occurs.

The bus cycle task of the PLC can be defined globally for all fieldbuses in the PLC settings. For some fieldbuses, however, you can change this independent of the global setting. The task with the shortest cycle time is used as the bus cycle task (setting: *“unspecified”* in the PLC settings). The messages are normally sent on the bus in this task.

Other tasks copy only the I/O data from an internal buffer that is exchanged only with the physical hardware in the bus cycle task.



```
(1) Read inputs from input buffer              (2) IEC task
(3) Write outputs to output buffer      (4) Bus cycle
(5) Input buffer                                    (6) Output buffer
(7) Copy data to/from bus
(9) Bus cycle task, priority 1, 1 ms
(10) Bus cycle task, priority 5
(11) Bus cycle task, priority 10, interrupted by task 5
```

Task usage

The *“Task Deployment”* tab provides an overview of used I/O channels, the set bus cycle task, and the usage of channels.

WARNING!
If an output is written in various tasks, then the status is undefined, as this can be overwritten in each case.

If the same inputs are used in various tasks, then it is possible for the input to change during the processing of a task. This happens when the task is interrupted by a task with a higher priority and causes the process image to be read again. Solution: At the beginning of the IEC task, copy the input variables to variables and then work only with the local variables in the rest of the code.

Conclusion: Using the same inputs and outputs in several tasks does not make any sense and can lead to unexpected reactions in some cases.

1.4.2.4 EtherNet/IP Configurator

Refer to the general description for information about the following tabs of the device editor.

- *Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*
- *Chapter 1.4.1.20.2.8.12 “Tab '<device name> IEC Objects'” on page 859*
- *Chapter 1.4.1.20.2.8.3 “Tab 'Parameters'” on page 844*
- *Chapter 1.4.1.20.2.8.18 “Tab 'Status'” on page 870*
- *Chapter 1.4.1.20.2.8.19 “Tab 'Information'” on page 870*

Only in the case of special features is there an additional help page for the specific device editor.

If the "<device name> Parameters" tab is not shown, then select the “Show generic device configuration editors” option in the CODESYS options (“Device Editor” category).

An EtherNet/IP network consists of an EtherNet/IP scanner and one or more EtherNet/IP adapters. In this case, the scanner is the master in the network and the adapters are the slaves.

The CODESYS runtime can act as either a scanner or an adapter.

CODESYS continues to differentiate between a remote adapter and a local adapter.

- **EtherNet/IP Remote Adapter**: In CODESYS, a remote adapter is a device that you insert in the device tree of a project below an EtherNet/IP scanner.
- **EtherNet/IP Local Adapter**: In CODESYS, a local adapter is a device that you insert in the device tree of a project directly below an Ethernet adapter (TCP) or Modbus port (COM). As a result, you can use the CODESYS runtime as a EtherNet/IP adapter.

See also

- *Device Editor Options*

1.4.2.4.1 EtherNet/IP Bus Cycle Task

General information

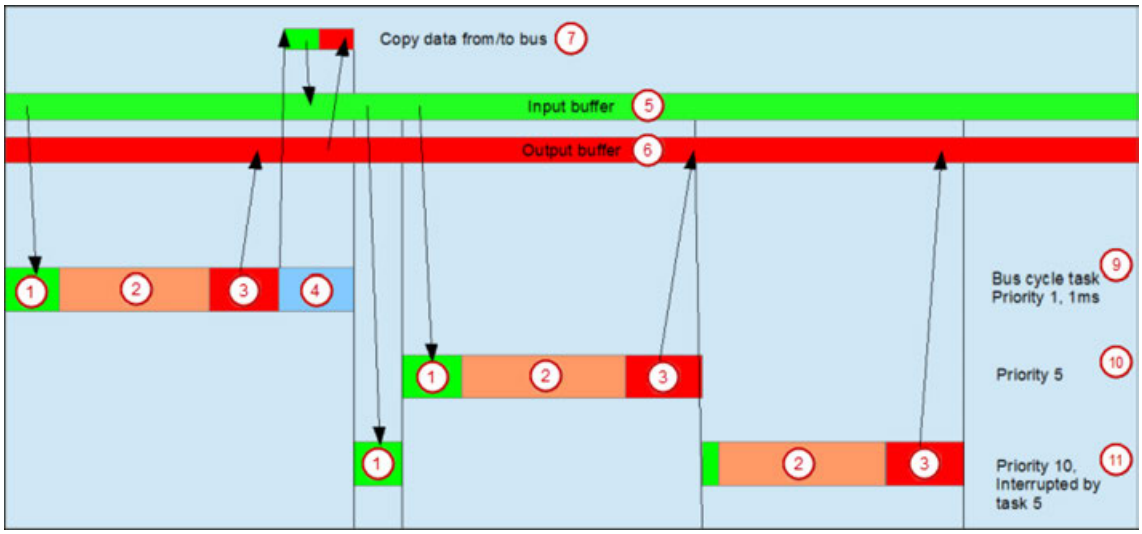
By "bus" it means all fieldbuses including I/O bus. There is no bus cycle task for Modbus because it is controlled by POUs. Modbus does not provide IO mapping.

It's recommended to define a dedicated bus cycle task for each fieldbus configured in the project. It's strongly recommended not to use "unspecified" in the "*"PLC Settings"*" to avoid unexpected behavior. The task defined in *"PLC Settings"* determines the bus cycle task of I/O bus and, depending on the configuration, of the additional fieldbuses (the setting is by default inherited).

Especially in case of EtherCAT, a dedicated bus cycle task should be used which is not shared with other fieldbuses. If *[unspecified]* is set in *"PLC Settings"*, the EtherCAT task might be automatically used by other fieldbuses, potentially causing EtherCAT task processing to fail. This should be avoided by specifying a task different to the EtherCAT task in *"PLC Settings"*.

As a rule, for each IEC task the used input data is read at the start of each task and the written output data is transferred to the I/O driver at the end of the task . The implementation in the I/O driver is decisive for further transfer of the I/O data. The implementation is therefore responsible for the timeframe and the specific time when the actual transmission occurs on the respective bus system.

Other tasks copy only the I/O data from an internal buffer that is exchanged only with the physical hardware in the bus cycle task.




```
(1) Read inputs from input buffer        (2) IEC task
(3) Write outputs to output buffer       (4) Bus cycle
(5) Input buffer                         (6) Output buffer
(7) Copy data to/from bus
(9) Bus cycle task, priority 1, 1 ms
(10) Bus cycle task, priority 5
(11) Bus cycle task, priority 10, interrupted by task 5
```

Using tasks

The *"Task Deployment"* provides an overview of used I/O channels, the set bus cycle task, and the usage of channels.

WARNING!

If an output is written in various tasks, then the status is undefined, as this can be overwritten in each case.

When the same inputs are used in various tasks, the input could change when a task is processed. This happens if the task is interrupted by a task with a higher priority and causes the process map to be read again. Solution: At the beginning of the IEC task, copy the input variables to variables and then work only with the local variables in the rest of the code.

Conclusion: Using the same inputs and outputs in several tasks does not make any sense and can lead to unexpected reactions in some cases.

See also

- *Chapter 1.4.1.20.2.8.17 “Tab 'Task deployment'” on page 869*
- *Chapter 1.4.1.20.2.8.9 “Tab 'PLC Settings'” on page 850*

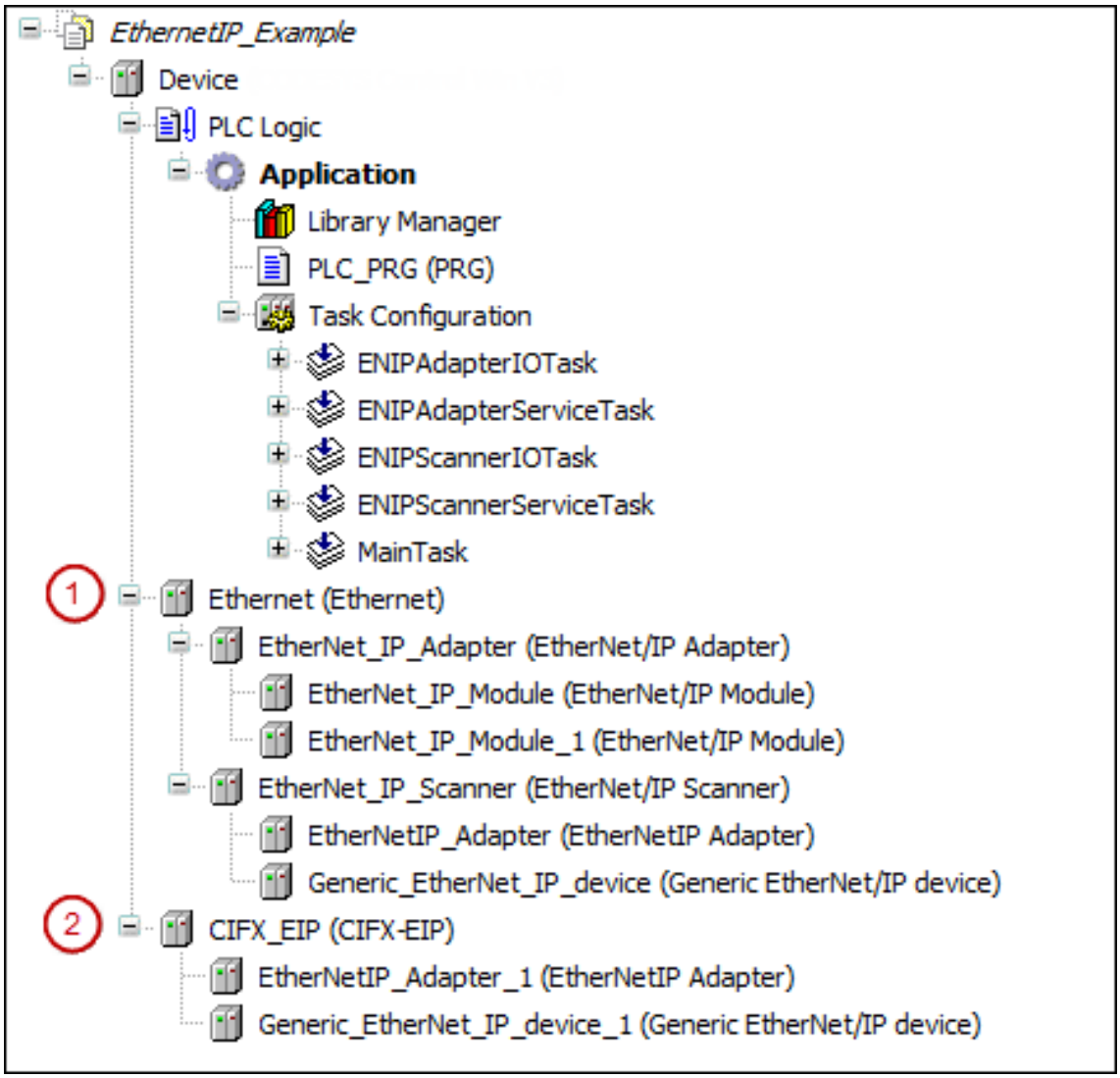
1.4.2.4.2 EtherNet/IP Scanner

CODESYS runtime as EtherNet/IP scanner

CODESYS provides two different EtherNet/IP scanners:

- (1): A device that you insert directly below each network adapter A CODESYS Ethernet/IP scanner (IEC) can also be an adapter at the same time – functionally an originator and an adapter in one.
- (2): A device that needs a special cifX adapter

You insert one or more EtherNet/IP adapters below a scanner.



See also

- *Chapter 1.4.2.4.2.1 “Tab 'EtherNet/IP Scanner - General'” on page 1223*
- *Chapter 1.4.2.4.2.2 “Tab 'EtherNet/IP Scanner NetX - General'” on page 1224*

Tab 'EtherNet/IP Scanner - General'

Object: EtherNet/IP Scanner

This tab in the configurator of the EtherNet/IP scanner includes the basic settings. The network interface used by the scanner is configured in the settings of the Ethernet adapter.

Table 242: “Options”

“Automatic restoring of connections ”	☑: The scanner always attempts to automatically re-establish an interrupted connection. For example, if a timeout is detected for UDP I/O messages or the TCP connection to the adapter is interrupted. If the option is activated, then the scanner reconnects to the adapters with the lost connection.

Tab 'EtherNet/IP Scanner NetX - General'

Object: EtherNet/IP Scanner NetX

This tab in the configurator of the EtherNet/IP scanner contains the basic settings for communication in the network.

Table 243: "Address Settings"

"Use static IP address"	
"IP address" *"Subnet mask"* *"Gateway address"*	These entries each occupy four bytes and serve to identify the scanner within the network environment
"Optain IP address automatically"	This option is available only for the NetX scanner
"BOOTP"	Assignment of the IP address by a server by means of Bootstrap Protocol (BOOTP)
"DHCP"	Automatic configuration of the network settings by the host by means of Dynamic Host Configuration Protocol (DHCP)

Table 244: "Ethernet Settings"

"Speed and duplex:"	Bit rate of the transmission. In case of *"Auto-negotiation"*, the highest of the available bit rates is selected automatically.

Table 245: "Options"

"Auto-reestablish connections"	☑: The scanner always attempts to automatically re-establish an interrupted connection. For example, if a timeout is detected for UDP I/O messages or the TCP connection to the adapter is interrupted. If the option is activated, then the scanner reconnects to the adapters with the lost connection.

Tab 'NetX Configuration'

Object: EtherNet/IP Scanner

As an alternative to the general EtherNet/IP node in case of a NetX field bus the EtherNet/IP node (NetX) can be added to the device tree. This node provides the additional NetX configuration dialog to select the NetX chip (slot for the card) and the communication channel of this chip. The name of the setting used in the dialog and the possible settings provided by the selection lists are defined by the device description.

Table 246: "NetX Settings"

"Slot"	Slot to be used. In case of PCI cards with NetX chip, the slot numbers usually correspond to the PCI card numbers.
"NetX Com channel"	Channel on the card to be used for the communication. A NetX board may have up to four communication channels for different fieldbusses.
"Auto-initialize bus"	☑ The user is asked to determine if the bus should be reinitialized when downloading or when resetting the application. A new initialization will interrupt the bus and may lead to unwanted behavior of the machine.

Tab 'EtherNet/IP Scanner - I/O Mapping'

Object: EtherNet/IP Scanner

Note: No project variables can be mapped to the outputs and inputs with the EtherNet/IP scanner.

See also

- *Tab '<device name> I/O Mapping'*

EtherNet/IP Remote Adapter

Tab 'EtherNet/IP-Adapter - General'

Object: EtherNet/IP Adapter

The tab in the device editor of the EtherNet/IP adapter contains the basic settings for network communication.

Table 247: “Address settings”

“IP address ”	Address for the identification of the EtherNet/IP adapter device.

Table 248: “BOOTP”

Bootstrap Protocol This option is available only for adapters under the NetX scanner.	
“MAC address”	Device-specific MAC address of the slave
“Save IP address”	☑: The address of the slave is saved. The requirement, however, is that the slave supports this function. This option is only available for the CIFX scanner.

Table 249: “Electronic keying”

“Compatibility check”	☑: The adapter uses its own keying values to perform a compatibility check of the keying values from the EDS file. All keying values are sent to the device. Then the device decides whether it is compatible with the received values. ☐: The adapter uses its own keying values to perform an exact check of the keying values from the EDS file. The user decides which keying information should be checked. ● Vendor ID ● Device type ● Product code ● Major revision ● Minor revision If the check fails, then an I/O connection is not established to the device and an error message is issued to the status page.
“Restore default values”	For generic devices only.

See also

- *Chapter 1.4.2.4 “EtherNet/IP Configurator” on page 1220*

Tab 'EtherNet/IP Adapter - Connections'

Object: EtherNet/IP Adapter

The upper part of this tab displays a list of all configured connections. When there is an "Exclusive owner" connection in the EDS file, it is inserted automatically when adding the adapter. The configuration data for these connections can be changed in the lower part of the dialog.

The configuration data is defined in the EDS file. The data is transmitted when the connection to the adapter is established.

“RPI (ms)”	Requested Packet Interval: Exchange interval of the input/output data
“O -> T size (bytes)”	Size of the producer data from the scanner to the adapter (**O**riginator --> **T**arget)
“T -> O size (bytes)”	Size of the consumer data from the adapter to the scanner (T --> O)
“Proxy Config Size (Bytes)”	Size of proxy configuration data
“Target Config Size (Bytes)”	Size of adapter configuration data
“Connection Path”	Address of the - configuration objects - input objects - output objects
“Add Connection”	Opens the *“New Connection”* dialog. The parameters for the new connection are determined here.
“Delete Connection”	Deletes the selected connection from the list
“Edit Connection”	Opens the *“Edit Connection”* dialog. The parameters for the existing connection are modified here.

Table 250: “Configuration Data”

The table shows the connections with the configuration parameters from the EDS file. The connections are divided into configuration groups.	
“Raw data values”	If the scaling parameters are defined in the EDS file for the data, then you can show the values as raw data or converted data. ☑: The data is displayed without conversion. In the case of Enum data types, the index of the enumeration value is shown. ☐: The data is displayed with conversion. In the case of Enum data types, the enumeration value is shown.
“Display parameter groups”	☑: If groups are defined in the EDS file, then the parameters that are defined in these groups are displayed in a sorted list.
“Defaults”	Resets to the default values
“Value”	Double-click to change the value. Depending of the data type, you can specify the value directly in the input field or select from a list box. In the case of bit field data types and deactivated raw data values, a dialog opens for you to choose the individual bits. Only those bits can be selected which fall within defined minimum and maximum values. If bit field data types contain enumerations in the associated EDS file, then only these enumerations are shown with the associated bit positions. If a connection contains a parameterizable connection path in the EDS file, then here you can modify the different parameters of the respective connection.

See also

- *Chapter 1.4.2.4.2.5.3 “Dialog 'New Connection'” on page 1227*
- *Chapter 1.4.2.4 “EtherNet/IP Configurator” on page 1220*

Dialog 'New Connection'

Object: EtherNet/IP Adapter

Generic connection (freely configurable)

The dialog contains the parameters for the new connection.

Table 251: “Connection Path Settings”

“Automatically generated path”	The *“Connection Path”* is generated automatically from the values for *“Configuration assembly”*, *“Consuming assembly”*, and *“Producing assembly”*.
“User-defined path”	The *“Connection Path”* is specified manually in the corresponding input field.
“Path defined by symbolic name”	The path is specified by a symbolic name. Requirement: The device must support symbolic connection paths.

Table 252: “General Parameters”

“Connection Path”	The connection path is used to address one or more objects in the adapter that provide the input data and receive the output and configuration data. Requirement: The connection path is set to *“User-defined path”*.
“Symbolic name”	An ANSI string is used instead of the normal connection path. See the manual of the respective EtherNet/IP adapter for permitted ANSI strings. Requirement: The connection path is set to *“Path defined by symbolic name”*.
“Trigger type”	● *“Cyclic”*: Data exchange takes place cyclically at intervals set by the RPI. ● *“Change of State”*: Data is exchanged automatically after a change to the scanner outputs or adapter inputs. ● *“Application”*: Not implemented
“Transport Type”	Details for this can be taken from the specifications CIP Volume 1 and Volume 2.
“RPI (ms)”	(Requested Packet Interval) Length of the time interval (in milliseconds) in which the transmitting application requests the transmission of data to the target application. This value must be a multiple of the bus cycle task.
“Timeout multiplier”	In case of device failure, there is a time delay (RPI * Timeout multiplier) before the device state switches to "Error".

Predefined connection (EDS file)

Use this option to employ existing connections from an EDS file. The data that can be changed are defined in the EDS file.

Table 253: “Scanner to Adapter (Output)”

“O--> T size (bytes)”	Amount of data from scanner to adapter
“Proxy Config Size (Bytes)”	Size of proxy configuration data
“Adapter Config Size (Bytes)”	Size of adapter configuration data

“Connection type”	● *“Null”*: A network connection is not established. ● *“Multicast”*: A network connection is established. The connection data can be received by multiple consumers. ● *“Point to Point”*: A network connection is established. The connection data can be received by exactly one consumer.
“Connection Priority”	Two scanners using different priorities to one adapter can cause conflicts. Adapting the connection priority solves this problem.
“Fixed/Variable”	See the specifications CIP Volume 1 and Volume 2 for details of the parameters.
“Transfer format”	
“Inhibit time”	
“Heartbeat multiplier”	Requirement: The *“Transfer format”* is *“Heartbeat”*. Extends the interval at which the scanner sends heartbeat messages to the adapter. This value is multiplied by the *“RPI”* value. Example: *“RPI”* = 10ms and *“Heartbeat multiplier”* = 10 causes a message to be sent every 100ms.

Table 254: “Adapter to Scanner (Input)”

“T--> O size (bytes)”	See description for *“Scanner to Adapter”*.
“Connection type”	
“Connection Priority”	
“Fixed/Variable”	
“Transfer format”	
“Inhibit time”	

See also

- *Chapter 1.4.2.4.2.5.2 “Tab 'EtherNet/IP Adapter - Connections'” on page 1226*

Tab 'EtherNet/IP Adapter - Assemblies'

Object: EtherNet/IP Adapter

The upper part of this tab displays a list of all configured connections. When a connection is selected, the associated assemblies in the lower area of the tab are displayed.

Table 255: Connections

A description of the columns is found on the "Connections" tab.

Table 256: “Output Assembly”, “Input Assembly”

“Add”	Opens the *“Select Parameters”* dialog.
“Delete”	Deletes all selected parameters.
“Move Up” *“Move Down”*	Moves the selected parameter within the list. The order in the list determines the order in the I/O mapping.

“Name” *“Bit length”* *“Help string”*	You can double-click in the text field to edit the values.
“Show filling bytes of assemblies”	☑: The filling bytes of the assemblies are shown in the I/O mapping. This can be helpful in case the parameter layout of the assemblies is not mapped correctly in the EDS file.

See also

- *Chapter 1.4.2.4.2.5.2 “Tab 'EtherNet/IP Adapter - Connections'” on page 1226*

Dialog 'Select Parameters'

“Display parameter groups”	☑ The dialog displays all parameters from the EDS file by group. ☐ The dialog displays all parameters from the EDS file in a flat structure. Individual parameters from this list can be selected and added to the list of assemblies by clicking *“OK”*.
“Generic parameters”	☑ You can add generic parameters. Individual values of the parameter can be edited.

Tab 'EtherNet/IP Adapter - User Parameters'

Object: EtherNet/IP Adapter

The tab displays all additional parameters that are transmitted once only into the bus system during the phase of the starting procedure allotted to this.

> **NOTICE!**
> The user parameters are also transmitted again when a connection is reestablished, for example after the failure of a remote adapter.

“New”	Opens the *“Select Parameters”* dialog for adding a new parameter. The new parameter is inserted before the selected line.
“Modify”	Opens the *“Select Parameters”* dialog for changing an existing parameter.
“Move Up”, *“Move Down”*	Changes the order of the user parameters. The order of the parameters in the list corresponds to the order at the initialization.
“Value”	The value of the respective parameter can be changed directly by double-clicking the value. If applicable, a list box opens containing possible values.
“Abort If Error”	☑: In case of error, the entire transmission of the parameters is aborted.
“Jump to Line If Error”	☑: In case of error, the program resumes with the line specified in the *“Next Line”* column. In this way, an entire block can be skipped during the initialization, or a return can be defined. Note: A return can lead to an infinite loop if it is never possible to write a certain parameter.

See also

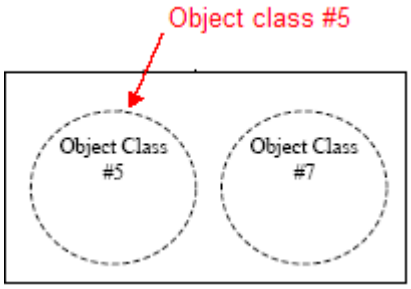
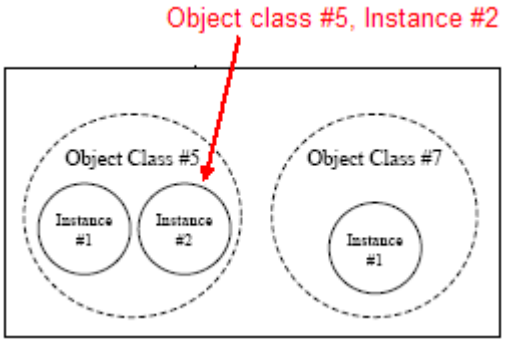
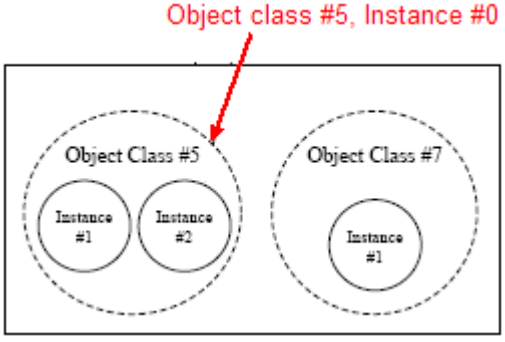
- *Chapter 1.4.2.4.2.5.6 “Dialog 'Select Parameters'” on page 1230*

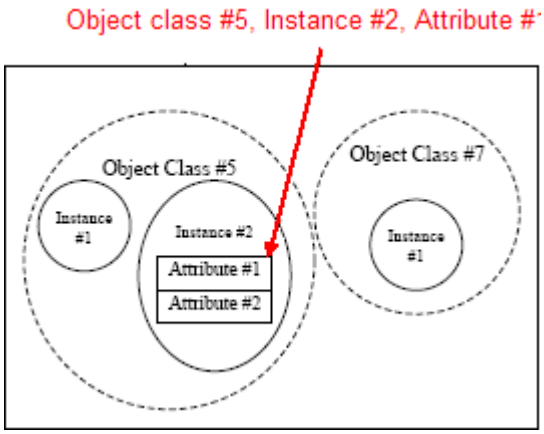
Dialog 'Select Parameters'

Object: EtherNet/IP Adapter

The dialog contains a list of the parameters that are defined in the EDS file. You can define your own generic parameters in addition to the specified parameters.

The values of the selected parameter are displayed in the lower section of the dialog. They can be changed there.

“Display parameter groups”	☑: Display of the parameters sorted by parameter groups
“Generic parameters”	☑: Enables the creation of generic parameters
“Name”	Name of the generic parameter
“Class”	Each object class that can be addressed by the network is identified by an integer value.  A class can also be addressed from the class by specifying a special object instance (see *“Instance”*).
“Instance”	Integer value for the unique identification of an object instance within a class. Example of an object instance:  If the value 0 is assigned to the instance, then the class itself is referenced by this special instance. Example – object instance 0: 

“Attribute”	Integer value that can belong to a certain class or instance. Example attribute: 

The values for “Class”, “Instance”, and “Attribute” are defined in the "CIP Networks Library" (Vol. 1 and 2) or in the manual of the device manufacturer.

CAUTION!
When individual values are entered, a plausibility check is not performed. Any errors are identified only when the bus is started and they are reported with a message in the log file.

See also

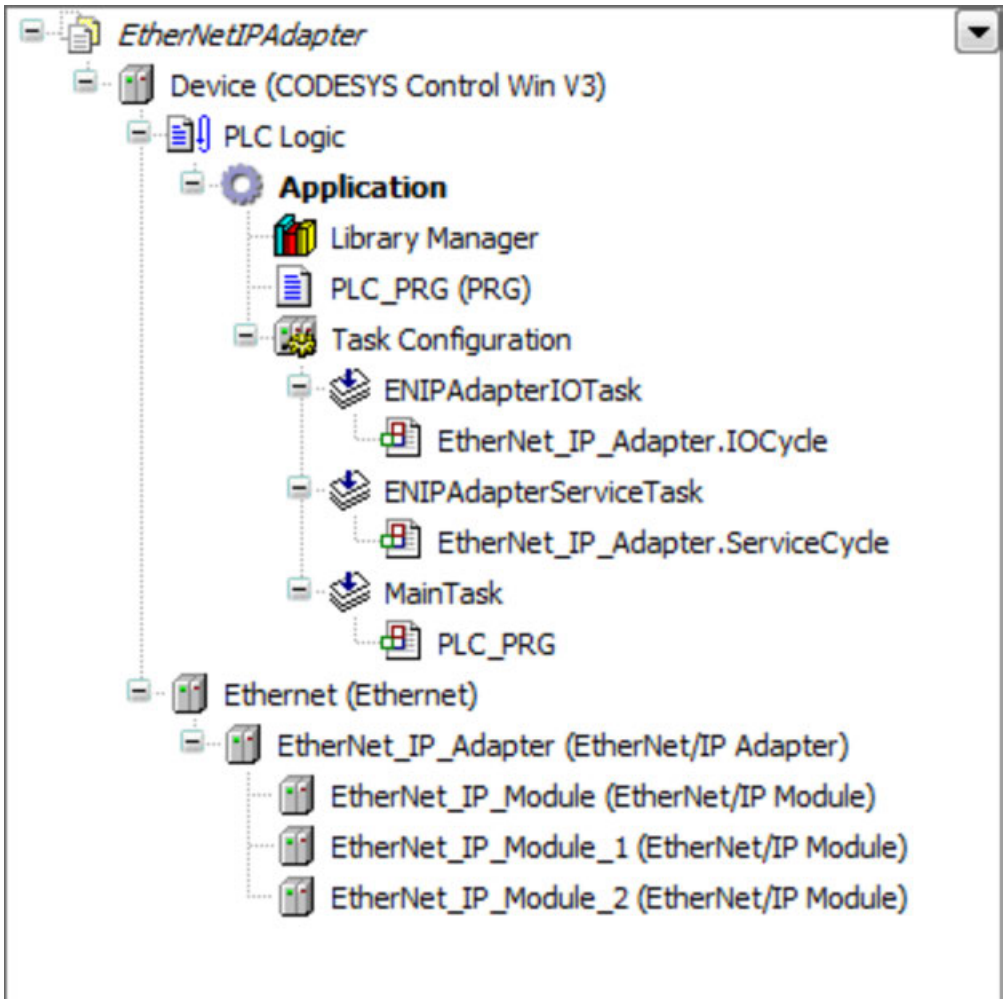
- *Chapter 1.4.2.4.2.5.5 “Tab 'EtherNet/IP Adapter - User Parameters'” on page 1229*

1.4.2.4.3 EtherNet/IP Local Adapter

CODESYS runtime as EtherNet/IP adapter

First, you insert the EtherNet/IP adapter below an Ethernet adapter. Then, you insert the modules below the EtherNet/IP adapter.

The sum of the input and output data of the modules determines the connection size of the adapter.



See also

- *Chapter 1.4.2.4.3.1 “Tab 'EtherNet/IP-Adapter - General'” on page 1232*
- *Chapter 1.4.2.4.3.3.1 “Tab 'EtherNet/IP Module - General'” on page 1233*

Tab 'EtherNet/IP-Adapter - General'

Object: EtherNet/IP Adapter

The device editor tab shows general information from the device description file. You can modify these values.

Table 257: “EDS File”

“Vendor name”	
“Vendor ID”	Provided by the ODVA (Open DeviceNet Vendors Association)
“Product name”	Values from the EDS file
“Product code”	
“Major revision”	
“Minor revision”	

“Enable ACD”	☑ Enables the ACD functionality (Address Conflict Detection) for the EtherNet/IP adapter. Note: The ACD functionality is normally applied by the operating system. Therefore, the user should only use this function very conscientiously. By enabling ACD, complications can result between the controller and the operating system. ACD is a mechanism that EtherNet/IP devices can use to detect and respond to IPv4 address conflicts. The ACD mechanism used in EtherNet/IP complies with the IETF RFC 5227 standard.
“Install to Device Repository”	If a device with the same device identification has already been installed, then you are asked whether the device should be overwritten. If the device is inserted as a remote adapter below an EtherNet/IP scanner, then you will be asked to automatically update the device.
“Export EDS File”	The EDS file is created and stored on the local computer. In this way, the EDS file can be used in an external configuration file.

Tab 'EtherNet/IP Adapter - Tags'

Object: EtherNet/IP Adapter

The tab of the device configurator is used for communication between an EtherNet/IP scanner and an EtherNet/IP adapter. The tab shows all device connections from the device description. The user can define a connection tag for each of these device connections.

No additional connections can be added on this tab.

Requirement: This tab is displayed only if the device description contains the parameter `ShowTagsPage` and the value of the parameter is set to `TRUE`.

Table with the device connections defined in the device description.	
“Connection Name”	Information originates from the device description Not editable
“ Transport Type”:	Information originates from the device description Not editable
“Connection Path”	Information originates from the device description Not editable
“Symbolic Connection Tag”	Connection tag for the connection predefined in the device description. Specified by the user.

See also

- *Chapter 1.4.2.4.3.1 “Tab 'EtherNet/IP-Adapter - General'” on page 1232*

EtherNet/IP Module

Tab 'EtherNet/IP Module - General'

Object: EtherNet/IP Adapter

This device editor tab displays general information from the device description file: You can adjust these values.

Table 258: “Module Information”

“Module”	Provides a selection of all module EDS files stored in the device description. The I/O data is then read from the selected module EDS to create corresponding I/O channels.
“Vendor”	
“Vendor ID”	Provided by the ODVA (Open DeviceNet Vendors Association)
“Product name”	Values from the EDS file
“Product code”	
“Major revision”	
“Minor revision”	

1.4.2.4.4 Command 'EtherNet/IP - Scan Devices'

Function: The command establishes a brief connection to the hardware and determines the devices in the network. Then you can apply the devices found into the device tree of your project.

Call: Menu bar: *“Project”*; context menu of a device object in the device tree

Requirement: The communication settings to the controller are correct. The gateway and the PLC are started. The device supports the scan function.

The following devices provide the scan function: EtherCAT master, EtherNet/IP Scanner (IEC), Sercos master, CANopen Manager, CANopen Manager SIL2, PROFINET controller und PROFIBUS DP master.

You can perform the device scan immediately if the scan function is permanently implemented in the PLC. When scan function is implemented in a library, you have to log in only one time to download the library to the controller.

The command refers to the master controller selected in the device tree. For example, an already inserted PROFINET IO controller can be selected and the command used to determine the I/O devices and I/O modules assigned to it.

After performing the scan operation, the *“Scan Devices”* dialog opens and displays the found devices.

Dialog 'Scan Devices'

Table 259: “Scanned Devices”

“Device name, Device type, Address, Station name, etc.”	Data about the scanned device depending on network type. When you change a value in the list of scanned devices, the value is shown in italics. This indicates that the new value has been changed in the editor in CODESYS, but not in the device. When you download the value to the device, it is shown normally. Value that indicate differences between the project and the scanned device are shown in orange. If multiple device descriptions are available for the scanned device, then the name is displayed in bold. The selection of the matching device description is resolved differently for different fieldbuses. For more detailed information, see the corresponding fieldbus chapters. If a device description cannot be found, then the following message is shown: "Attention! The device was not found in the repository." Depending on the bus system, additional information is displayed, such as manufacturer number and product number. The device cannot be inserted into the project without the installed device description.
“Show differences to project”	☑: The table in the dialog also shows additional configured devices (in the device tree of the project). ☐: The table shows all scanned devices. The configured devices are not shown.
“Scan for Devices”	Starts a new search.
“Copy All Devices to Project”	The device that is selected in the table is inserted into the device tree in the project. If nothing is selected, then all scanned devices are shown.

NOTICE!
If you insert devices, which are available in the device tree, to the device tree with *“Copy All Devices to Project”*, then the following should be noted. The data of the *“Process Data”* and *“<...> I/O Mapping”* tabs of the existing devices can be overwritten with the data of the recently inserted devices.

Table 260: “Configured Devices”

This part of the dialog is visible only when you select the *“Show differences to project”* option. Differences between the scanned and configured devices are color-coded. Devices displayed in green are identical on both sides. Devices displayed in red are available only in the view of the scanned or configured devices.	
	If you have selected a device in both views, then the scanned devices are inserted above the selected configured device.
	If you have selected a device in both views, then the scanned devices are inserted below the selected configured device.
	If you have selected a device in both views, then the configured devices are replaced by the selected scanned device.
	All scanned devices are copied to the project.
	Deletes the selected configure device.

Scanning an adapter can fail if the PLC is in RUN mode and a connection already exists from the scanning controller to the adapter. Then the scanning causes another connection to be established to the adapter, which interrupts the existing connection in some adapters. Then the scanner restarts the connection to the adapter, which causes the adapter to interrupt the connection to the scanning controller.

For this reason, it makes sense to perform a network scan in STOP mode after a "Reset". If RUN mode cannot be interrupted, then scanning is possible without an projected remote adapter (EtherNet/IP scanners in the device tree only).

For experts

When accepting the remote adapter by means of the *"Copy to Project"* command, the I/O dimensions with which the adapter responded are set for the first "exclusive owner" connection. In order to log all of the detected assembly instances after scanning, the definition `IODRVETHERNETIP_PRINT_SCAN_RESULT` must be set. By default, it is scanned by the instance ID 100–199. This can be adapted by means of the library parameters `ParamScanStartOfInstanceAssem` and `ParamScanLastOfInstanceAssem` from the library `IoDrvEtherNetIP Library`. This might be necessary, for example to scan in another manufacturer-specific range (assembly instance ID ranges).

1.4.3 OPC UA server for AC500 V3 products

1.4.3.1 General

OPC UA server can be added as an object below the Ethernet interfaces ETH1 or ETH2.

The user can access the variable interface of the PLC via a client. At the same time, communication can be protected by means of encryption.

The CODESYS OPC UA server supports the following features:

- Browsing of data types and variables
- Standard read/write services
- Notification for value changes: subscription and monitored item services
- Encrypted communication according to "OPC UA standard (profile: Basic256SHA256)"
- Imaging of the IEC application according to "OPC UA Information Model for IEC 61131-3"
- Supported profile: Micro Embedded Device server Profile
- By default, there is no restriction in the number of sessions, monitored items, and subscriptions. The number depends on the performance of the respective platform.
- Sending of events according to the OPC UA standard.

Application example

The application example How to use OPC server V3 - for DA and UA is available to gain a deeper understanding of the OPC UA protocol and to configure AC500 V3 accordingly.

1.4.3.2 Creating a project for OPC UA access

1. Click *"File ➔ New Project ➔ AC500 project"* in Automation Builder 2.1 or newer.
2. Choose a *PLC - AC500 V3* and click *[Add object]*.
3. Right-click on node *ETH1 or ETH2* and *"Add object"*.
4. Choose *OPC UA Server* in the dialog and click *[Add object]*.
5. Declare some variables of different types in the program.

6. Right-click *“Application ➔ Add object”*. Choose *Symbol configuration* and click *[Add object]*.
7. Enable checkbox *Support OPC UA Features* in the dialog *Add symbol configuration*.
8. Double-click *“Symbol configuration”* in the *Devices* tree to open the editor *Symbol configuration*.
9. Click *[Build]*.
   ⇨ The variables are displayed in a tree structure.
10. Activate the variables that you want to publish to an *OPC UA client*. Specify the access rights.
11. Download the project to the PLC.

1.4.3.3 Use node name

1. Double-click node *“OPC_UA_Server”*.
2. Set parameter *Use node name* to TRUE.
3. Double-click node *“PLC_AC500_V3 <...>”*.
4. Click *“Device”* and *“Rename active device...”*
5. Enter new device name in the following dialog and click *[OK]*.

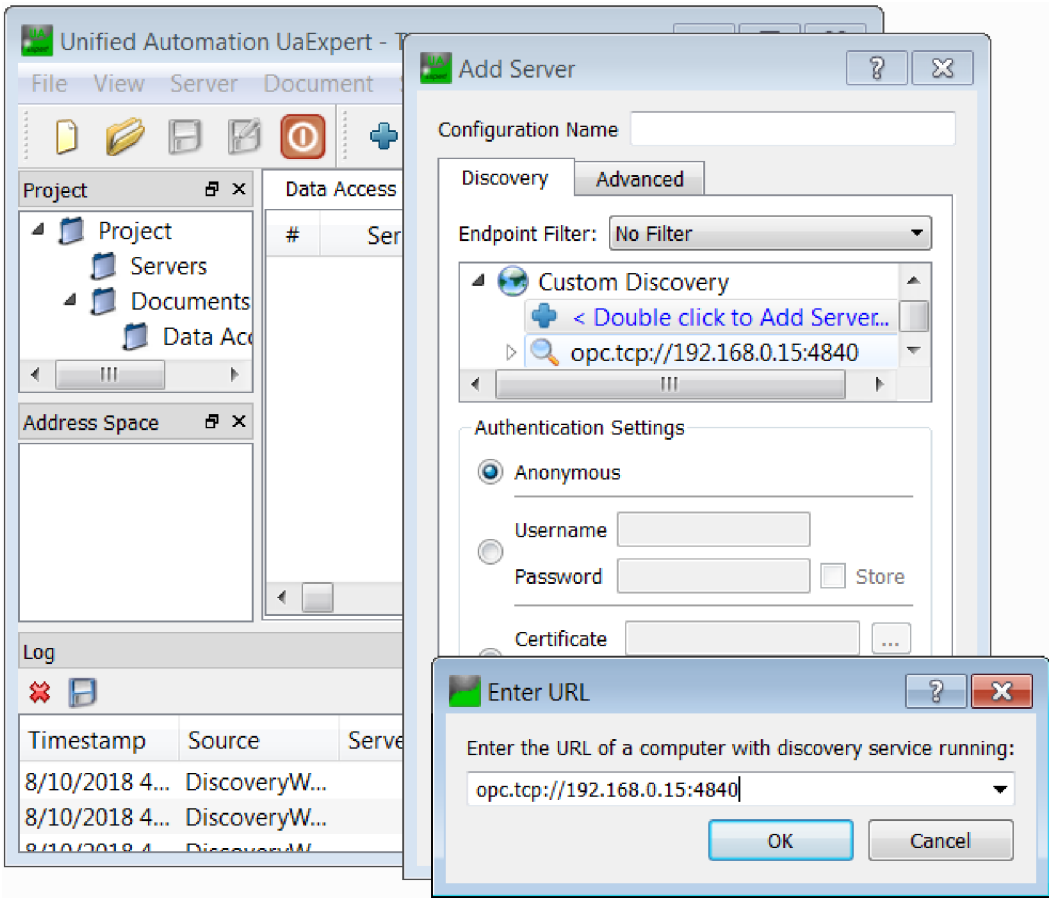
1.4.3.4 Use UaExpert client

The OPC UA client *UaExpert* is available for download from the Unified Automation website and can be used free of charge (freeware license).

Using this client, you can connect to the AC500 OPC UA server.

The following description refers to this program. Other OPC UA clients work in a similar way.

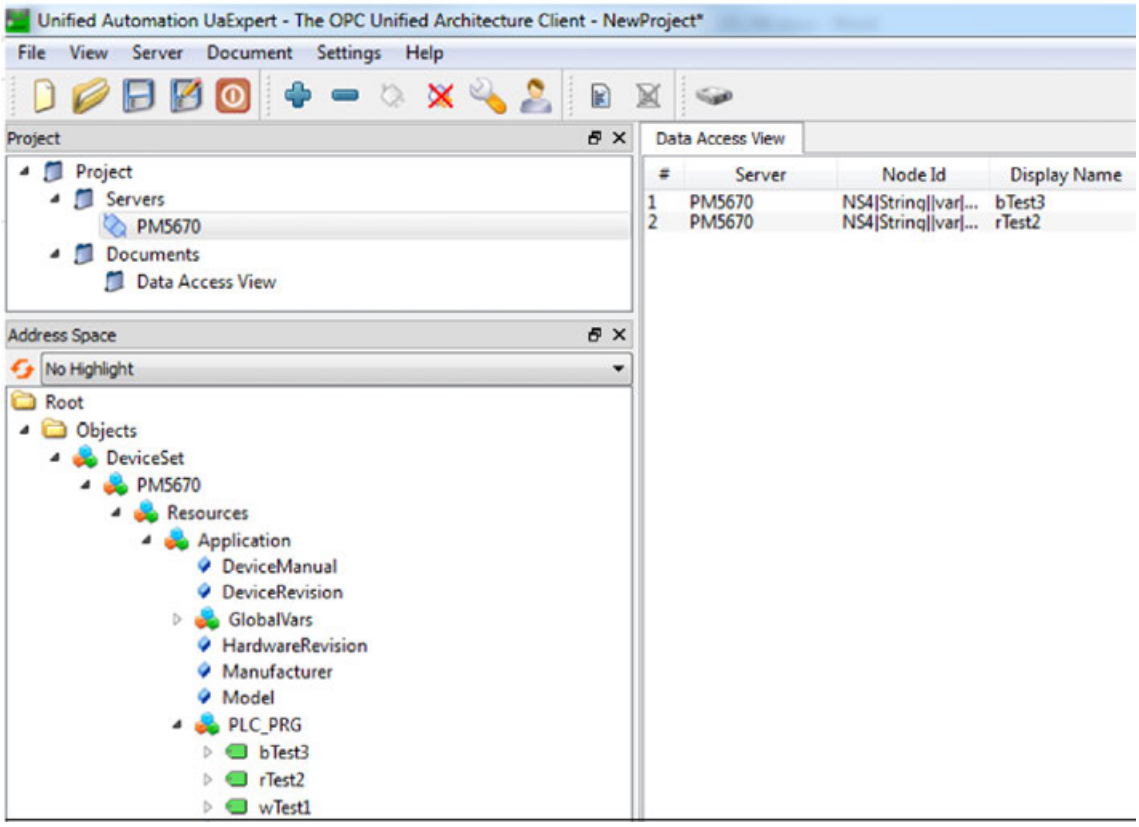
1. Start the *UaExpert* program.



2. Click on the *"blue cross symbol"*.
3. Double-click on the *"blue cross symbol"* in the *Add Server* dialog.
4. Enter URL and click *[OK]*.

   ⇨ The URL appears in the *Add Server* dialog.
5. Select *"Advanced"* tab and click *[OK]*.
6. Click *[Connect]* button.

7. Expand the project tree in the *Address Space* window.



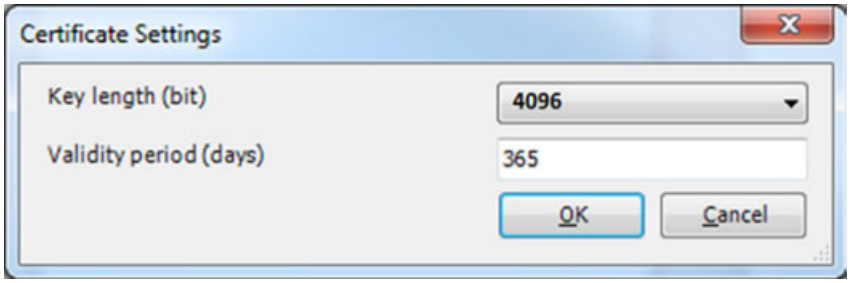
8. Drag and drop the needed symbols to *Data Access View*.

1.4.3.5 Working with encryption

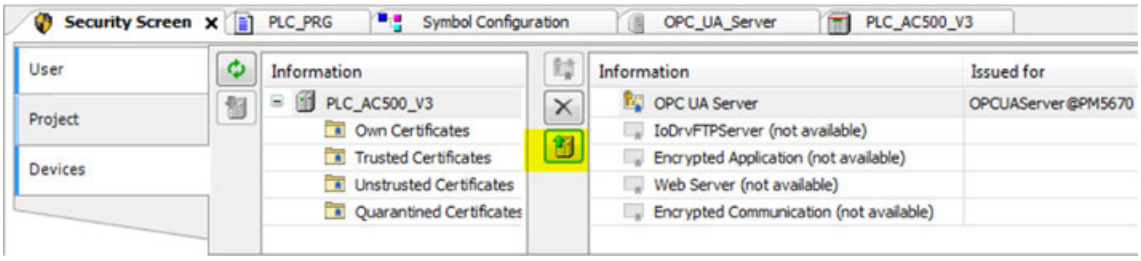
1.4.3.5.1 Creating a certificate for the OPC UA server

☑ Prerequisite: A battery is inserted and the clock is set to actual time.

1. Double-click the Security symbol in the lower right corner of Automation Builder.
2. Select the *"Devices"* tab.
   ⇨ The certificate information opens.
3. Select the PLC in the left *Information* view.
   ⇨ All services of the PLC that require a certificate are displayed in the right *Information* view.
4. Select the service *"OPC UA Server"*.
5. Click the icon to create a new certificate for the device.
   ⇨ *Certificate Settings* dialog appears.



6. Define the certificate parameters according the figure above and click “*[OK]*.
   ⇨ The certificate is created on the PLC.

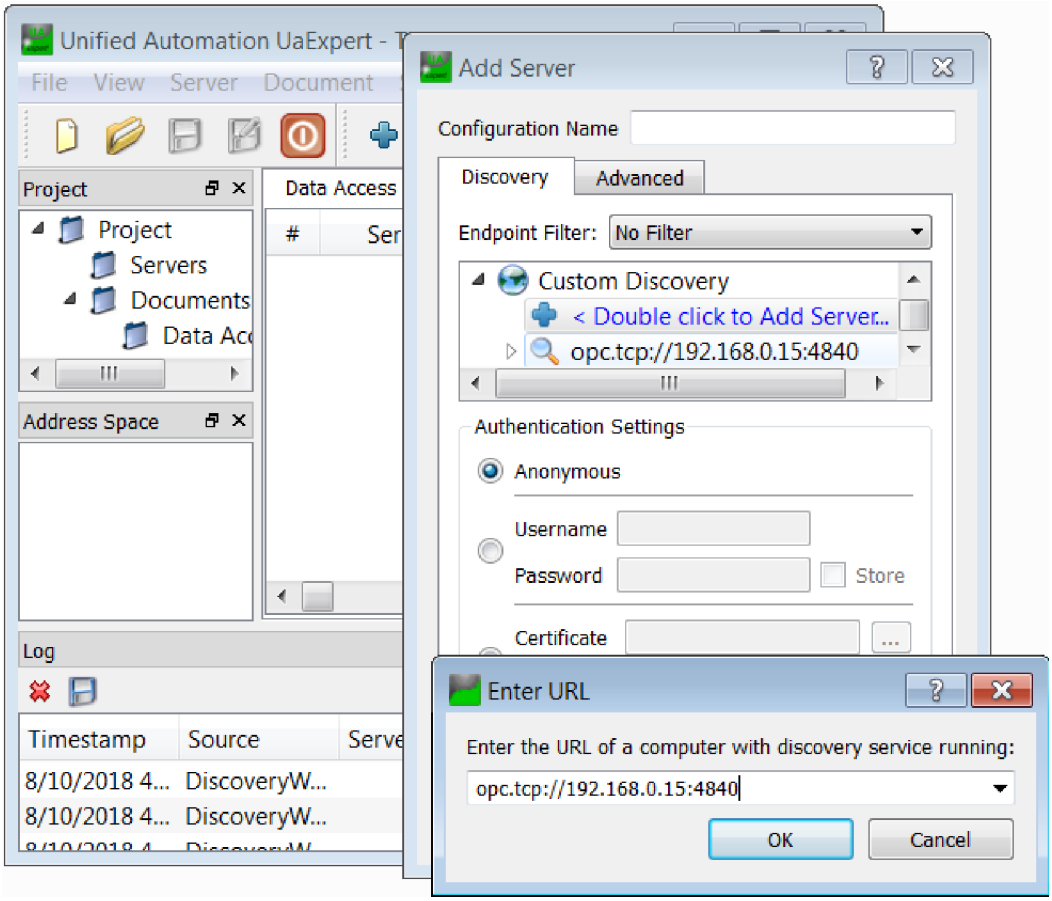


7. Upload the certificate to your PC.
8. Restart the runtime system.

For further information see ↳ *Chapter 1.6.6.3.7.3.4 “OPC UA secure” on page 3923*.

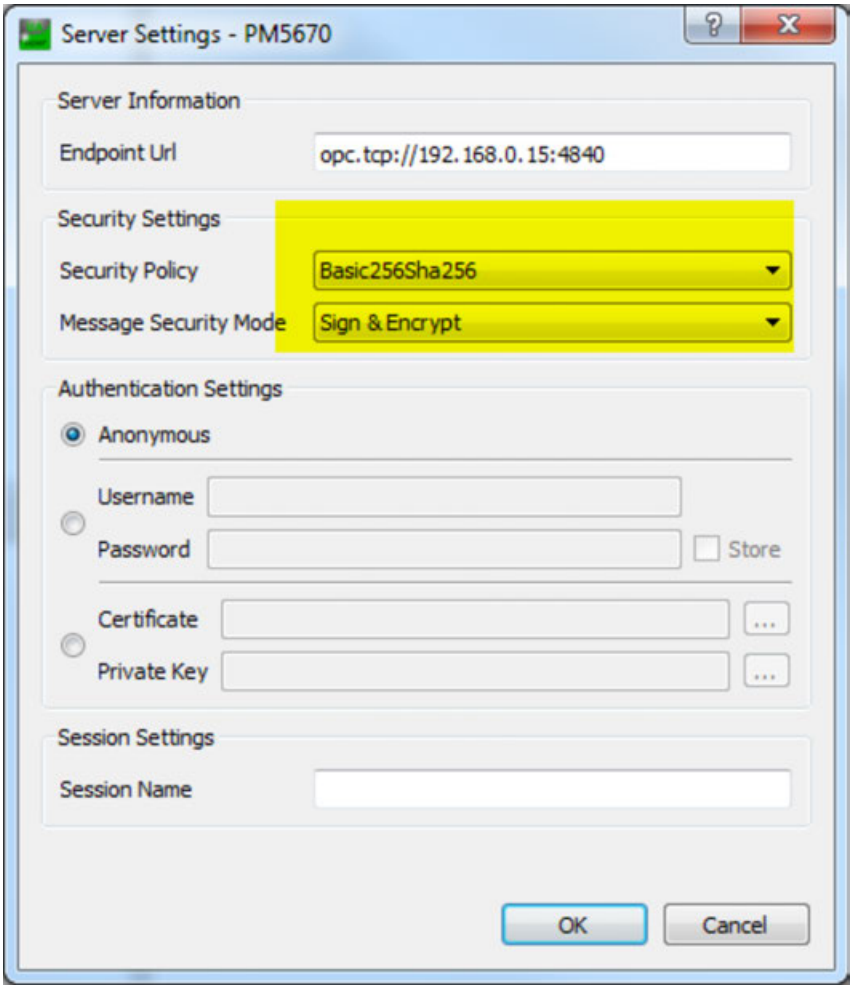
1.4.3.5.2 Encrypted connection with UaExpert client

1. Start the *UaExpert* program.

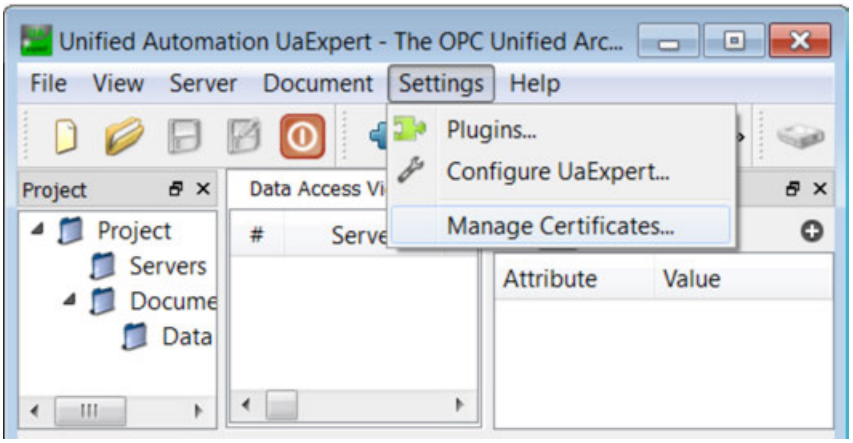


2. Click on the *“blue cross symbol”*.
3. Double-click on the *“blue cross symbol”* in the *Add Server* dialog.
4. Enter URL and click *[OK]*.
   ⇨ The URL appears in the *Add Server* dialog.

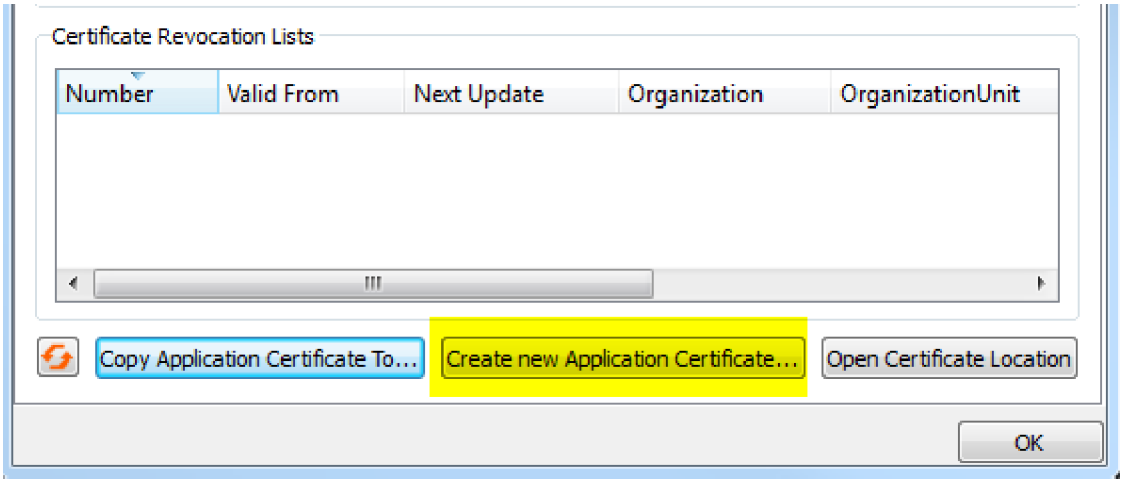
5. Select *"Advanced"* tab.



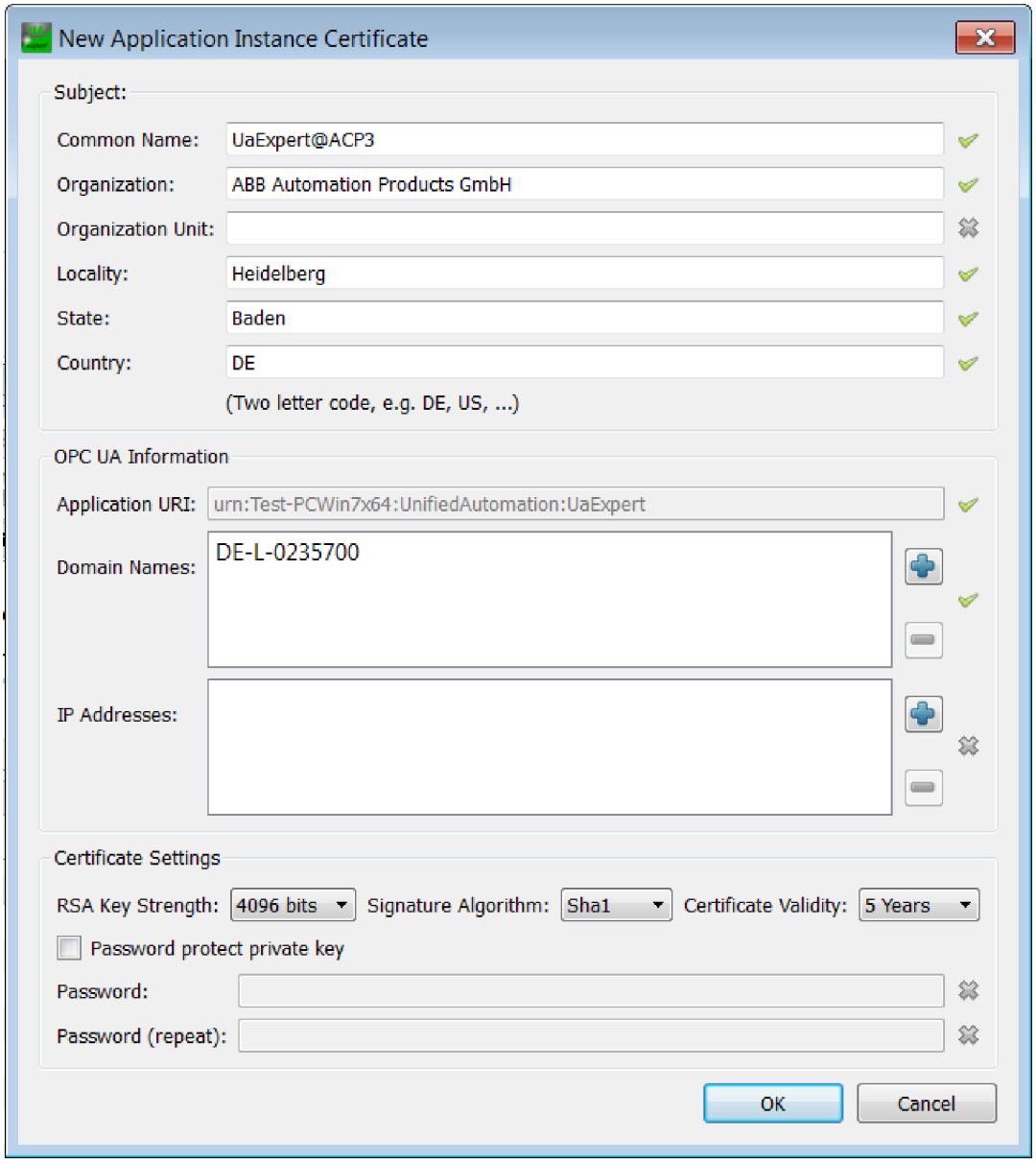
6. Choose option *"Basic256ha256"* of drop-down list *Security Policy* and *"Sign & Encrypt"* of drop-down list *Message Security Mode* and click *[OK]*.



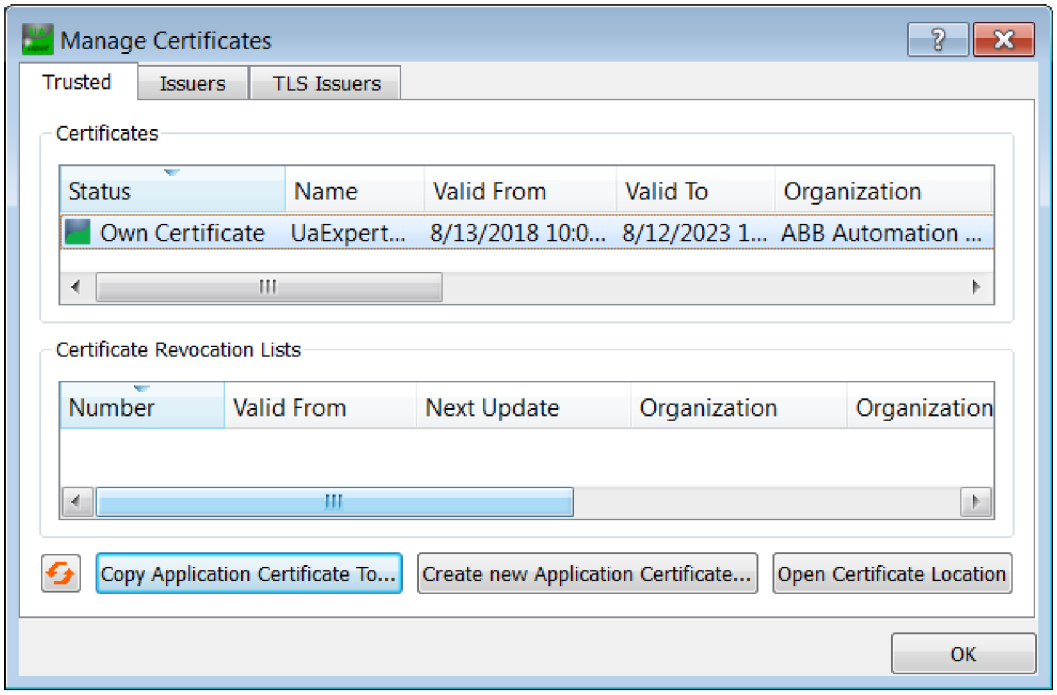
7. Click menu *“Settings”* and *“Manage Certificates”*



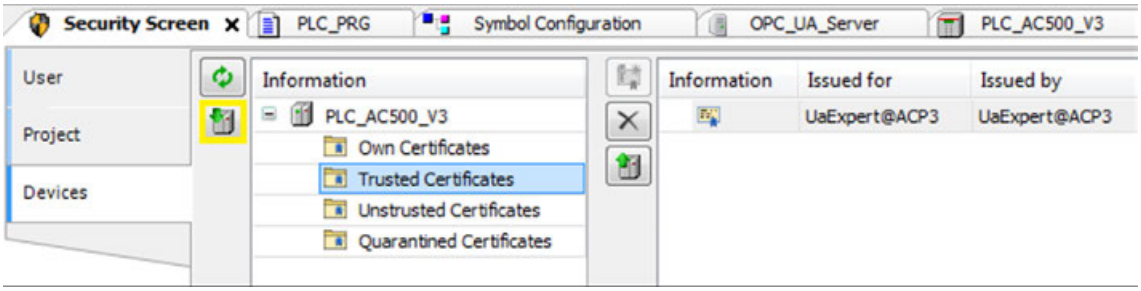
8. Click *[Create new Application Certificate...]*.
   ⇨ Dialog *New Application Instance Certificate* opens.



9. Enter the required informations and click *[OK]*.
   ⇨ Dialog *"Manage Certificates"* opens

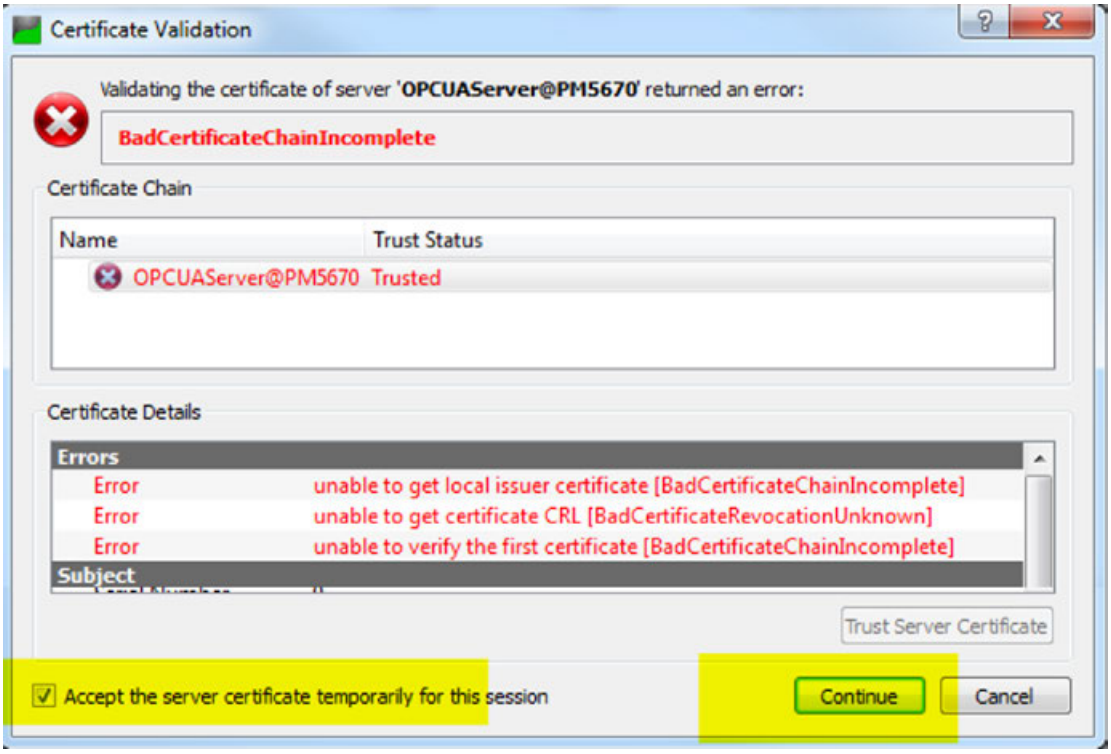


10. Click *[Copy Application Certificate To...]* your PC.



11. Download the certificate to AC500 via the *Security Screen* view.
12. Click *[Connect]* button in the UaExpert client.

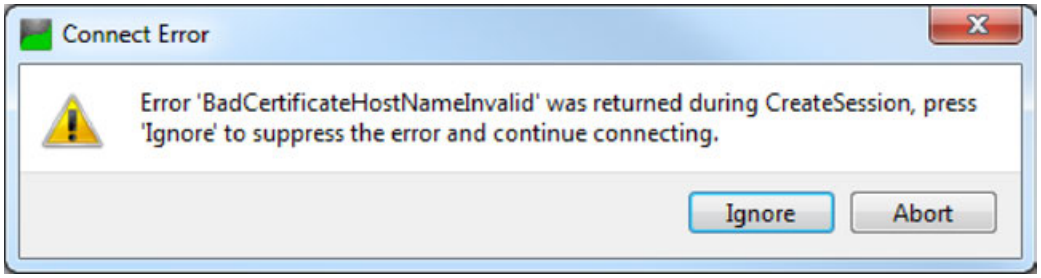
⇨ Dialog *Certificate Validation* opens.



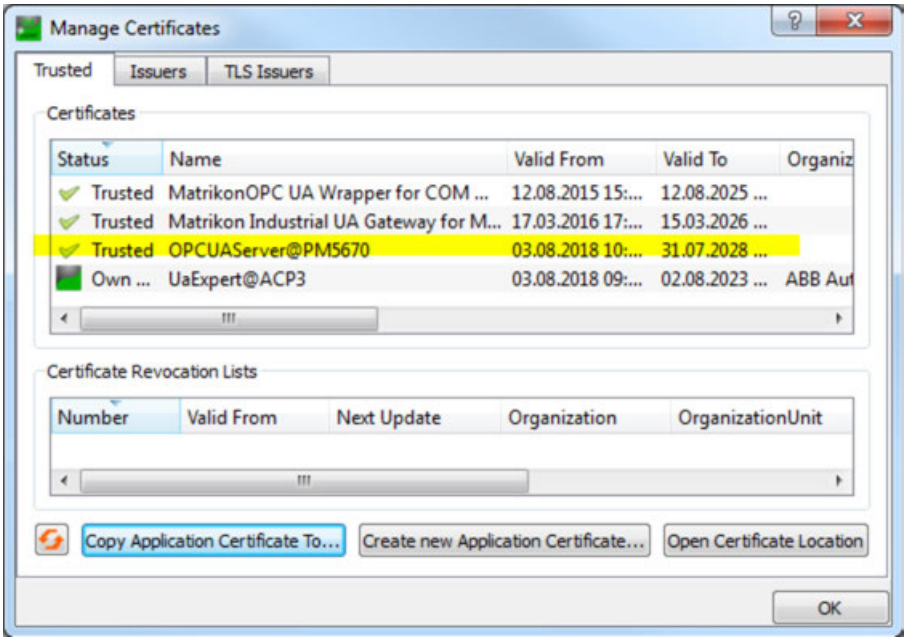
Working with a trusted certificate will avoid this error message.

14. Enable checkbox *Accept the server certificate temporarily for this session* and click *[Continue]*.

⇨ Dialog *Connect Error* opens



15. Click *[Ignore]*

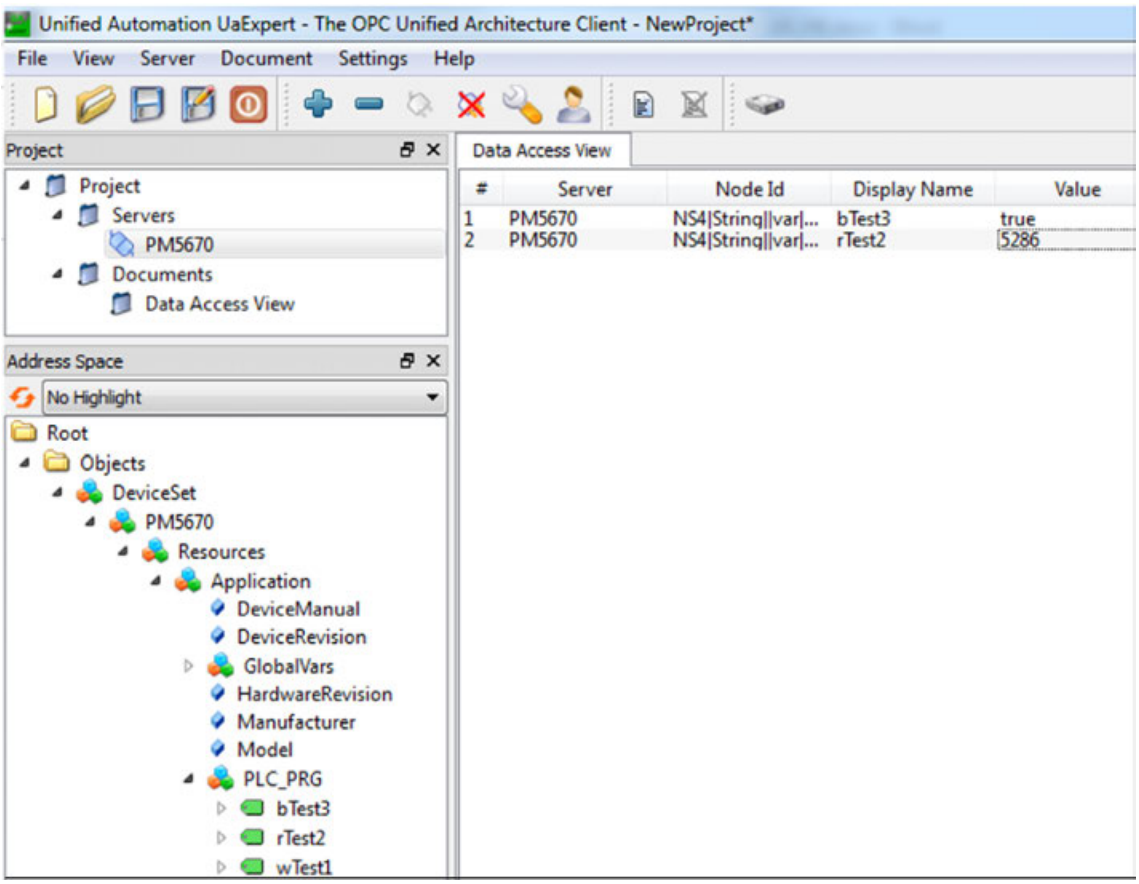


16. Check settings in dialog *Manage Certificates*.

1.4.3.6 Changing variables via UaExpert client

1. Expand in view *Address Space "Objects ➔ DeviceSet ➔ PM5670 ➔ Resources ➔ Application ➔ PLC_PRG"*.

   ⇨ The variables of the global variable list are visible.



2. Drag and drop the variables to the Data Access View.
3. Change values in the column *Value*.

1.4.3.7 Configuring OPC UA client

1.4.3.7.1 Operating modes

Polling

- Objects will be continuously updated in a defined interval
- Create higher load then Subscription
- Is recommended only for a few Symbols

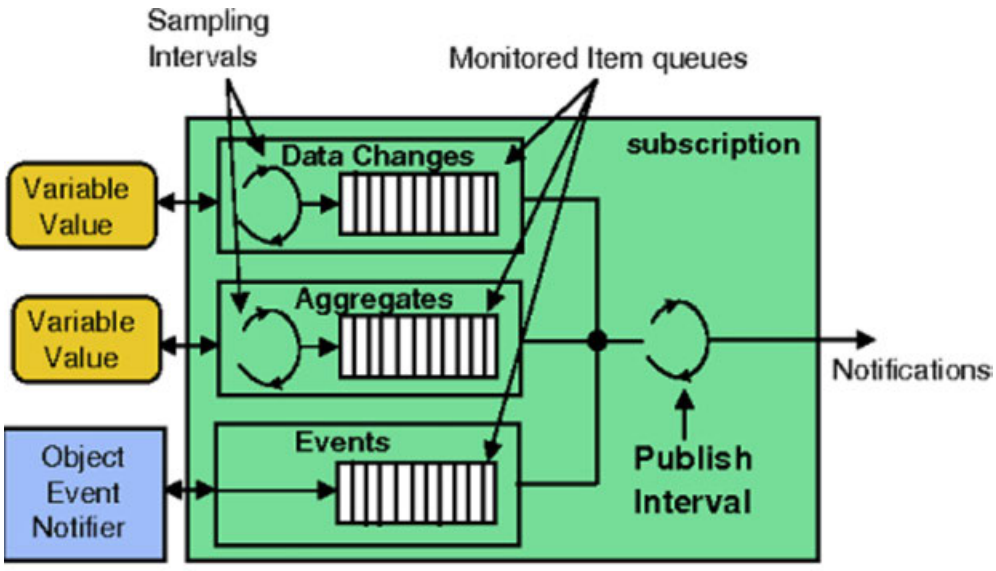
Pub/Sub

Not yet supported

Subscription (recommended mode)

- Updated objects depending on the publishing interval and filters
- Method to reduce load
- Different intervals
- Filter possible (coming in AC500)

Client defines a group of symbols with	Description
Publishing interval	Interval, in which server publish data to client
Sampling interval	Interval for sampling and storing data at server and send in each publishing interval
Queue size	Array of data to save data if sampling Interval is faster than publishing Interval (At AC500 in the moment only 1)
Data change filter	Can be used to reduce traffic from server to client. Criteria: ● Change of data, ● Change of status ● Change of time stamp AC500 is fix configured for change of data and change of status.

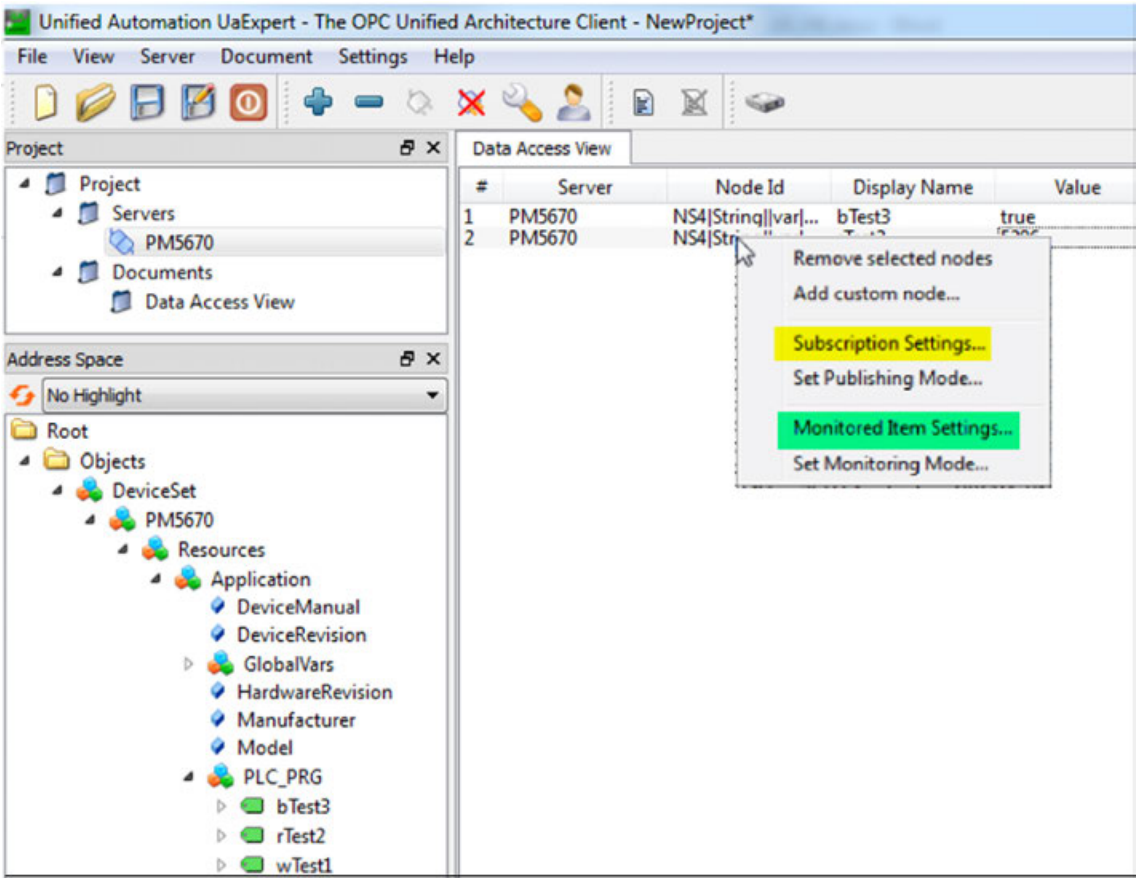


1.4.3.7.2 Using OPC UA with subscription mode

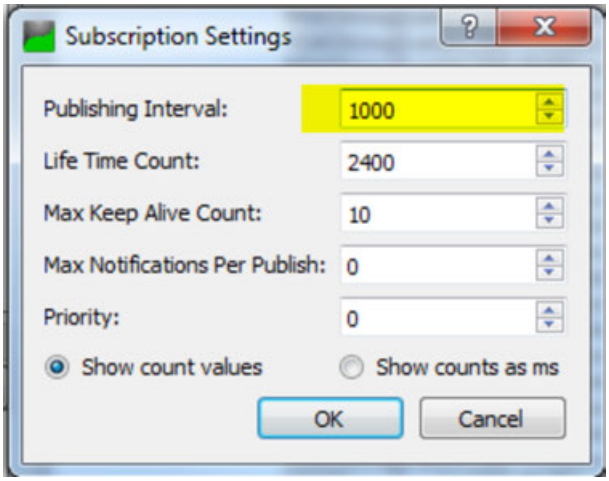
Recommendations:

- *Define only variables you need as symbols*
- *Do not configure publishing Intervals to short (increase load)*
- *Use different subscriptions with different publishing intervals in order to decrease load*
- *Do not use sampling intervals faster then publishing intervals as long as AC500 OPC UA server don‘t support Queue Size different from 1*
- *Be careful: Setting „0“ at sampling Interval at client will be interpreted in server as „as fast as possible“, which is 100ms at AC500 and create a high load.*

Publishing and sampling intervals in UaExpert



1. Right-Click on an Item in *Data Access View* and click *“Subscription Settings”*.



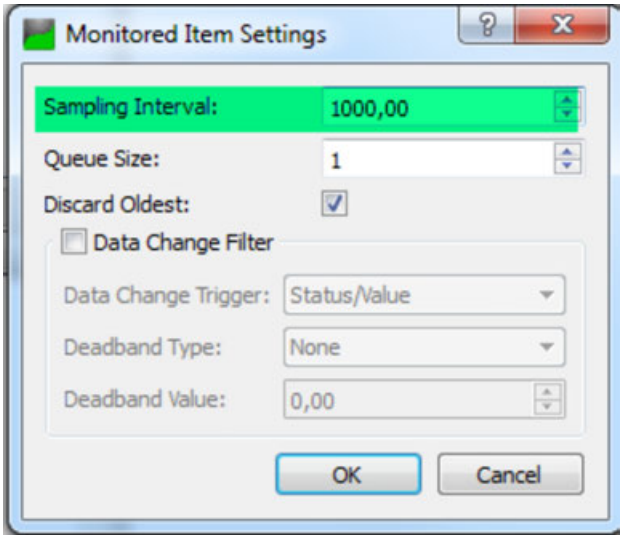
2. Set the recommended values.

   Life Time Count: Number of publishing intervals in which client has to send publish requests to the server. After this period without request from client, subscription in server will be deletet.

   Max Keep Alive Count: If there are no new data to send, server can skip a publishing interval. After the alive count, server has to send, even if there are no new data.

   Click *[OK]*.

3. Right-Click on an Item in *Data Access View* and click *“Monitored Item Settings”*.



4. Set the recommended values.

1.4.4 Libraries

Libraries are used for preparing POUs and functions for use in CODESYS applications. In addition to the descriptions presented here in the help, always see the documentation included in the library as well.

For using libraries in your CODESYS project, see the "Managing Libraries" chapter.

To create your own CODESYS libraries, follow the guidelines for library developers.

See also

- *Chapter 1.4.1.16 “Using Libraries” on page 448*

1.4.4.1 Guidelines for creating libraries

Libraries must be created according to specific rules to avoid compatibility issues.

The main items include the following:

- Select a meaningful library name (required)
- Use templates to ensure consistency (optional)
- Use a familiar and uniform project structure, when possible (optional)
- Register a unique library namespace (required)
- Enter all project information (required)
- Apply the correct method for referencing other libraries correctly (required)
- Design smart external and internal interfaces (required)
- Implement a user-friendly error handling (required)
- Apply the correct method (protection) for deployment (required)
- Apply a consistent naming convention to get clean code (optional)
- When revising an existing library, consider the interface compatibility with previous versions.

Please follow these guidelines when developing libraries in CODESYS: "Library Development Summary". You will find this document as a CHM file (LibDevSummary.chm) in the installation directory of CODESYS, or in the online help.

See also

- *Chapter 1.4.1.16 “Using Libraries” on page 448*

1.4.5 CODESYS Visualization

Everything in one project

In the same CODESYS project, you use CODESYS Visualization to create the suitable user interface for your application. You link the visualization to the application variables and in this way they can animate and display data. When creating a visualization and an application, you use common functions, for example, as library and source code management or find/replace throughout the project.

Overview of functionality

- **Display variant depending on the target platform**
  You can execute the same visualization on various target platforms. Possible display variants are CODESYS WebVisu, CODESYS TargetVisu. In addition, there is a display integrated in the development system.
- **Visualization editor**
  In the graphic editor you design the desired user interface from visualization elements. The visualization elements are provided via libraries in a "ToolBox". You drag them into the editor area and adapt them with the help of a property configurator.
- **Referenceable visualizations**
  A visualization can be referenced in other visualizations. This enables the creation of user interfaces with a complex structure. For this purpose CODESYS Visualization also provides predefined visualizations, e.g. for dialogs.
- **Simple design change**
  The simple change of the look & feel of a visualization is possible in one place by creating a different visualization style.
- **Multilingualism**
  You can conveniently prepare visualization texts in several languages with the help of text lists. You can configure a user input element for switching to a different language in online mode.

- **User management**
  You can set up the visualization's own user management for access control up to individual element level.
- **Other useful features**
  Function block instances of visualizations, array accesses to the visualization, real-time data logging, extendability of the pool of visualization elements, provision of graphic objects via symbol libraries, calls of PLC functions from the visualization, reusability of visualizations by depositing them in libraries.

Table 261: Overview of the objects, editors, repositories, etc. relevant for the visualization in the CODESYS Development System

Visualization	Object below an application in the device tree or in the POUs pool that contains a visualization image. A visualization can reference other visualizations.
Visualization editor and additional views	In this IEC 61131-3-compliant editor you can create the desired graphical user interfaces, panels, dialogs, etc. from visualization elements. The editor is made up of the following components: ● Graphic editor area for arranging the elements ● *"Interface Editor"*: for the parameterization of the visualization ● *"Hotkey Configuration"*: editor for defining keys for online operation ● *"Elementlist"*: overview of all visualization elements used, editor for the position of the elements on the z-axis The following views are also available: ● *"ToolBox"*: view for the provision of visualization elements ● *"Properties"*: view with editor for the configuration of the element that currently has the focus in the graphic editor
Visualization element	Ready-to-use elements from the visualization libraries are available in the Tools view of the visualization editor for insertion.
Visualization profile	The profile defines which visualization elements are available. Each project that contains a visualization is based on such a profile (project settings).
Visualization Styles	The selected style determines the "look & feel" of the elements. It is set application-wide in the visualization manager. Ready-to-use styles are provided and you can also create your own.
Visualization Manager	Each application has a visualization manager of its own for its visualizations with various settings such as user management, style, language, input type, etc. The *"Visualization Manager"* object is suspended in the device tree below the application.
Display variant	A visualization can be displayed in online mode in the following variants, which are created as objects under the visualization manager: ● CODESYS TargetVisu (target visualization and remote target visualization on PLC devices) ● CODESYS WebVisu (web visualization via a web browser) ● Visualization integrated in the development system

Visualization library	Collection of visualization elements that are provided in the toolbox.
Symbol library	Collection of images and graphics that you can use in visualizations. When inserting a visualization object you can choose whether the installed system libraries should be available in the project.
Visualization Element Repository	Repository for the management of the visualization profiles and the visualization element libraries.
Visualization Styles Repository	Repository for the management of visualization styles.
VISU_TASK	This task is automatically present as an object in the task configuration of an application as long as an object for a display variant of the type WebVisu or TargetVisu is also inserted under the Visualization Manager.

System overview and mechanism, display variants

The user interfaces created in CODESYS can be used in different display variants, depending on which ones the controller employed supports.

The display variants

- Visualization ("diagnostic visualization") integrated in the CODESYS Development System:
  The integrated visualization in the development system is ideal for application tests, for service or diagnostic purposes and for the commissioning of a system. As soon as a connection to the controller has been established, the visualization editor switches over and animates the elements displayed. This variant is part of the free CODESYS Development System and can always be used, irrespective of the controller employed.
- CODESYS WebVisu:
  This variant means web-based display of the user interface in a standard browser (PCs, tablets, smartphones), enabling remote access, remote monitoring and service and diagnosis of a system via the Internet. A standard web browser communicates by Java Script (optionally with SSL encryption) with the web server in the controller and displays the visualization by means of HTML5. This technology is supported by virtually all browsers and is thus also available on terminal devices with iOS or Android.
- CODESYS TargetVisu:
  This variant runs independent of the platform on control systems with an integrated display. Logic application and user interface run on the same device; the user interface is displayed directly on the controller. This variant is suitable for the operation and monitoring of machines and plants. An optional extension of the runtime system is required for the use of CODESYS TargetVisu.

1.4.5.1 Preparing CODESYS and projects

The following provides details of the presets that exist for visualizations and the steps that are necessary for creating a visualization in a project.

Presets

When you create a visualization in a project, you should know that the following presets apply:

Scope	Location	Setting
Throughout CODESYS	*“Tools ➔ Options”* Categories *“Visualization”* and *“Visualization styles”*	● Visualization editor: display, handling ● Paths of the basic text and image files ● Visualization styles
Throughout the project	*“Project ➔ Project settings”* Categories *“Visualization”* and *“Visualization profiles”*	● "Properties handling" for the visualization elements ● Paths of the basic text and image files ● Symbol libraries with ready content ● Visualization profile
Throughout the application	*“Visualization Manager”*:	● Unicode, CurrentVisu variable, multitouch, semi-transparency, memory size, data transmission, number of clients ● Visualization styles ● Language setting, language-specific font ● Default keyboard configuration ● Visualizations and visualization references ● Font for each language ● User management
Single visualization	*“Properties”* of the visualization object Category *“Visualization”*	● Purpose and scope of use ● Size definition
Display variant of a single visualization	Editor of the WebVisu or TargetVisu object	● Start visualization, refresh rate, buffer size, html file name ● Scaling options ● Display options ● Default text input

Project-specific updates of the visualization profile, the visualization styles, and the visualization symbol libraries are possible in *“Project ➔ Project environment”* of the respective tabs.

Customization of the visualization menu is performed in *“Tools ➔ Customize”*.

See also

- *Chapter 1.4.5.19.3.9 “Dialog Box 'Options' - 'Visualization'” on page 1763*
- *Chapter 1.4.5.19.3.7 “Dialog 'Options' - 'Visualization Styles'” on page 1761*
- *Chapter 1.4.5.19.3.13 “Dialog 'Project Settings' - 'Visualization'” on page 1766*
- *Chapter 1.4.5.19.3.14 “Dialog ‘Project Settings’ - ‘Visualization Profile’” on page 1767*
- *Chapter 1.4.5.19.3.10 “Dialog 'Project Environment' - 'Visualization Profile'” on page 1764*
- *Chapter 1.4.5.19.3.11 “Dialog 'Project Environment' - 'Visualization Styles'” on page 1765*
- *Chapter 1.4.5.19.3.12 “Dialog 'Project Environment' – 'Visualization Symbols'” on page 1765*
- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Creating visualization objects in the project

For each visualization, you insert a *“Visualization”* object into your project like any other object. This also applies to visualizations that should be used later only within other visualizations. You can insert the new visualization object directly below an application, or below the root node of the *“Devices”* view (for availability throughout the entire project).

The required base libraries and other objects, such as the Visualization Manager, are inserted automatically. When you insert the visualization object below an application, the subordinate objects for the display variants supported by the device are also displayed.

Every visualization object can be edited separately in the visualization editor.

The following steps describe a simply example for creating an object for an application-specific visualization.

Requirement: A project is open. An application is created in the device tree.

1. Select the application in the device tree. Click *"Add object ➔ Visualization"* in the context menu.

   ⇨ The *"Add visualization"* dialog box opens. In the *"Symbol libraries"* table, there is at least the standard entry *"VisuSymbols Vx.x.x. (System)"*, and possibly other installed symbol libraries.

2. Accept the default name `Visualization`. Activate the *"VisuSymbols"* option. Then the visualization symbols (graphical objects) are contained in the library in the visualization project. Click *"Add"* to close the dialog box.

   ⇨ In the device tree, the *"Visualization manager"* and *"Visualization"* objects are inserted below the application. Depending on the device in use, the *"TargetVisu"* and/or *"WebVisu"* objects are also created below the visualization manager.

   If a *"TargetVisu"* object or *"WebVisu"* object is created, then a *"VISU_TASK"* object is also created below the task configuration with an implicit program call.

   The required visualization libraries are added automatically in the *"Library Manager"* of the application.

   The visualization editor opens with the *"Visualization"* editor window and the *"ToolBox"* and *"Properties"* views.

   In the *"ToolBox"* view, there is a *"Symbols"* button for viewing the symbols from the library `VisuSymbols.library`.

3. Now you can create the required visualization in the visualization editor.

4. Note: You can create structured visualizations by using a frame element to reference one visualization in another visualization. Dialog visualizations are a special option for this. In this case, the input configuration of a visualization element is used for referencing.

For creating an application-dependent visualization, insert the visualization object directly below the root node of the device tree. This corresponds to insertion in the "POUs" view. In this case, the visualization manager is not created with objects for the display variants.

See also

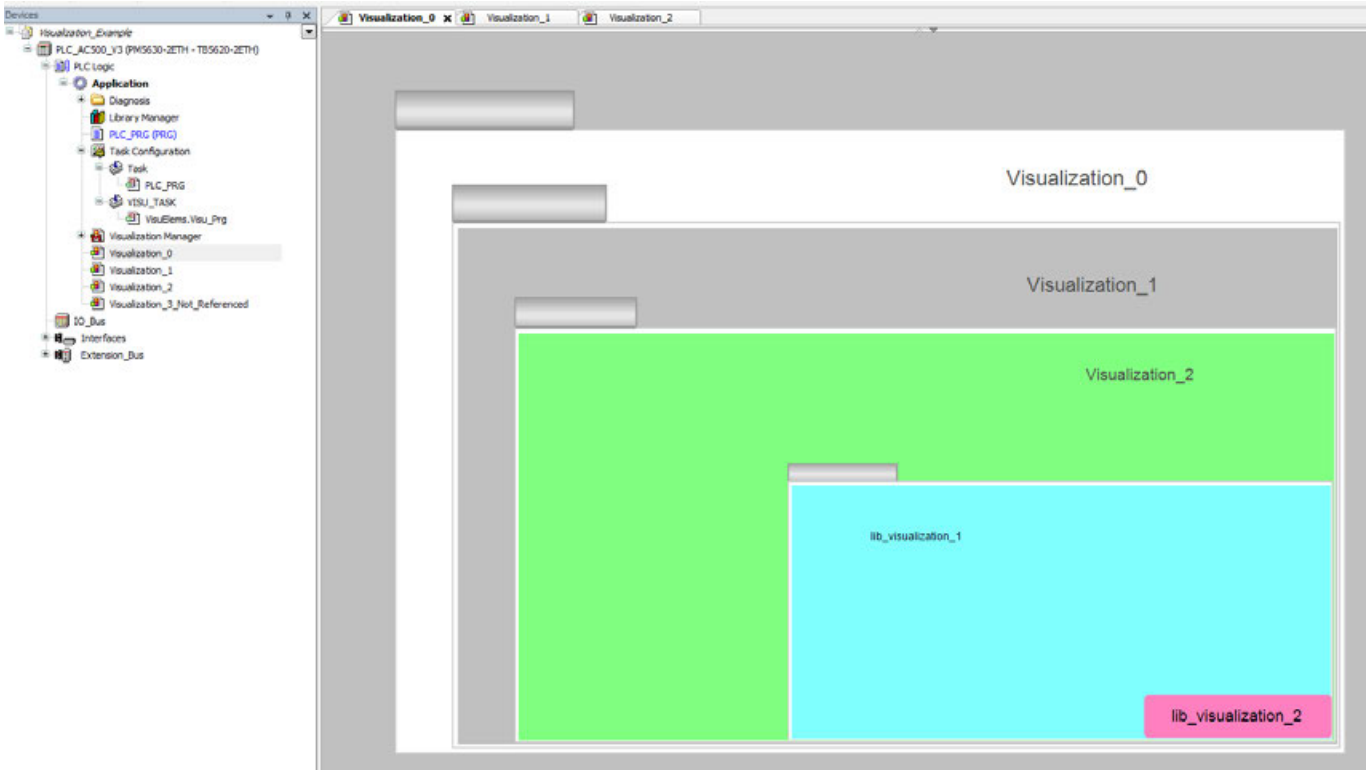
- *Chapter 1.4.5.19.3.7 "Dialog 'Options' - 'Visualization Styles'" on page 1761*
- *Chapter 1.4.5.19.3.13 "Dialog 'Project Settings' - 'Visualization'" on page 1766*
- *Chapter 1.4.5.19.3.15 "Dialog 'Properties' of Visualization Objects" on page 1767*

1.4.5.2 Limitation of the number of usable web pages on AC500 V3 PLCs

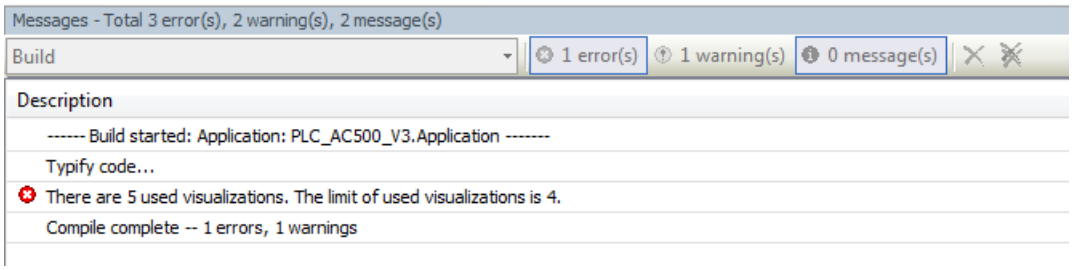
Automation Builder will get all the available visualizations in the project and count those reachable from the visualization client objects (WebVisu and RemoteTargetVisualization).

In case the predefined number of visualizations is exceeded an error is shown in the message window, preventing the user from compiling the project.

The error will be shown under *"Build"* category when the user executes the build command. The PLC program won't download to the PLC until this error is solved (like with any other build errors). In the image below, there are 5 visualizations being used (3 of them added directly into the Automation Builder project and the other 2 referenced from a library that was added to the project).



The error will look like this when build command is executed:



If the visualizations are in the project but not being referenced (e.g. not reachable from the *“Start Visualization”* in the WebVisu) they are not taken into account for this limitation. If the error condition is solved, the error will disappear when the user executes the build command again.

1.4.5.3 Designing a visualization with elements

The visualization editor provides the visualization elements for designing a user interface in the *“Visualization Toolbox”* view.

Drag the desired element into the editor view and adapt it in the *“Properties”* view: purely visual design, labeling, display of data, reaction to user inputs, possibility to input values, etc.

Static or dynamic configuration of the properties is possible. This means the assignment of fixed values or the assignment of application variables. A dynamic configuration allows for an animation which is executed at runtime.

See also

- *Chapter 1.4.5.19.4.1.1 “Visualization Editor” on page 1772*
- *Chapter 1.4.5.8 “Animating visualization elements” on page 1293*

1.4.5.3.1 Select Element

The *“Visualization Toolbox”* view provides the following elements for selection:

- All visualization elements which the set visualization profile defines.
- Image elements for all images of the project from the integrated libraries or symbol libraries.
- Frame elements for all visualizations of the project or from the libraries.

> *See the “Project Settings” for the currently set visualization profile and the currently used symbol libraries.*

The elements are combined into specific categories, each of which has its own button in the *“Visualization Toolbox”* view. You can create new categories and assign its elements.

The elements of the categories are displayed in the *“Visualization Toolbox”* view as preview images. It is also possible to search for an element name.

Simply drag the preview image of the element to the desired position in the editor window. Then the configurable properties of the element are displayed automatically in the *“Properties”* view of the visualization editor.

See also

- *Chapter 1.4.5.19.4.1.1 “Visualization Editor” on page 1772*
- *Chapter 1.4.5.19.4.1.2 “View 'Visualization Toolbox'” on page 1773*
- *Chapter 1.4.5.18.1.5 “Visualization Element 'Image'” on page 1418*
- *Chapter 1.4.5.18.1.6 “Visualization Element 'Frame'” on page 1432*

Create or remove new element category

Requirement: The visualization editor is open.

1. In the *“Visualization Toolbox”* view, click the button.
   ⇨ the *“Configure Categories and Items”* dialog opens.
2. In the dialog, click the + symbol to open the *“Add Category”* dialog. Note: Click the − symbol or press the *[Del]* key to delete the definition of a category.
3. In the *“Name”* field, specify a name (example: `tagA`) and click *“OK”* to close the dialog.
   ⇨ In the *“Configure Categories and Items”* dialog, the new custom category `tagA` is inserted below in the tree view. It is provided with the symbol.
4. Click the *“Enable”* option for the new category, and click *“OK”* to close the dialog.
   ⇨ CODESYS adds a *“tagA”* button in the *“Visualization Toolbox”* view. When you click the button, all elements that are assigned to this category are displayed.

See also

- *Chapter 1.4.5.19.3.4 “Dialog 'Configure Categories and Items'” on page 1747*

Assigning a visualization element of an element category

Requirement: The visualization editor is open. You have already created a custom category `tagA`. A button labeled `tagA` is visible in the *“Visualization Toolbox”* view.

1. In the *“Visualization Toolbox”* view, right-click an element to open its context menu.
   ⇨ A context menu opens. It contains the *“Add Item to Category 'tagA'”* and *“Add to Categories”* commands.
2. Click *“Add to Category 'tagA'”* and click *“OK”* to close the dialog.

3. Click the *"tagA"* button.
   ⇨ All elements are displayed which are assigned to this category, below it also the currently assigned element.

See also

- *Chapter 1.4.5.19.4.1.2 "View 'Visualization Toolbox'" on page 1773*

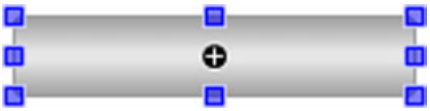
1.4.5.3.2 Positioning the Element, Adapting Size and Layer

A visualization is a raster image in pixels. The pixel position is specified in X/Y-coordinates. The origin (0,0) is located at the upper left corner of the window. The positive X-values run to the right, and the positive Y-values run downwards. The position of an element on the Z-axis of the visualization is controlled by the position in the element list (see below).

Configuring the size and position in the editor

The size and position of an element are specified as pixel coordinates in the *"Properties"* view. These settings are displayed graphically in the editor view at the same time.

When you drag a visualization element from the *"Visualization Toolbox"* view to the editor view, it is shown as selected, as in the following example of a rectangle element:



The possible positions depend on the set grid. You can change its settings CODESYS options. Commands in the context menu are available for alignment and grouping.

Now you can move or resize the element directly in the editor. As an alternative, you configure the *"Position"* property in the properties editor, which opens automatically for the selected element. See the description for this, for example in the help page for the *"Button"* element. The changes are also updated in the other editor.

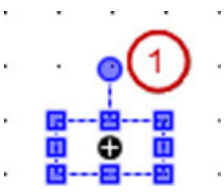
Changing the element size and position in the editor

1. Focus the element so that the shape of the mouse pointer indicates movement (example:).
2. Drag the element to any position.
   ⇨ The position of the element is also updated in the properties *"Position ➔ X"* and *"Position ➔ Y "*.
3. Focus on a blue box.
   ⇨ The shape of the mouse pointer is a double arrow that indicates the direction you can drag the box in order to resize the box: .
4. Drag the blue box to resize the element.
   ⇨ The position of the element is also updated in the properties *"Position ➔ X"* and *"Position ➔ Y "*.

Moreover, you can rotate the *"Rectangle"*, *"Line"*, *"Polygon"*, and *"Pie"* elements.

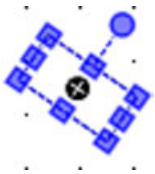
Static rotation of rectangle, line, polygon, pie, or image

1. Select the element for static rotation. Example: Rectangle
   ⇨ The rectangle is displayed with a handle next to the movable position boxes.



(1) Handle

2. Drag the mouse pointer over the handle.
   ⇨ The cursor is displayed as a rotating arrow .
3. Rotate the element to any position.



   ⇨ In the property *"Position ➔ Angle"*, the set angle is displayed in degrees.

See also

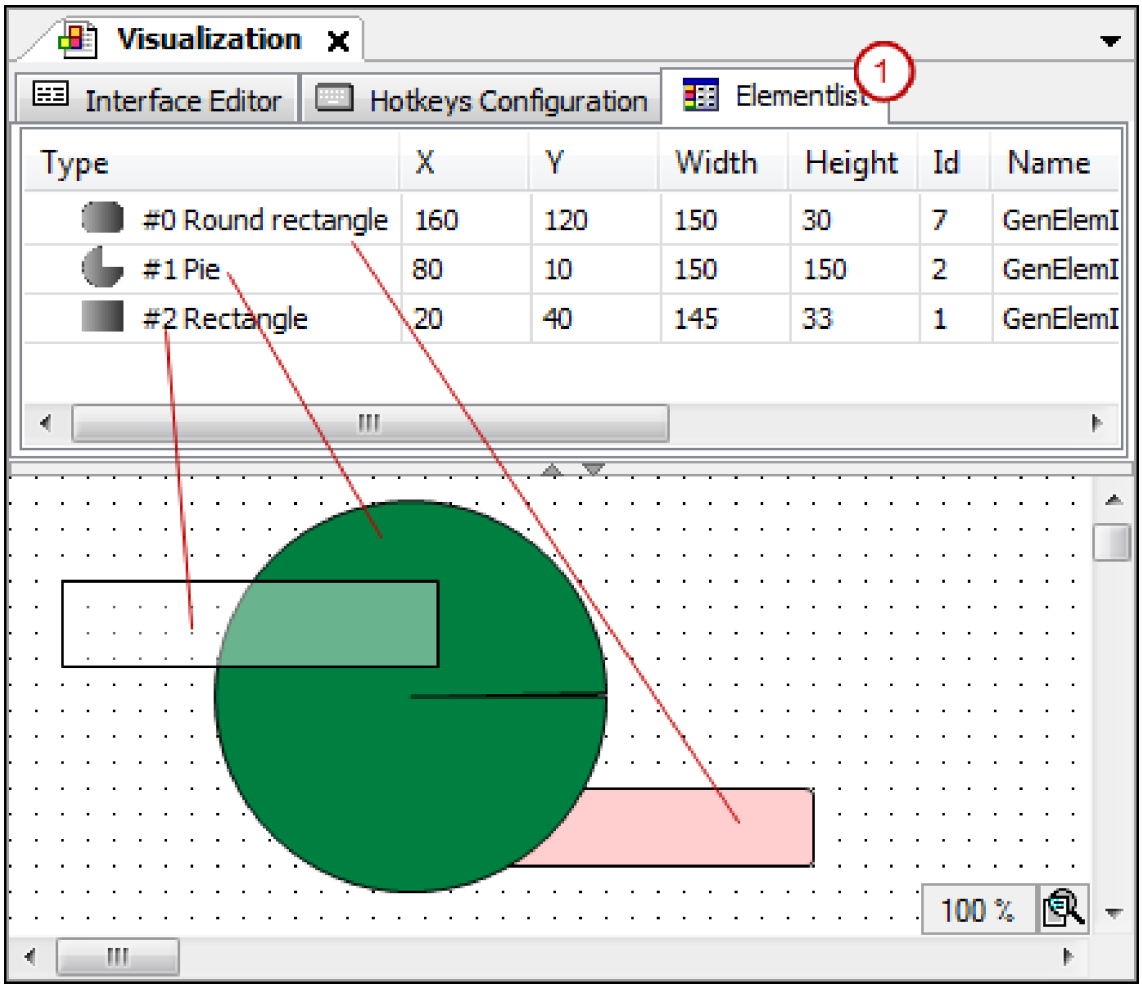
- *Chapter 1.4.5.8.1 "Configuring rotations and offsets" on page 1293*

Moving the visualization element forward and back

Each visualization element is in its own layer of the visualization (Z-axis). It can be hidden by other elements in the foreground and hide other elements in the background. The order of layers is visible on the *"Element List"* tab above the editor view. The order of elements from front to back specifies the order of visualization layers from back to front.

Use the commands from the *"Order"* context menu to move a selected element.

Example of an element list (1):



See also

- *Chapter 1.4.5.19.4.1.1 “Visualization Editor” on page 1772*
- *Chapter 1.4.5.18.1.11 “Visualization Element 'Button'” on page 1468*
- *Chapter 1.4.5.19.2.3 “Command 'Visualization Element List'” on page 1721*
- *Chapter 1.4.5.19.2.5 “Command 'Order'” on page 1723*
- *Chapter 1.4.5.19.2.6 “Command 'Alignment'” on page 1723*
- *Chapter 1.4.5.19.2.7 “Command 'Group'” on page 1726*

1.4.5.3.3 Assigning a color

You configure the color of a visualization element either statically by means of the *“Color”* property, or dynamically by assigning an application variable by means of the *“Color variables”* property. Depending on the element, color assignments are also available in other properties. For example, for the font color, this is provided in the *“Text”* property of a labeled element.

For the static assignment of a color value, you can always use the color dialog in the properties editor, which provides color palettes to choose from.

You can specify the color as a style color. Style colors are color names for color definitions from the actively applied style. When configuring an corresponding property, you are provided with a list of available style colors. **We recommend that you use style colors** because then you can change colors centrally by means of a style selection or a style customization. You can also open the *“Color”* dialog to select a value from color palettes.

In addition, you can define the fill color of an element as a *“Gradient”*. Then the color changes linearly, radially, or axially from the initial color to the final color. You configure the *“Gradient setting”* in the *“Gradient Editor”* dialog.

See also

- *“Element property 'Colors'” on page 1369*

Designing a visualization element with a style color or a fixed color value

> **NOTICE!**
> A color assignment with style color allows for easy global color changes.

☑ Requirement: The visualization editor is open.

1. Insert some `Rectangle` elements.
2. Select an element.
   ⇨ The *“Properties”* view is active.
3. Click in the *“Colors ➔ Normal state ➔ Fill color”* property.
   ⇨ A list box and the [...] button appear.
4. Assign a **style color** to the rectangle. For example, select *“Elementfillcolor”* from the list box.
5. Define the degree of transparency in the *“Colors ➔ Normal state ➔ Fill color ➔ Transparency”* property. Use the slider to select the value *“136”*.
6. Select another rectangle. Click in the *“Colors ➔ Normal state ➔ Fill color”* property.
   ⇨ A list box and the [...] button appear.
7. Assign a **fixed color value** to the rectangle. Click [...] to do this.
   ⇨ The *“Color”* dialog opens.

8. Select a standard color or *“Define Custom Colors”* to fine-tune your selection. Then click *“OK”*.
   ⇨ The color is set as a fixed value. The color is displayed as a small rectangle. The RGB values are also indicated next to it.
9. Click in the *“Colors ➔ Normal state ➔ Fill color ➔ Transparency”* property.
10. Use the slider to select the value *“136”*.
    ⇨ The color is semitransparent.

See also

- ↳ *Chapter 1.4.5.17 “Applying Visualization Styles” on page 1360*

Designing a visualization element with a color gradient

☑ Requirement: The visualization editor is open.

1. Drag a *“Rectangle”* element to the visualization.
2. Select the *“Colors ➔ Use gradient color”* property.
3. Click in the *“Colors ➔ Gradient setting”* property.
   ⇨ The *“Gradient Editor”* dialog opens.
4. Define the color gradient for the element:
   - *“Gradient type”*: *“Radial”*
   - *“Standard radial”*: *“Center”*
   ⇨ The fill color of the element changes radially from white to black.

See also

- ↳ *Chapter 1.4.5.19.3.5 “Dialog 'Gradient Editor'” on page 1748*

Configuring a visualization element for color animation

The *“Color variables”* property, which certain elements may have, is used for the color animation of the element. If you assign a variable there, then you can program color changes in the application code or configure a user input that results in a color change.

You can see an example in the "Animating Visualization Elements" chapter.

See also

- ↳ *Chapter 1.4.5.8 “Animating visualization elements” on page 1293*
- ↳ *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*
- ↳ *Chapter 1.4.5.17 “Applying Visualization Styles” on page 1360*
- ↳ *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

1.4.5.3.4 Using texts

You can get displayed text in an element by assigning a string in the element property *"Texts ➔ Text"*. For example all base elements have this property. Also, you can get displayed a text as a tooltip (element property *"Texts ➔ Tooltip"*). Texts assigned in this way are static. They are managed in the object *"GlobalTextList"* in view *"POUs"* and they cannot be modified during runtime, neither programmatically nor via an user input.

However, you can extend a static text by (exactly) 1 placeholder containing a formatting specification, in order to output the content of a variable at this place. At runtime the current value of the variable, which you have assigned to the element via property *"Textvariable"*, will be output.

> *On the possible formatting specifications please see: Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

By dynamic configuration you can animate the optical representation of the text.

You can localize the static texts, if you have set up multilingualism in your project.

See some examples for the text configuration of visualization elements in the following chapters.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *Chapter 1.4.5.6 "Setting Up Multiple Languages" on page 1286*

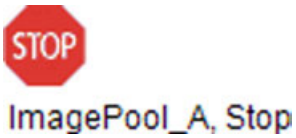
Labeling an image element with a static text

☑ Precondition: A project containing a visualization is opened. You have an image file representing a stop symbol.

1. Below the Application object insert an object *"Image Pool"* named `ImagePool_A`.
2. In the image pool `ImagePool_A` add your stop symbol image file with ID `Stop`.

   ⇨ 
3. Open the visualization and from the ToolBox draw an element *"Image"* into the editor.

   ⇨ The input assistant opens. In tab `Category` you see the image pool`ImagePool_A`.
4. Select the image `Stop` and close the dialog with *"OK"*.
5. Configure the property *"Text"* of the image: `ImagePool_A, Stop`
6. Configure the property *"Text properties ➔ Horizontal alignment"*: `Left`.
7. Configure the property *"Text properties ➔ Vertical alignment"*: `Bottom`.



Text output: Element outputs the result of ST code which is executed on a mouse-click

☑ Precondition: A project containing a visualization is opened.

1. Open the visualization and insert a *"Button"* element.
   ⇨ The *"Properties"* view opens for the new element.
2. Configure property *"Text"*: `Number of clicks: %I`
   ⇨ The string contains the placeholder `%I`.
3. In POU `PLC_PRG` of the application declare a type-conform variable: `iClicks : INT;`
4. Configure the property *"Text variable"* of the button element with `PLC_PRG.iClicks`.
   ⇨ At runtime the variable value will be output instead of the placeholder.
5. Below property *"Inputconfiguration"*, in the cell containing the input event `OnMouseClick`, click on *"Configure"*.
6. From the list of possible actions choose `Execute ST-Code`.
7. Enter the code for the action in the editor *"Execute ST-Code"*:
   ```
   PLC_PRG.iClicks := PLC_PRG.iClicks + 1;
   ```
8. Close the dialog with *"OK"*.
   ⇨ The user input is configured.
9. Build, download and start the application.
   ⇨ The application is running. The visualization opens. The element is labeled and the number of clicks will be output. If you as user click on the button, the number will be increased.

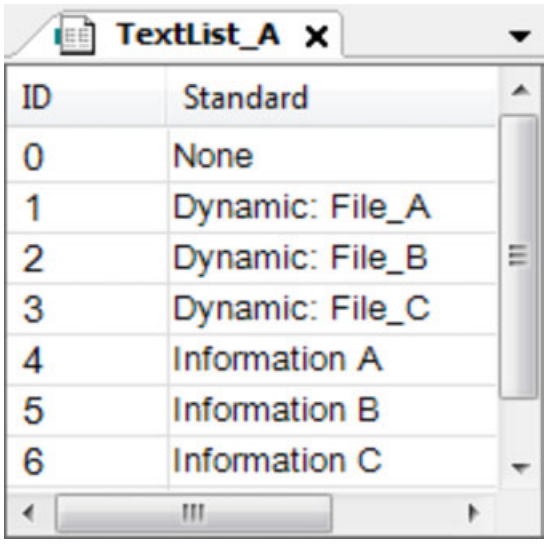
Text output: Dynamic output using a textlist

Using the *"Text field"* element you can produce a dynamic text output. The text output can be effected via an user input or via the application program.

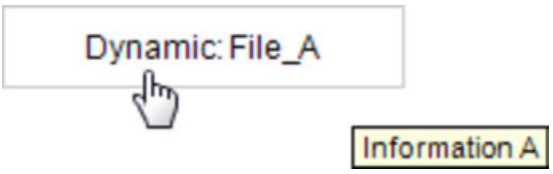
☑ Precondition: A project containing a visualization is opened.

1. Open the visualization and insert a *"Text field"* element.
   ⇨ The *"Properties"* view shows the configuration of the element.

2. Below the application add a Text List with the following entries: `Textlist_A`.



3. In POU `PLC_PRG` of the application declare the text variable:`strTextID : STRING := '0';`
4. Also declare the variable `strTooltipID : STRING := '0';`
5. Also declare the variable `iText : INT;`
6. Configure the property *"Dynamic texts ➔ Text List"* with `'Textlist_A'`.
7. Configure the property *"Dynamic texts ➔ Text index"* with `PLC_PRG.strTextID`.
8. Configure the property *"Dynamic texts ➔ Tooltip index"* with `PLC_PRG.strTooltipID`.
9. In POU `PLC_PRG` implement the CASE instruction as shown below.
   ⇨ The variables in property *"Dynamic Texts"* are programmed.
10. Configure the property *"Inputconfiguration ➔ OnMouseclick"* for `Execute ST-Code` with `PLC_PRG.iText := (PLC_PRG.iText + 1) MOD 4;`
    ⇨ For element *"Text field"* an user input is configured.
11. Build, download and start the application.
    ⇨ The application is running. The visualization opens. In the text field the text `None` is output. When you as user click on the element, the text changes to `Dynamic_File_A`. And the matching tooltip is available: `Information A`. With each click the text changes according to the CASE instruction.



CASE instruction

```
CASE iText OF
0:    strTextID := '0';
    strToolTipID := '0';

1:    strTextID := '1';
    strToolTipID := '4';

2:    strTextID := '2';
    strToolTipID := '5';

3:    strTextID := '3';
```

```
    strToolTipID := '6';
ELSE
    strTextID := '0';
    strToolTipID := '0';
END_CASE;
```

Text output: Configuring a static + dynamic text output

In property *"Texts ➔ Text"* you can define a text in order to get a static text output. A text in *"Texts ➔ Tooltip"* will be displayed as tooltip. You can configure the text in a way, that the content of a variable is additionally output.

You can extend a static text by (exactly) 1 placeholder including a formatting definition, in order to output the content of a variable at this place at runtime. The variable must be assigned in property *"Text variable"*. When the variable value changes in the application code, then at the same time the output in the visualization changes.

☑ Precondition: A project containing a visualization is opened.

1. Open the visualization and insert an element *"Text field"*.
   ⇨ The *"Properties"* view shows the element configuration.
2. Configure the property *"Texte ➔ Text"*: `File name: %s`
   ⇨ The text contains the placeholder `%s`.
3. In POU `PLC_PRG` of the application declare a type-conform variable `strFileName : STRING := 'File_A';`
4. Configure the property *"Text variable"* of the text field with `PLC_PRG.strFileName`.
   ⇨ At runtime the variable value will be output instead of the placeholder.
5. Build, download and start the application.
   ⇨ The application is running. The visualization opens. The text field element displays the text: `File name: File_A`

See also

- ↳ *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Configuring text input in a text field

You can use the *"Text field"* element in order to output the text given by a variable, or to provide a place, where the user can give input on the variable.

Additionally you can configure a text input. In this case on an user input an input field in the element *"Text field"* will appear. As a precondition you must have configured an user input action in the property *"Inputconfiguration"*.

See also

- ↳ *Chapter 1.4.5.6 "Setting Up Multiple Languages" on page 1286*

Showing text as a tooltip

☑ Precondition: A project containing a visualization is opened.

1. Open the visualization and insert an element *"Text field"*.
   ⇨ The *"Properties"* view shows the element configuration.
2. Configure the property *"Texts ➔ Text"*: `File name: %s`
   ⇨ The text contains the placeholder `%s`
3. Configure the property *"Texts ➔ Tooltip"*: `Storage location: %s`
4. In POU `PLC_PRG` of the application declare a type-conform variable `strFileName : STRING := 'File_A';`

5. In POU `PLC_PRG` of the application declare also the variable `strFileDir : STRING := 'D:/Data';`
6. Configure the property *"Text variable"* of the text field with `PLC_PRG.strFileName`.
   ⇨ At runtime the variable value will be output instead of the placeholder.
7. Configure the property *"Tooltip"* of the text field with `PLC_PRG.strFileDir`.
8. Build, download and start the application.
   ⇨ The application is running. The visualization opens. The text field element shows the text `File name: File_A`. When the mouse cursor is moved above the text field, the tooltip will be displayed: `Storage location : D/Data`.

See also

- ↳ *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Configuring element *"Text field"* for text input

The user should be able to enter text in a text field. For this configure an input of type *"Write variable"* on a text output variable. This text output variable will store the text input of the user and will display this text instead of the placeholder (this is `%s` in the example below). You specify the text output variable in the property *"Text variables ➔ Text variable"*.

☑ Precondition: A project containing a visualization is opened.

1. In POU `PLC_PRG` of the application declare a string variable: `strInput : STRING;`
2. Open the visualization and insert an element *"Text field"*.
   ⇨ The *"Properties"* view shows the configuration of the element.
3. In property *"Texts ➔ Text"* enter `Input: %s`.
4. In property *"Inputconfiguration"* for mouse action *"OnMouseClick"* click on *"Configure"* to open the *"Input Configuration"* dialog box. There choose action *"Write a Variable"* and activate option *"Use text output variable"*. Close with *"OK"*.
5. In the element property *"Text variables ➔ Text variable"* assign the text output variable `PLC_PRG.strInput`.
6. Build, download and start the application.
   ⇨ The application is running. The visualization opens. The element outputs the text: `Input:`. Click in the element to open an input field, where you can enter a string. After having terminated the input by *[Enter]*, the text will be adopted.

Animating the text display

Configure the property *"Font variables"* in order to animate the display of a text. All base elements have this property, additionally the table, scrollbar and text field element.

See also

- ↳ *Chapter 1.4.5.8.2 "Animating a text display" on page 1295*

Configuring the 'Label' element

☑ Precondition: A project containing a visualization is opened.

1. Open a visualization and insert an element *"Label"*.

   ⇨ The *"Properties"* view with the pre-set property configuration opens.

2. Configure the property *"Texts ➔ Text"*: `Visualization A`.

Making an element invisible

You can configure the property *"State variables ➔ Invisible"* in order to hide an element in the visualization.

☑ Precondition: In the visualization you have configured a text field, which gets visible only, if a certain application variable gets TRUE. For example in order to show certain instructions or descriptions only in a certain state of the machine.

1. For the text field element configure the property *"Texts ➔ Text"* with `Error detected: Do the following....`

   Configure the property *"Text properties ➔ Font color"* with *"dark red"*.

2. In `PLC_PRG` declare the variable `bIsInvisible : BOOL : TRUE;` (this is the initialization for the current example; normally the variable should be set to TRUE by the application program under certain conditions.

3. Configure the property *"State variables ➔ Invisible"* with `PLC_PRG.bIsInvisible`.

4. Build, download and start the application.

   ⇨ The application is running. The visualization opens and the text field is not visible. When you set `bIsInvisible` to `TRUE`, the textfield will be displayed.

1.4.5.3.5 How to display variable values in the visualization

There are simple to very specialized visualization elements for displaying data from a running application.

Examples:

- Simple output of variable values: For example, you can configure a purely formatting specification for a *"Rectangle"* element in the *"Text"* property and the variable whose value is to be displayed in the element in the *"Text variable"* property.
- Display of structured variable values (structure, array, function block): You use the *"Table"* element and specify an array variable in its configuration in the *"Data array"* property whose values are to be displayed in the table. One-dimensional arrays can also be displayed in a *"Histogram"*.
- Display of values by image switching. Example: A specific screen is displayed depending on the error message that occurs. You do this by configuring an *"Image"* element with a variable for the *"Bitmap ID variable"* property.
- Display a variable value as a bar or with a pointer on a scale: You specify a variable in the *"Value"* property of the *"Bar Display"* element or *"Meter"* element to display its value as a bar on a horizontal or circular scale.
- Display of alarms: The alarms configured in the alarm management of the application can be made visible by means of the *"Alarm Table"* and *"Alarm Banner"* elements in the user interface.
- *"Trace"* and *"Trend"*: For graphical recording of variable values over a period of time.

For details, see the descriptions of the element properties.

See also

- *Chapter 1.4.5.18.1 "Visualization Elements" on page 1367*
- *Chapter 1.4.5.3.4 "Using texts" on page 1260*
- *Chapter 1.4.5.21.4 "Displaying Array Data in a Histogram" on page 2138*

1.4.5.3.6 How to Change Variable Values via the Visualization

In addition to displaying values from the controller, a user interface is also used to enter and change values.

In general, you can configure user input for each element in its *“Input configuration”* properties. Moreover, elements have been developed especially for specific input.

Examples:

- A *“Button”* element (or *“Rectangle”* element, and so on) that is clicked to open a predefined dialog visualization for easily specifying a value.
- A *“Slider”* element for changing the value of a variable by moving visual element parts, for example with the mouse. In the case of the slider: The element adjusts the value of a variable, depending on the position of the slider within the slider. You define the value range of the slider bar by means of the scale start and scale end.
- A switch element (example; *“Power Switch”*) for setting a Boolean value.
- A *“Spin Box”* element for incrementing or decrementing the value of a variable in defined intervals.
- A *“Button”* element for writing a recipe, executing a specific ST code, writing a specific variable, and so on (definition in the input configuration).

See also

- *Chapter 1.4.5.19.5 “Visualization Elements” on page 1791*
- *Chapter 1.4.5.4 “Configuring user inputs” on page 1267*

1.4.5.3.7 Designing a background

You can design the background of your visualization in color or with an image. To do this, use the command *“Visualization ➔ Background”*.

See also

- *Chapter 1.4.5.19.2.10 “Command 'Background'” on page 1728*
- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Configuring an image as a background

In addition, you can use the property “Integrate background” in the dialog “Properties” of a visualization object to specify whether the background image should always be displayed in its entirety or whether it should be truncated.

☑ Requirement: A project with a visualization is open.

1. Open the visualization and select the command *“Visualization ➔ Background”*.
2. Activate the option *“Image”* and open the input assistant.
3. Select an image in the dialog *“Input Assistant”*.

   ⇨ The image serves as a background image.

See also

- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Configuring a colored background

☑ Requirement: A project with a visualization is open.

1. Open the visualization and select the command *“Visualization ➔ Background”*.
2. Activate the option *“Color”*.
3. Select a style color such as *“Element background color”* from the selection list.
   ⇨ The background of the visualization is colored.

1.4.5.4 Configuring user inputs

User inputs for a visualization are configured in order to operate the visualization.

For this purpose, you configure input events on visualization elements where follow-up actions are triggered. The combination of user inputs and follow-up actions are defined in the *“Input configuration”* of an element. For example, you can select a mouse click on an element as the input event and opening a dialog box as the input action.

Keyboard events can also be configured that trigger actions in a specific visualization window when the events occur. You program this kind of input configuration for a visualization in its *“Keyboard configuration”* editor.

In addition, keyboard events can be configured that occur in all visualizations programmed in the application. You configure this kind of input configuration per application below the visualization manager in the *“Standard keyboard shortcuts”* tab.

Input is usually performed with the mouse and keyboard as controlling device. You can also configure a user operation by means of gestures.

If a visualization device is not equipped with a mouse, then you can activate default keyboard usage. Then a user can operate the visualization with the keyboard only by navigating with the arrow keys and triggering events by pressing the *[Enter]* key.

If a visualization device is not equipped with a keyboard, then you can call a virtual keyboard or a virtual numeric keypad.

> **NOTICE!**
> Configure keyboard events only for keys that the visualization device supports.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*
- *Chapter 1.4.5.19.4.3 “Tab 'Visualization Manager' - 'Default Hotkeys'” on page 1781*

Processing order of keyboard events

1. Event handler of the application. Requirement: The event handler is activated.
2. Events of the default keyboard usage
3. Events of the keyboard usage are configured in the tab *“Visualization manager” - “Default hotkeys”*.
4. Events of keyboard usage are configured in the tab *“Keyboard configuration”* for the currently visible visualization.

See also

- *Chapter 1.4.5.4.5 “Capturing user input events” on page 1277* *Chapter 1.4.5.19.1 “Keyboard Shortcuts for Default Keyboard Action” on page 1717*
- *“Tab 'Keyboard configuration'” on page 1720* *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.4.3 “Tab 'Visualization Manager' - 'Default Hotkeys'” on page 1781*
- *Chapter 1.4.5.19.4.3 “Tab 'Visualization Manager' - 'Default Hotkeys'” on page 1781*

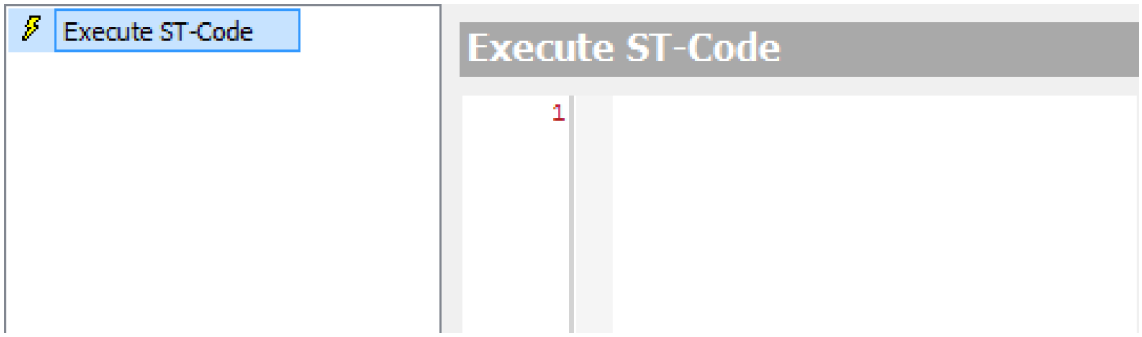
1.4.5.4.1 Configuring user inputs for visualization elements

All base elements and some common control elements have the *“Input configuration”* property. This is where you can configure a user input for an element. For this purpose, you select an input event and an input action.

Configuring user inputs

☑ Requirement: A project is open with a visualization.

1. Open the visualization and added a *“Button”* element.
   ⇨ The *“Properties”* view opens for the new button.
2. Configure the property *“Text”* with `Number of clicks: %i`.
3. Declare a variable `iClicks : INT;` in the application in the `PLC_PRG` POU.
4. Configure the *“Text variable”* property of the button as `PLC_PRG.iClicks`.
   ⇨ At runtime, its variable value will replace the placeholder in the *“Text”* property.
5. In the *“Input configuration”* property, click the *“Configure”* button in the `OnMouseClick` line.
6. Select the `Execute ST code` action from the list of possible actions and click the > symbol.
   ⇨ The action appears in the list of actions to be executed. The blank implementation of the action appears in the window area to the right of the list.



7. Program the action in the editor at *“Execute ST code”*:
   ```
   PLC_PRG.iClicks := PLC_PRG.iClicks + 1;
   ```
8. Click *“OK”* to close the dialog box.
   ⇨ The user input is configured.

9. Compile, download, and start the application.

   ⇨ The application runs. The visualization opens. If the user clicks the button, then the action is executed, the variable `PLC_PRG.iClicks` is incremented, and the number of clicks is printed.

See also

- ↳ *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

1.4.5.4.2 Configuring gesture recognition

You can execute a visualization on a device that is operated by means of gestures. The visualization retains its user input configuration for mouse and keyboard operation and also recognizes gestures and multi-touch events. Gesture events are recognized and interpreted as mouse events.

For this purpose, activate the *“Activate multi-touch”* setting in the visualization manager.

Elements of the type *“Frame”* or *“Tab control element”* display contents that a user should be able to move. Therefore, configure their *“Scaling type”* property with *“Fixed and scrollable”*.

Gesture recognition for:

- Tapping
  A quick tap on the element is interpreted as a mouse click.
- Panning
  Pressing, moving, and releasing with one finger in a frame or with a tab control element (in the window area of the element) will move the contents.
- Multi-finger touch detection
  Touching several elements at the same time will input for all elements. These touch events are interpreted as the respective mouse events.
  Example:
  Two-hand operation in order to trigger an action with two simultaneous inputs on two different elements.
  Virtual mixing console where multiple sliders can be operated at the same time.

In addition, the `IGestureEventHandler` interface is available in the `VisuElems.VisuElemBase` library. You can use this to implement application code that recognizes gestures and executes follow-up actions.

The following display variants can execute a visualization on a multi-touch device

- CODESYS WebVisu

See also

- ↳ *“Implementing event handling with multi-touch” on page 1270*

Using gestures to control visualizations

☑ Requirement: A project is open with a visualization and a user input configuration. It contains one button. The visualization device is a display with multi-touch support.

1. Double-click the *"Visualization manager"* object.
   ⇨ The editor opens.
2. Click the *"Settings"* tab.
3. In the *"Additional settings"* group, activate the *"Activate multi-touch"* option.
4. Compile, download, and start the application.
   ⇨ The application runs. The visualization opens. When a user touches the display of the visualization device, the visualization responds. Elements that respond to mouse events also respond to touch events Several buttons can be pressed at the same time. Scrollable frames or tab control elements are displayed without scrollbars and can be moved by panning.

   Note: The *"Scaling type"* property of elements type *"Frame"* or *"Tab control element"* must be set to *"Fixed and scrollable"*.

See also

- *Chapter 1.4.5.18.1.6 "Visualization Element 'Frame'" on page 1432*
- *Chapter 1.4.5.18.1.10 "Visualization Element 'Tabs'" on page 1463*

Implementing event handling with multi-touch

☑ Requirement: The device is multi-touch capable

1. Implement and register a function block that receives the gesture events.
   ⇨
```
FUNCTION_BLOCK GesturesHandler IMPLEMENTS
VisuElems.VisuElemBase.IGestureEventHandler2

VisuElems.g_VisuEventManager.SetGestureEventHandler(THIS^);
```
2. Implement and register a function block that sets the touch areas.
   ⇨
```
FUNCTION_BLOCK RectProvider IMPLEMENTS
VisuElems.VisuElemBase.IApplicationRectangleProvider

VisuElems.g_VisuRectangleProvider := THIS^;
```
3. Implement actions as application code that are executed when a gesture event occurs

1.4.5.4.3 Configuring text input with the virtual keyboard

A visualization is usually configured so that it calls a virtual keyboard for a text input event when an input device is not available. For this purpose, the follow-up action *"Write variable"* is preset accordingly in the user input: The value *"Standard"* is selected for the *"Input type"* setting.

However, you can also configure especially how text is input. For this purpose, more input types are available in the user input, such as `Text input` or the listed visualizations. These visualizations have the visualization type *"Numpad/Keypad"* and display virtual keyboards or numeric keypads.

In the *"Settings for default text input"* setting of the visualization manager you can preset a keyboard visualization that is called from all visualizations in the application when required. This is possible without having to customize the user inputs of the visualizations.

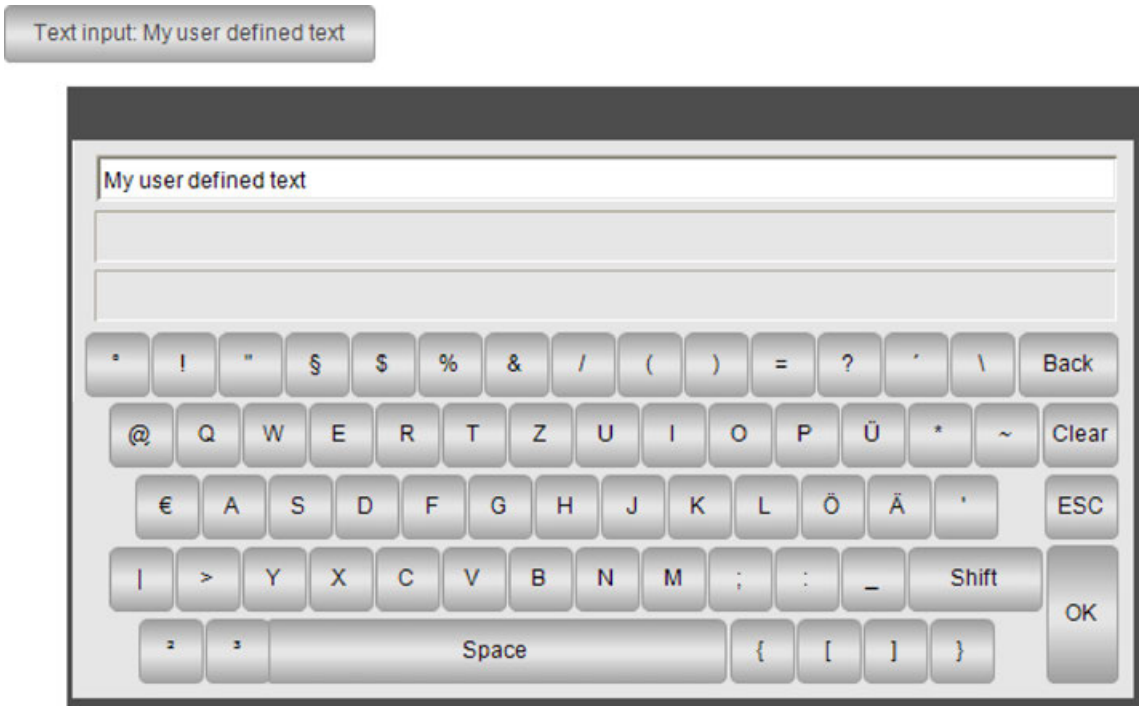
See also

- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Configuring text input especially for virtual keyboards

☑ Requirement: A project is open with a visualization.

1. Declare an input variable in the `PLC_PRG` program.
   ⇨ `VAR_INPUT stInput : STRING; VAR_END`
2. Add a button to the visualization and select the element.
3. Configure the property *“Texts ➔ Text”* with `Text input: %s`.
4. Configure the property *“Text variables ➔ Text variable”* with `PLC_PRG.stInput`.
5. Click auf *“Configure”* in the property *“Input configuration ➔ OnMouseClick”*.
   ⇨ The *“Input Configuration”* dialog box opens. The selected input event is printed below the caption.
6. Select the *“Write variable”* action.
7. Select the visualization `Visudialogs.Keypad` in *“Input type”* of the implementation of the action.
   ⇨ The virtual keyboard `Visudialogs.Keypad` is selected as the input device.
8. Compile, download, and start the application.
   ⇨ The visualization opens.
9. Click the button as a visualization user.
   ⇨ The virtual keyboard appears and allows text input by means of the mouse.



Configuring numeric input especially for virtual numeric keypads

☑ Requirement: A project is open with a visualization.

1. Declare an input variable in the `PLC_PRG` program.

   ⇨ `VAR_INPUT iInput : INT; VAR_END`

2. Open the visualization and added a *“Rectangle”* element.
3. Select the element in the editor.

   ⇨ The properties are visible in the *“Properties”* view.

4. Configure the property *“Texts ➔ Text”* with `Number input: %i`.
5. Configure the property *“Text variables ➔ Text variable”* with `PLC_PRG.iInput`.
6. In the *“Input configuration”* property, click the *“Configure”* button in the `OnMouseClick` line.

   ⇨ The *“Input Configuration”* dialog box opens. The selected input event is printed below the caption.

7. Select the `Write variable` action from the list of possible actions and click the > symbol.

   ⇨ The action appears in the list of actions to be executed. The blank implementation of the action appears in the window area to the right of the list.

8. Select the following settings:

   “Input type” set to `VisuDialogs.Numpad`.

   “Choose variable to edit” set to *“Use text output variable”*.

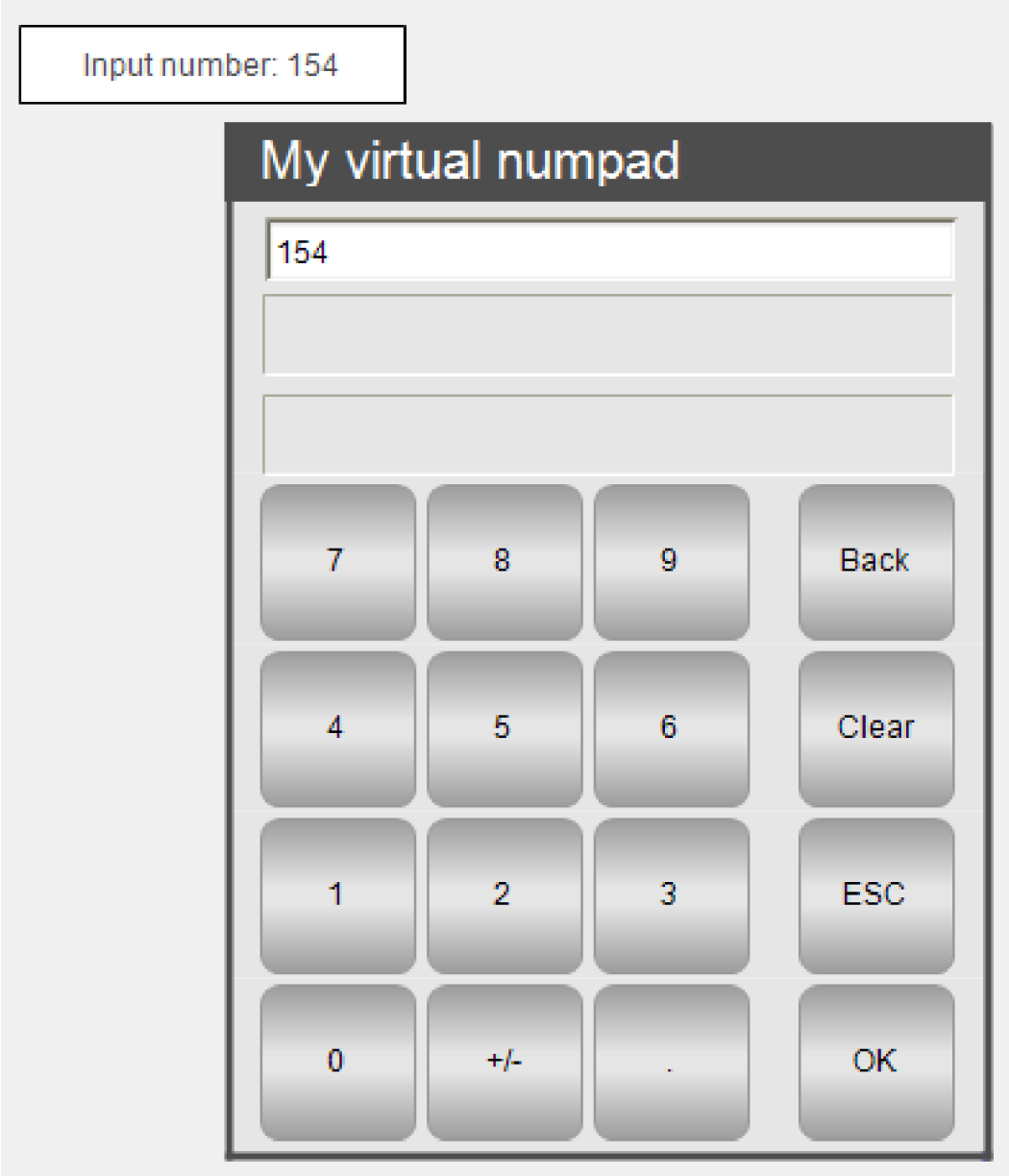
   “Dialog title” set to `'My virtual numpad'`.

9. Click *“OK”* to close the dialog box.

   ⇨ The user input is configured.

10. Compile, download, and start the application.

    ⇨ The application runs. The visualization opens. When a user clicks the rectangle, the numeric keypad opens.



Defining standard text input

☑ Requirement: A project is open with a visualization and a user input configuration. For all *"Write variable"* follow-up actions, the value *"Default"* is selected for the *"Input type"* setting.

1. Double-click the visualization manager.
2. Click in the default text input in the *"Settings"* tab (*"Default text input"* group) and assign visualizations.

    ⇨ These visualizations are defined as default text input. If a display variant does not have a keyboard, then these visualizations are called without you having to adapt the user input.

1.4.5.4.4 Configuring Keyboard Shortcuts

You can define keyboard shortcuts and assign specific actions to them. At runtime, a visualization detects the keyboard input event and executes the action.

There are different locations where you can configure a keyboard input event.

The options include the following:

- Configure keyboard input for a specific element.
- Configure keyboard input for a specific visualization.
- Configure keyboard input that is valid for all visualizations.
- Select the default hotkeys.

If the visualization integrated in CODESYS is executed, then you can deactivate the keyboard input of the visualization in order to use the keyboard shortcut from CODESYS in this state.

Configuring keyboard shortcuts for elements

You can define a keyboard shortcut that triggers an action for an element. The element has to be visible and operable. For this purpose, the property *"Input configuration ➔ Keyboard shortcuts"* is available in the *"Properties"* view of the visualization editor.

☑ Requirement: A CODESYS project is open with the existing visualizations `visEllipse` and `visRectangle`.

1. Select the application in the device tree and add a visualization named `visMain`.

   ⇨ The visualization editor opens.

2. In the *"Visualization Toolbox"* view, select and drag the *"Frame"* element to the editor.

   ⇨ The *"Configuration of Frame Visualizations"* dialog opens.

3. Double-click in succession the `visEllipse` and `visRectangle` visualizations in *"Available Visualizations"*.

   ⇨ The visualizations appear in *"Selected Visualizations"*.

4. Click *"OK"* to exit the dialog.

   ⇨ The visualization contains a new element type *"Frame"*. The 2 selected visualizations appear under its property *"References"*.

   In the editor, the frame shows the visualization with the index `0`.

5. Add a button and configure its properties:

   Select `Rectangle` in the property *"Texts ➔ Text"*.

   In the *"Input configuration ➔ OnMouseDown"* property, select *"Toggle frame visualization"* for the visualization `visRectangle`.

   Specify the value `R` in the property *"Input configuration ➔ Keyboard shortcuts ➔ Key"*.

   ⇨ The button has a user input and a keyboard shortcut.

6. Add a button and configure its properties:

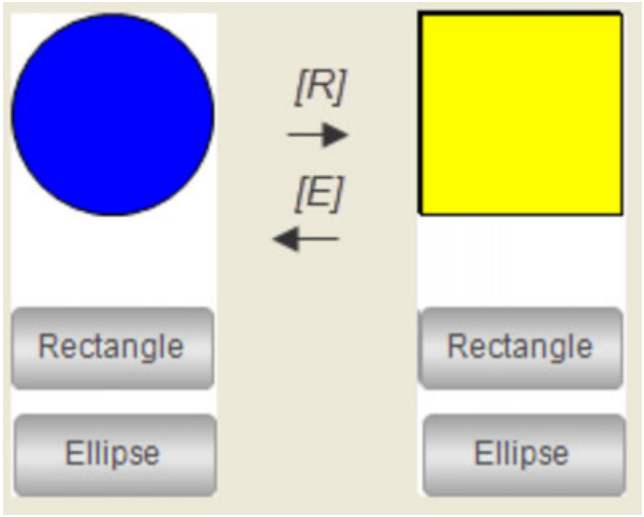
   Select `Ellipse` in the property *"Texts ➔ Text"*.

   In the *"Input configuration ➔ OnMouseDown"* property, select *"Toggle frame visualization"* for the visualization `visEllipse`.

   Specify the value `E` in the property *"Input configuration ➔ Keyboard shortcuts ➔ Key"*.

   ⇨ The button has a user input and a keyboard shortcut.

7. Click *"Online ➔ Login"* for the device and start the application.
   ⇨ The visualization starts. It has a frame where one of the referenced visualizations runs. Focus on the `visEllipse` visualization and press *[E]*. The visualization switches the contents in the frame to the `visEllipse` visualization. When you press *[R]*, the visualization switches the contents in the frame to the `visRectangle` visualization.



See also

- *Chapter 1.4.5.18.1.6 "Visualization Element 'Frame'" on page 1432*
- *"Input action 'Switch Frame Visualization'" on page 1756*

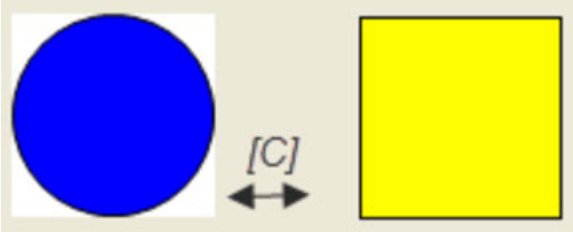
Configuring keyboard shortcuts for a specific visualization

You can define keyboard shortcuts that trigger an input action on a specific visualization. The *"Keyboard Configuration"* tab in the editor of the visualization is used for this purpose.

☑ Requirement: A CODESYS project is open with the visualizations `visEllipse` and `visRectangle`.

1. Open the CODESYS TargetVisu object and select `visEllipse` as the start visualization.
2. Open the `visEllipse` visualization and click the *"Keyboard Configuration"* tab.
3. Click *"Visualizations ➔ Keyboard Configuration"*.
   ⇨ The *"Keyboard Configuration"* tab opens.
4. Select the value `C` in the *"Key"* column.
5. Activate the *"Press key"* option.
6. Select the *"Change shown visualization"* value in the *"Action Type"* column.
7. Select `visRectangle` in the *"Action"* column.
   ⇨ The user input is configured for *[C]*.
8. Open the `visRectangle` visualization and click the *"Keyboard Configuration"* tab.
9. Select the value `C` in the *"Key"* column.
10. Activate the *"Press key"* option.
11. Select the *"Change shown visualization"* value in the *"Action Type"* column.
12. Select `visEllipse` in the *"Action"* column.
    ⇨ The user input for *[C]* is also configured for this visualization.

13. Build the application.
14. Click *"Online ➔ Login"* for the device and start the application.
    ⇨ The visualization starts and displays an ellipse. Focus on the `visEllipse` visualization and press *[C]*. The `visRectangle` visualization is displayed. Focus on the visualization and press *[C]* again. Now the visualization is switched again to `visRectangle`.



See also

- *"Tab 'Keyboard configuration'" on page 1720*
- *"Input action 'Change Shown Visualization'" on page 1752*

Configuring keyboard shortcuts for all visualizations in the application

You can define keyboard shortcuts that trigger the same input action for all visualizations of the application. The *"Default Hotkeys"* tab in the Visualization Manager is available for this purpose.

☑ Requirement: A project is open with a visualization.

1. Open the visualization.
2. Add a rectangle.
3. Configure the property *"Texts ➔ Text"* with `Keyboard shortcut`.
4. Double-click the *"GlobalTextList"* object.
5. Click in the table, *"Add Language"*, and then specify `de`.
   ⇨ The language `de` is configured.
6. Click in the table, *"Add Language"*, and then specify `en`.
   ⇨ The language `en` is configured.
7. Configure translations for `de` and `en` for the text `Keyboard shortcut`.
   ⇨ `Hotkey Keyboard Shortcut Hotkey`
8. Open the Visualization Manager and select the *"Default Hotkeys"* tab.
9. Specify `D` in the *"Key"* column.
10. Activate the *"Press key"* option.
11. Select the *"Change language"* value in the *"Action Type"* column.
12. Select the language `de` in the *"Action"* column.
    ⇨ The keyboard event for *[D]* is configured.
13. Specify `D` in the *"Key"* column.
14. Activate the *"Press key"* option.
15. Select the *"Alt"* option.
16. Select the *"Change language"* value in the *"Action Type"* column.

17. Select the language `en` in the *"Action"* column.
    ⇨ The keyboard event for *[Alt]+[D]* is configured.
18. Compile, download, and start the application.
    ⇨ The visualization opens.
19. As the visualization user, press *[D]*.
    ⇨ The text is displayed in the language `de`.

See also

- *Chapter 1.4.5.19.4.3 "Tab 'Visualization Manager' - 'Default Hotkeys'" on page 1781*
- *"Input action 'Change Language'" on page 1751*

Activating standard keyboard handling

When you activate the universal keyboard shortcuts for standard keyboard handling, the user can operate the visualization without a mouse. Elements that respond to user input can process a keyboard event instead of a mouse event without adapting its input configuration.

☑ Requirement: A project with a visualization is open.

1. Click the *"Visualization Manager"* object.
2. Activate the *"Activate standard keyboard handling"* option.
   ⇨ The universal keyboard shortcuts are activated.
3. Download the application to a device and start the application.
   ⇨ The visualization starts. Now operation can proceed without the mouse. You can navigate in the window by means of the *[Arrow]* and *[Tab]* keys and press *[Enter]* instead of the mouse button.

See also

- *Chapter 1.4.5.19.1 "Keyboard Shortcuts for Default Keyboard Action" on page 1717*

Activating and deactivating keyboard shortcuts for integrated visualizations

If you execute the visualization as an integrated visualization, then the *"Visualization ➔ Activate Keyboard Usage"* command is available in order to deactivate the capturing of keyboard events. It is actually possible for the same keyboard shortcuts to be defined in the visualization and in CODESYS

When you activate the command, the **visualization** executes the configured keyboard events.

When you deactivate the command, CODESYS executes the keyboard events. Capturing keyboard events is then deactivated for the visualization.

See also

- *Chapter 1.4.5.19.2.4 "Command 'Activate Keyboard Usage'" on page 1722*

1.4.5.4.5 Capturing user input events

You can capture user input events in the application. For this purpose, you can implement a function block that is executed when user events occur.

Capturing the writing of variables

When the user completes the input of a value (in an input field), an edit control event is closed. You can capture this event in the application as follows.

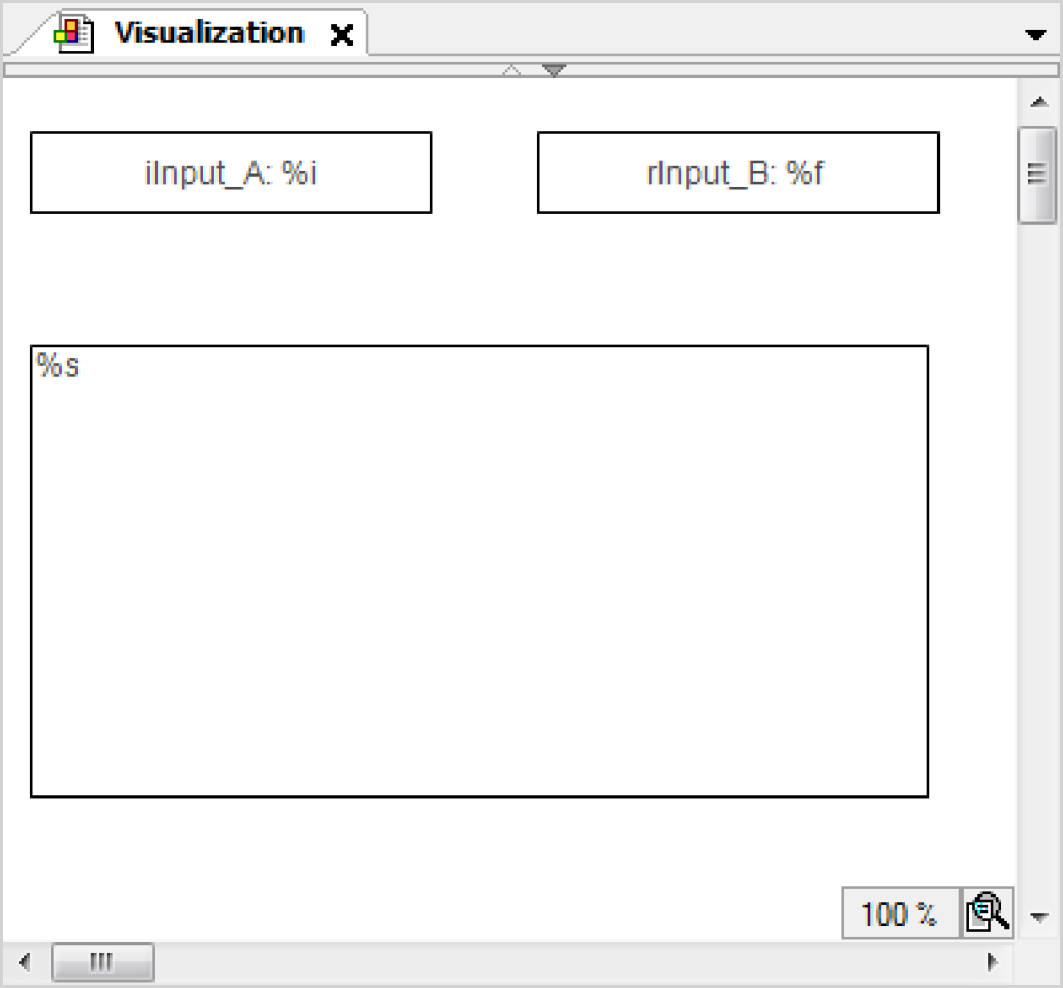
1. Create a function block that implements the `VisuElems.IEditBoxInputHandler` interface from the `VisuElemBase` library.
2. Pass the instance to the global event manager `VisuElems.Visu_Globals.g_VisuEventManager` by calling the `SetEditBoxEventHandler` method.

Example

A visualization has two input fields for `iInput_A` and `rInput_B` and one text output element.

The input fields are rectangles that the user is prompted to click in order to input text.

The text output element is a rectangle where the contents of the text variable `PLC_PRG.stInfo` are printed. The text variable contains the last input by a user in one of the input fields and the additional information that was added.



Properties of the rectangle `iInput_A`	
"Texts ➔ Text"	`iInput_A: %i`
"Text variables ➔ Text variable"	PLC_PRG.iInput_A
Properties of the rectangle `rInput_B`	
"Texts ➔ Text"	`iInput_B: %i`
"Text variables ➔ Text variable"	PLC_PRG.rInput_B
Properties of the rectangle for the text output	
"Texts ➔ Text"	`%s`
"Text variables ➔ Text variable"	PLC_PRG.stInfo

PLC_PRG implementation

```
PROGRAM PLC_PRG
VAR_INPUT
    iInput_A:INT;   (* Used in the visualization as user input
variable*)
    rInput_B:REAL; (* Used in the visualization as user input
variable*)
    stInfo : STRING;   (* Informs about the user input via the edit
```

```
control field;
        String gets composed by method 'VariableWritten;
        Result is displayed in the lower rectangle of the
visualization *)
END_VAR
VAR
    inst : POU;
    bFirst : BOOL := TRUE;
END_VAR

IF bFirst THEN
    bFirst := FALSE;

VisuElems.Visu_Globals.g_VisuEventManager.SetEditBoxEventHandler(ins
t);
    (* Call of method VariableWritten *)
END_IF
```

POU implementation

```
FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IEditBoxInputHandler
(* no further declarations, no implementation code *)
```

Method VariableWritten assigned to POU

```
METHOD VariableWritten : BOOL
(* provides some information always when an edit control field is
closed in the visualization, that is a variable gets written by
user input in one of the upper rectangles *)
VAR_INPUT
    pVar : POINTER TO BYTE;
    varType : VisuElems.Visu_Types;
    iMaxSize : INT;
    pClient : POINTER TO VisuElems.VisuStructClientData;
END_VAR

// String stInfo, which will be displayed in the lower rectangle,
is composed here
PLC_PRG.stInfo := 'Variable written; type: ';
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, INT_TO_STRING(varType));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', adr: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, DWORD_TO_STRING(pVar));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', by: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo,
SEL(pClient^.globaldata.clienttype =
VisuElems.Visu_ClientType.Targetvisualization,'other visu',
'targetvisu'));
```

Capturing keyboard events

When the user presses and releases the key, a keyboard event is triggered in the visualization. You can capture this event in the application as follows.

1. Create a function block that implements `VisuElems.IVisuUserEventManager` from the `VisuElemBase` library.
2. Pass the instance to the global event manager `VisuElems.Visu_Globals.g_VisuEventManager` by calling the `SetKeyEventHandler` method.

Example

A visualization has one text output element. The text output element is a rectangle where the contents of the text variable `PLC_PRG.stInfo` are printed. The text variable contains information about the last key pressed by the user.

Properties of the rectangle for the text output	
"Texts ➔ Text"	`%s`
"Text variables ➔ Text variable"	PLC_PRG.stInfo

Implementation of the `PLC_PRG` program

```
PROGRAM PLC_PRG
VAR_INPUT
    stInfo : STRING;
END_VAR
VAR
    inst : POU;
    bFirst : BOOL := TRUE;
END_VAR

IF bFirst THEN
    bFirst := FALSE;

VisuElems.Visu_Globals.g_VisuEventManager.SetKeyEventHandler(inst);
END_IF
```

Implementation of the `POU` function block

```
FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IKeyEventHandler
(* no further declarations, no implementation code *)
```

Implementation of the `VariableWritten` method of the `POU` function block

```
/// This method will be called after a key event is released.
/// RETURN:
/// TRUE - When the handler has handled this event and it should
not be handled by someone else
/// FALSE - When the event is not handled by this handler
METHOD HandleKeyEvent : BOOL
VAR_INPUT
    /// Event type. The value is true if a key-up event was
released.
    bKeyUpEvent : BOOL;
    /// Key code
    dwKey : DWORD;
    /// Modifier. Possible values:
    /// VISU_KEYMOD_SHIFT : DWORD := 1;
    /// VISU_KEYMOD_ALT : DWORD := 2;
    /// VISU_KEYMOD_CTRL : DWORD := 4;
    dwModifiers : DWORD;
    /// Pointer to the client structure were the event was released
    pClient : POINTER TO VisuStructClientData;
END_VAR
VAR
END_VAR

PLC_PRG.stInfo := 'KeyEvent up: ';
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo,
BOOL_TO_STRING(bKeyUpEvent));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', key: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, DWORD_TO_STRING(dwKey));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', modifier: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo,
DWORD_TO_STRING(dwModifiers));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', by: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo,
```

```
SEL(pClient^.globaldata.clienttype =

VisuElems.Visu_ClientType.Targetvisualization,


                                                'other visu',
'targetvisu'));
```

Recording variable value changes triggered by input events

All visualization elements that change the value of a variable by user input call the `IValueChangedListener` interface. With this interface, the value changes can be recorded and then processed programmatically.

1. Implement a function block (example: `POU`) that implements the `IValueChangedListener` interface.

   ```
   FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IValueChangedListener
   ```

   ⇨ In the device tree, the *"ValueChanged"* method is inserted below the function block.

2. In a program (example: *"PLC_PRG"*), implement the IEC code that registers the interface.

   ```
   VisuElems.g_itfValueChangedListenerManager.AddValueChangedListene
   r(itfValueChangedListener)
   ```

   ⇨ *"PLC_PRG"* receives all value changes by means of the *"ValueChanged"* method.

   Now you can record and process the value changes.

1.4.5.5 Setting Up User Management

In the visualization user management, you define users and user groups and assign access rights to user groups for individual visualization elements. In the user management dialogs, users can be registered and unregistered in runtime mode and passwords and user management can be changed.

In a project with several applications, you can configure user management for each application.

NOTICE!
When a visualization user management exists, an unregistered user automatically receives the access rights from the `None` group.

1.4.5.5.1 Setting up user management for visualizations

When you set up user management for your visualization, the following variants are possible:

- Empty user management
  An empty user management contains the `None` user group. You configure all users and groups yourself.
- User management with default users and groups
  This user management contains the `Admin`, `Service`, `Operator`, and `None` groups. The first three groups each contain one user with the same name as the group.

Create new user management

☑ Requirement: A user management does not exist yet for your visualization.

1. Click the *“Visualization Manager”* object in the device tree.
2. Select the *“User Management”* tab.
3. Click *“Create Empty User Management”* or *“Create User Management with Default Groups and Users”* depending on which variant you need.

   ⇨ The *“Groups”*, *“Users”*, and *“Settings”* tabs are displayed.

   The *“Group”* tab opens.
4. If you have created an empty user management, then you configure the new groups and users.

   If you have created a user management with default groups and users, then you can grant permissions for elements in your visualization. You can also select the user management dialogs and assign them to the buttons of the visualization.

See also

- *Chapter 1.4.5.19.4.5 “Tab 'Visualization manager' - 'User management'” on page 1782*
- *Chapter 1.4.5.5.2 “Configuring users and groups” on page 1283*
- *Chapter 1.4.5.5.4 “Configuring permissions for groups” on page 1285*
- *Chapter 1.4.5.5.3 “Editing and Selecting User Management Dialogs” on page 1284*

1.4.5.5.2 Configuring users and groups

Groups and their users for the basis for user management. A group has one or more users; a user can belong to multiple groups. The permissions of the visualization elements are always assigned to a group.

Adding groups

☑ Requirement: You have already created a user management by clicking *“Create Empty User Management”* or *“Create User Management with Default Groups and Users”* in the *“Visualization Manager”* (*“User Management”* tab).

1. Click the *“Visualization Manager”* object in the device tree.
2. Select the *“User Management”* tab.
3. Click in the last line of the list.

   In this line, the field of the *“Group Name”* column is still empty.
4. Click in the field of the *“Group Game”* column and specify the name for the new group.
5. If necessary, activate the options *“Automatic Logout”* and *“Permission to Change User Data”*.

Adding users and assigning groups

☑ Requirement: A user management exists with at least on group. The *“Visualization Manager”* is open.

1. Select the tab *“User Management ➔ User”*.
2. Click in the last empty line of the list.

3. Specify the *“Login Name”*.
   ⇨ CODESYS applies the *“Login Name”* as *“Password”*.
4. If you want to change the password, click in the *“Password”* field of the user.
   ⇨ The *“Change Password”* dialog box opens.
5. Click the *“User Group”* field.
   ⇨ The *“User Groups the User Belongs to”* dialog box opens.
6. Activate the *“Assigned”* option for the groups that the user should be long to and click *“OK ”* to confirm.
   ⇨ In the *“User Group”* field, all groups are listed that the new user belongs to.

See also

- *Chapter 1.4.5.5.1 “Setting up user management for visualizations” on page 1282*
- *Chapter 1.4.5.19.4.5 “Tab 'Visualization manager' - 'User management'” on page 1782*

1.4.5.5.3 Editing and Selecting User Management Dialogs

In the user management dialogs, you define the login, logout, changing of the user password, and editing of the user management in the visualization at runtime.

> **NOTICE!**
> If you create your own dialog as a user management dialog, then you should use the visualizations from the included library project `VisuUserMgmtDialogs.library` as the basis, because it uses the required interfaces. Your own user management dialog is listed then in *“Visualization Manager ➔ Settings”*, *“Settings for User Management Dialogs”*.

Editing user management dialogs

Requirement: The library project `VisuUserMgmtDialogs.library` exists in the installation directory.

1. Click *“File ➔ Open Project”*.
2. Select the project `VisuUserMgmtDialogs.library` from the `Projects` folder of the installation directory.
3. Click *“View ➔ POUs”*.
   ⇨ In the *“POUs”* view, the project is displayed with the visualizations *“UserMgmtChangePassword”*, *“UserMgmtConfig”*, and *“UserMgmtChangePassword”*.
4. Double-click a visualization (example: *“UserMgmtLogin”*).
5. Change the visualization as you like and save the project.
6. Then, reinstall the library and add it to the *“Library Manager”* of your application.

Selecting user management dialogs

☑ A user management already exists in your application in the *“Visualization Manager”* object (*“User Management”* tab).

☑ The *“VisuUserManagement”* library is in the Library Manager.

1. In the device tree, click the *“Visualization Manager”* object.
2. Select the *“Dialog Settings”* tab.
3. In *“Settings for User Management Dialogs”*, select the dialogs for *“Login dialog”*, *“Change password dialog”*, and *“Change configuration dialog”*.

> *If no entries can be seen in “Settings for User Management Dialogs” in the dialog lists, then close the “Visualization Manager” and reopen it.*

Configuring visualization buttons for the login, logout, change password, and user management dialogs

☑ Requirement: A visualization is open.

1. Drag a *“Button”* element from the *“Visualization Toolbox”* view (*“Common Controls”* category) to the visualization.
2. In the *“Properties”* view, click the *“Input configuration”* node.
3. In the *“Input configuration ➔ OnMouseClick”* property, click *“Configure”*.
4. In the *“Input configuration”* dialog, click *“User Management”* and >.
   ⇨ The following *“Dialogs and actions”* are listed on the right: *“Login”*, *“Logout”*, *“Change User Password”*, and *“Open User Configuration”*.
5. Select the dialog or action to assign to the button and click *“OK”*.
   ⇨ When the button is clicked at runtime, the selected dialog opens or the selected action is executed.

> *If you want to open and edit the user management in the visualization at runtime, you have to be a member of a group that has “Permission to Change User Data”.*

See also

- *Chapter 1.4.5.4.1 “Configuring user inputs for visualization elements” on page 1268*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

1.4.5.5.4 Configuring permissions for groups

Configuring permissions for an element of the visualization

Permissions for visualization-elements are not granted to individual users, but to groups with assigned users.

☑ Requirement: A *“Visualization”* object is open with at least one inserted visualization element.

1. Click a visualization element in the editor.
2. Click the *“Value”* field of the *“Permissions”* element property in the *“Properties”* view.

   ⇨ The *“Permissions”* dialog box opens.
3. Select the permissions that the respective user group should have for the visualization element.

   Note: If the option *“Group hierarchy is used”* is activated, the groups lower in the hierarchy cannot be granted more permissions than groups higher in the hierarchy.

In the “Element List” of the visualization, the “Permissions” column shows the element permissions granted to groups.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*
- *Chapter 1.4.5.19.4.5 “Tab 'Visualization manager' - 'User management'” on page 1782*
- *Chapter 1.4.5.19.4.1.1 “Visualization Editor” on page 1772*

1.4.5.6 Setting Up Multiple Languages

Texts and tooltip texts for visualizations are managed in text lists and can be displayed in different languages. To switch a visualization between the available languages, configure a visualization element with the corresponding input configuration for changing the language.

There are static texts that are managed in *“GlobalTextList”* (generated automatically) and dynamic texts from created text lists. A dynamic text can be changed at runtime with a variable that defines the index of the text list entry. Static texts are fixed labels within a visualization; dynamic texts are often used for displaying variable values or error messages.

For creating and using text lists, see: *Chapter 1.4.1.8.8 “Managing text in text lists” on page 266*.

You can modify the appearance and formatting of texts and tooltips with the element properties “Text properties” and “Font variables”.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Configuring language switching for texts from text lists

Requirement: An empty visualization object is inserted into the project and it is open for editing in the visualization editor. There is also a *“Visualization Manager”* object. User management is not created for the visualization.

The following instructions provide a simplified example:

- By means of two buttons, the user should be able to toggle the visualization texts between English and German.
- Static texts in the visualization include the labels "State, Machine 01", "State, Machine 02", "English", and "German". These texts are located in the *"GlobalTextList"* in English and German.
  Dynamic texts will describe the state of both machines. The texts are provided in the text list *"Status_Texts"* in English (`en`) and German (`de`).

1. Drag a *"Text Field"* from the *"Visualization Toolbox"* view (*"Common Controls"* category) to the editor view. Specify the value `State, Machine M01` in the properties editor for the element property *"Texts ➔ Text"*.
2. Copy the element and change the copy label to `State, Machine M02`.
3. See also the figure in step 14 for the following steps.

   Insert two elements of type *"Button"* from the *"Visualization Toolbox"* view (*"Common Controls"* category) in the visualization editor. With these elements, the user should be able to toggle the language of the visualization. Specify the text `German` or `English` in the properties editor for element property *"Texts ➔ Text"* (4).
4. Double-click and open *"GlobalTextList"* in the *"POUs"* view.

   ⇨ The texts are entered in the *"Standard"* (1) column, and the *"ID"*s `0` and `1` are assigned automatically as additional information.
5. Add the languages *"de"* and *"en"* with the texts shown in the following figure.

GlobalTextList

ID	Default (1)	de	en
3	English	Englisch	English
2	German	Deutsch	German
0	State, Machine M01:	Zustand Maschine M01:	State, Machine M01:
1	State, Machine M02:	Zustand Maschine M02:	State, Machine M02:

6. Close the *"GlobalTextList"*.
7. Add two elements of type *"Rectangle"* from the *"Visualization Toolbox"* view (*"Basic"* category) in the visualization editor. The current state of each machine should be displayed.
8. For managing the texts for describing the states, add an object of type *"Text List"* below the application. Name the list `Status_Texts`.
9. Specify the texts shown in the figure for the standard language (1) and the target languages *"en"* and *"de"* in the editor of *"Status_Texts"*.

Status_Texts

ID	Default (1)	de	en
0	running	läuft	running
1	stopped	läuft nicht	not running
2	starting	läuft gerade hoch	starting
3	error	Fehlermeldung!	error message!

10. Close the text list *"Status_Texts"*.
11. Select the rectangle element for displaying the state of machine M01. Select the text list `Status_Texts` from the combo box in the properties editor (2) for the *"Dynamic texts"* element property (5). Specify an application variable for *"Text index"* that shows the appropriate text index for the state of the machine at runtime. Example: `PLC_PRG.ivar_status_m01`.

12. Now configure the user input for both buttons for toggling the language in the visualization.

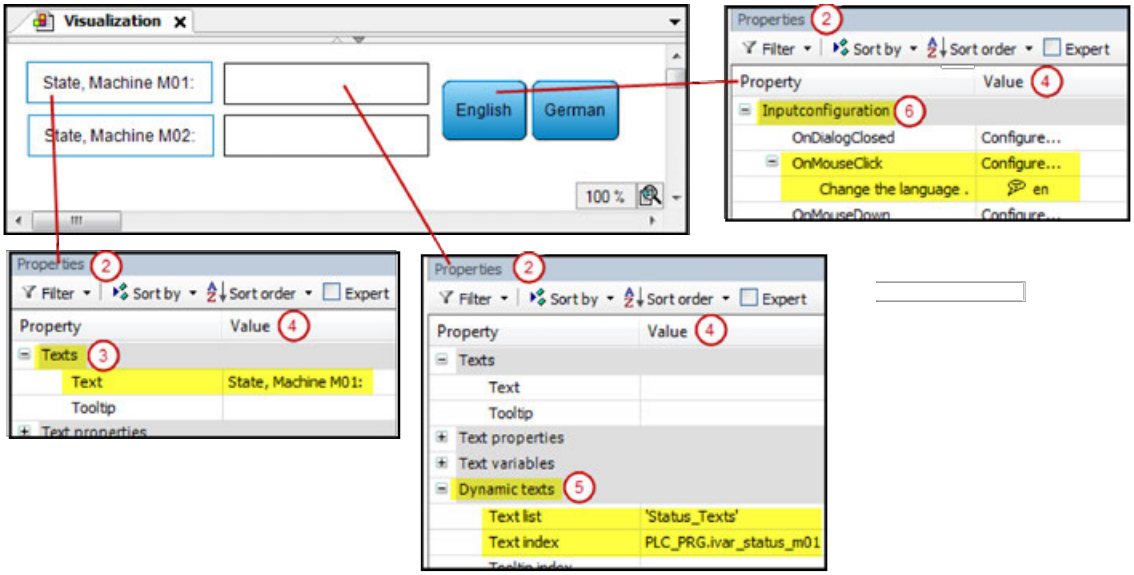
    Select the "German" button. Double-click *“Configure”* of the property *“Input configuration”* (6), *“OnMouseClick”*.

    ⇨ The *“Input configuration / OnMouseClick”* dialog opens.

13. Select *“Change the language”* on the left. Click the arrows to accept the setting to the right. Select *“de”* in *“Language”* to the right of the dialog in the ... input assistance. Click *“OK”* to confirm.

14. In the same way, configure the entry for the second button: `English`: *“Text list”*: `Status_Texts`, *“Text index”*: `3`, *“Change the language”*: English).

    ⇨ The following figure shows the performed properties configurations for the four visualization elements.

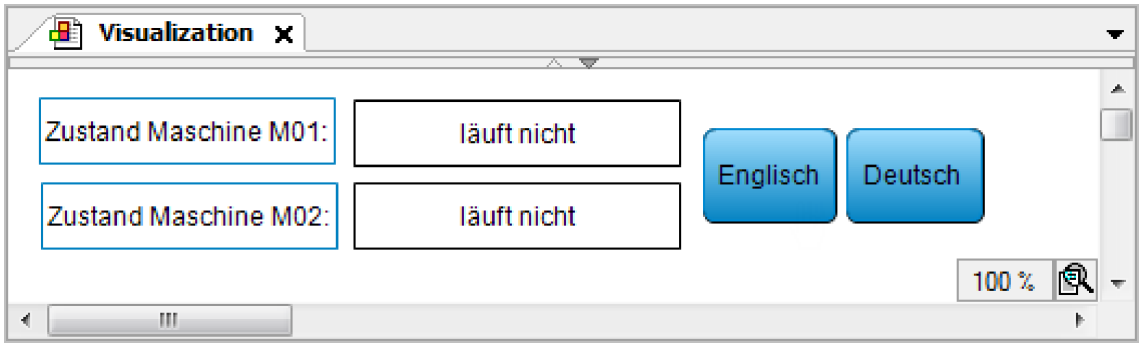


15. When the application is compiled without errors, you can test the visualization in simulation mode. Activate the option *“Online ➔ Simulation”*. Click *“Online ➔ Login”*.

    ⇨ The visualization appears in the visualization editor view in online mode:

16. Click the *"German"* button.

    ⇨ The language changes to German:



See also

- *"Input action 'Change Language'" on page 1751*

Setting up fonts for a language

The font for a visualization element is defined in the properties editor. If a language switch is provided, you can overwrite this basic font with another font for each language in the visualization manager.

Requirement: A visualization is set up with at least one language in addition to the default language. For an example, see *"Configuring language switching for texts from text lists" on page 1286*

1. Double-click and open the *"Visualization Manager"* object and select the *"Font"* tab.
2. Double-click the field in the *"Font"* line for a particular language. Select a font from the combo box.
3. In the *"Font size"* line, replace the value `1` with a value greater than 1 (example: `2`) in order to increase the size of the font as defined by the visualization style; or replace it with a value less than 1 in order to decrease is (example: 0.5).

   ⇨ In online mode, the font changes depending on the set language.

See also

- *Chapter 1.4.5.19.4.6 "Tab 'Visualization Manager' - 'Font'" on page 1786*

1.4.5.7 Visualizing alarm management

In CODESYS, the alarm management is a powerful object for creating and managing alarms. You can group alarms and set the acknowledgement behavior individually. The alarm display in the visualization can also be customized.

The *"Alarm Table"* and *"Alarm Banner"* visualization elements are available for displaying and processing alarms. The alarm table lists the alarm texts. The alarm banner is a simplified version of the alarm table. It visualizes a single alarm only. However, by adding scroll elements you can allow for switching the display from one active alarm to another active alarm.

See also

- *Chapter 1.4.1.8.20 "Alarm Management" on page 309*

Creating an alarm table

Requirement: In your project, alarms are defined in alarm groups and they are assigned to an alarm class. The following instructions are based on the example that is described in the "Configuring alarm management" chapter.

1. Open the visualization editor.
2. Drag the *“Alarm Table”* element from the *“Alarm Manager”* group to the visualization editor.
   ⇨ The *“Alarm Table”* visualization element is visible in the editor.
3. In the *“Alarm configuration”* / *“Alarm groups”* property, define the alarm groups that you want to visualize. Click into the value field.
   ⇨ The *“Select Alarm Group”* dialog opens.
4. Clear the *“All”* check box and select the *“PartsDeficit”* alarm group. Add the group to the selected alarm groups by clicking the > button.
5. In the *“Alarm configuration”* / *“Alarm classes”* property, define the alarm classes that you want to visualize. Click into the value field.
   ⇨ The *“Select Alarm Class”* dialog opens.
6. Clear the *“All”* check box and select the *“PartsDeficit”* alarm class. Add the alarm class to the selected alarm classes by clicking the > button.
7. Add an additional column. Click the *“Columns”* / *“Create New”* button.
   ⇨ CODESYS adds the column *“[2]”* to the properties. The *“Symbol”* column is added to the table.
8. Select data type *“State”* for column [2].
   ⇨ The default column heading *“State”* is shown in the table.
9. Name the *“Column heading”* column "Status".
10. Specify the appearance of the selected table cell. Set the *“Selection”* / *“Selection color”* to *“Green”*.
11. In the *“Control variables”* / *“Confirm selection”* property, specify the variable `bQuitAlarm` for confirming messages.
12. Adjust the other properties to your requirements. See the "Alarm table" visualization element for a complete description of the properties.

See also

- *Chapter 1.4.5.18.1.22 “Visualization Element 'Alarm Table'” on page 1545*

Inserting elements for acknowledging alarms

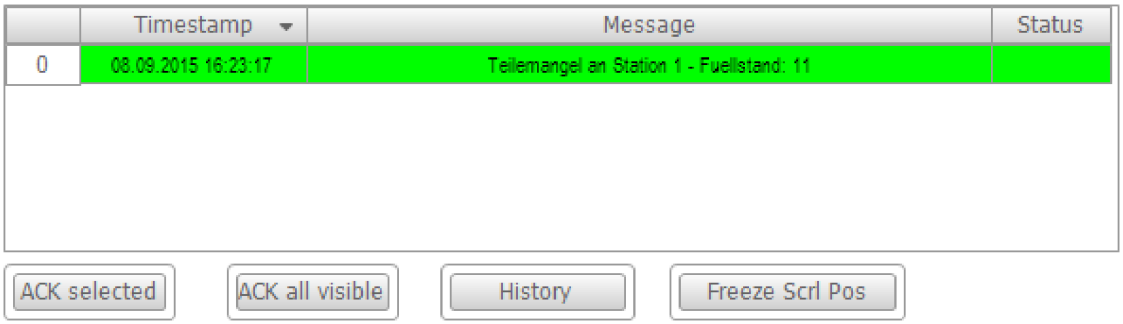
In CODESYS, predefined buttons are available for controlling the alarms in an alarm table.

Requirement: An *“Alarm table”* element exists in the visualization.

1. Select the visualization element in the editor.
2. Click *“Visualization ➔ Insert elements for acknowledging alarms”*.
   ⇨ The *“Alarm Table Wizard”* dialog opens.

3. Click *“OK”* to accept all settings.

   ⇨ Four buttons are added for controlling the alarm table.



See also

- ↳ *Chapter 1.4.5.18.1.22 “Visualization Element 'Alarm Table'” on page 1545*

Creating an alarm banner

Requirement: In your project, alarms are defined in alarm groups and they are assigned to an alarm class. The following statement is based on the example that is described in the "Configuring alarm management" chapter.

The alarm banner displays an active alarm in online mode. If there are multiple active alarms, filtering takes place by means of the filter criteria set in the alarm banner (newest for filter criterion "Priority" and most important for filter criterion "Newest"). See the instructions below for adding scroll elements in order to switch the display between multiple alarms.

1. Open the visualization editor.
2. Drag the *“Alarm banner”* element from the *“Alarm manager”* group to the visualization editor.

   ⇨ The *“Alarm banner”* visualization element is visible in the editor.
3. In the *“Alarm configuration”* / *“Alarm groups”* property, define the alarm groups that you want to visualize. Click into the value field.

   ⇨ The *“Select Alarm Group”* dialog opens.
4. Clear the *“All”* check box and select the *“PartsDeficit”* alarm group. Add the group to the selected alarm groups by clicking the > button.
5. In the *“Alarm configuration”* / *“Alarm classes”* property, define the alarm classes that you want to visualize. Click into the value field.

   ⇨ The *“Select Alarm Class”* dialog opens.
6. Clear the *“All”* check box and select the *“PartsDeficit”* alarm class. Add the alarm class to the selected alarm classes by clicking the > button.
7. Set the *“Alarm configuration”* / *“Filter criterion”* property to *“Newest”*.

   ⇨ In online mode, the newest alarm message is always shown.
8. Add an additional column. Click the *“Columns”* / *“Create new”* button.

   ⇨ CODESYS adds the column *“[2]”* to the properties. The *“Symbol”* column is added to the table.
9. Select data type *“State”* for column [2].

   ⇨ The default column heading *“State”* is shown in the table.
10. In the *“Confirmation variable”* property, specify the variable `bQuitAlarm` for confirming messages.

See also

- ↳ *Chapter 1.4.5.18.1.23 “Visualization Element 'Alarm Banner'” on page 1554*

Adding elements for scrolling the active alarms

Elements can be added to an alarm banner for switching the display between the individual active alarms. You can control the scrolling with visu-local variables or application variables.

1. Select the added "Alarm banner" visualization element. Click *“Insert Elements for Scrolling Alarms”* in the context menu.
   ⇨ The *“Alarm Banner Wizard”* opens.
2. Select the element type for the scroll elements: *“Button”* or *“Rectangle”*.
3. Activate the action(s) for which a control should be inserted: *“Scroll to next alarm”*, *“Scroll to previous alarm”*.
4. Specify a Boolean variable that gets the value `TRUE` when multiple active alarms are present. If you have already configured a project variable in the element properties, then it is also specified here in the wizard. Otherwise CODESYS automatically creates the visu-local variable *“xMultipleAlarmsActive”*.
5. In the next step, check the configuration of the element properties of the extended alarm banner.
6. Select the alarm banner element and look at the section *“Handling of multiple active alarms”* in the *“Properties”* view. You have two options:
7. Option 1: The display should switch automatically. Activate the *“Switch automatically”* property.
   ⇨ Now, in *“Every N seconds”* you define the time interval after which the display in the alarm banner in online mode should switch to the next alarm.
8. Option 2: The display should be controlled by means of the application. Deactivate the *“Switch automatically”* property.
   ⇨ Switching between the active alarms can be controlled by two variables. By default, `xNext` and `xPrev` are created for scrolling to the next or previous alarm. You can replace these variables with custom your own defined application variables.

Filter alarm events by the contents of the latch variable

Filtering by the contents of a latch variable can be useful when there are a lot of alarm events displayed. If the latch variable assigned to an alarm in the alarm group definition contains, for example, the error number or the name of a device instance, then you can filter the alarms in the visualization by it.

For this purpose, you configure an input option in the alarm visualization for the contents of the latch variable to be filtered by. For example, insert an input field which writes to the variable that is specified in the *“Alarm configuration”* - *“Filter by latch 1”* - *“Filter variable”* property of the configuration of the *“Alarm table”* element or *“Alarm banner”* element.

In addition, you configure an input option for the type of filtering. The type determines whether a numeric value (typed literal, LINT literal) or the string value of the latch variable is used for filtering. Filtering can also be switched off by means of type setting `0`. For example, in the visualization, insert another input field which writes to the variable that is specified in the *“Filter type”* property of the configuration of the alarm table or alarm banner.

For more information, see the "Alarm Filter Latch Example" sample project in the CODESYS *Store*.

1.4.5.8 Animating visualization elements

The animation of a visualization element at runtime can serve to visualize value curves in addition to serving purely visual purposes. Animation is possible through a dynamic configuration of certain element properties, i.e. by controlling these properties with a variable. See the following examples of possible animations.

1.4.5.8.1 Configuring rotations and offsets

You can animate a visualization element and have it shifted or rotated at runtime. To do this you assign variables in its property *"Absolute movement"* and then program the animation in the application code.

Configuring an offset

You can configure an offset of the element by programming the variables in *"Absolute movement ➔ Movement"*.

☑ Requirement: A project with a visualization is open.

1. Open the visualization and add an element *"Rectangle"*.
   ⇨ The view *"Properties"* displays the configuration of the element.
2. In the application in the POU `PLC_PRG`, declare type-compliant variables: `diOffsetX : DINT;` and `diOffsetY : DINT;`
3. Configure the property *"Absolute movement ➔ Movement ➔ X"* with `PLC_PRG.diOffsetX` and *"Y"* with `PLC_PRG.diOffsetY`.
4. Implement a shift of the element, for example by means of a modulo division of the value:

   ```
   diOffsetX := diOffsetX MOD 100;
   diOffsetY := diOffsetY MOD 100;
   ```

5. Compile, load and start the application.
   ⇨ The application runs. The visualization opens. The rectangle moves.

Configuring a rotating element

When an element rotates, then the center point of the element rotates precisely around its center. The center is defined in the property *"Center"*. The center point of an element is calculated internally. If the center point and center coincide, then there is no rotation.

You can configure a clockwise rotation of the element by increasing the value of the variable *"Absolute movement ➔ Rotation"*.

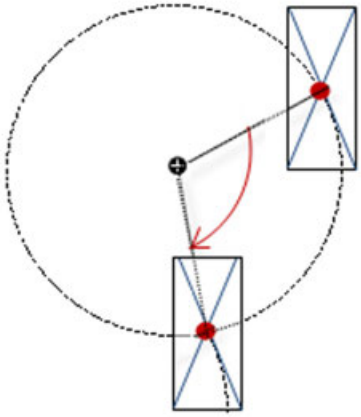
☑ Requirement: A project with a visualization is open.

1. Open the visualization and add an element *"Rectangle"*.
   ⇨ The view *"Properties"* displays the configuration of the element.
2. In the application in the POU `PLC_PRG`, declare a type-compliant variable:`rValue : REAL;`
3. Configure the property *"Absolute movement ➔ Rotation"* with `PLC_PRG.rValue`.
4. Implement the clockwise rotation of the element by increasing the value of the variable:

   ```
   rValue := rValue + 0.1;
   ```

5. Compile, load and start the application.

   ⇨ The application runs. The visualization opens. The rectangle rotates about the center. The alignment of the element with respect to the coordinate system is fixed.



Configuring a rotating element

When an element performs an inner rotation and rotates, then the center point of the element rotates precisely around its center. This is the point defined in the property *"Center"*. The alignment of the element also rotates relative to the coordinate system. If the center point of the element and the center coincide, this produces a rotation on the spot.

You can configure a clockwise rotation of the element by increasing the value of the variable *"Absolute movement ➔ Inner rotation"*.

If the visualization is In runtime, you can see that the element rotates (also relative to the coordinate system of the visualization).

☑ Requirement: A project with a visualization is open.

1. Open the visualization and add an element *"Polygon"*, which you form into a pointer.

   ⇨ The view *"Properties"* displays the configuration of the element.

2. Drag the center point of the element to the base of the pointer.
3. In the application in the POU `PLC_PRG`, declare a type-compliant variable:

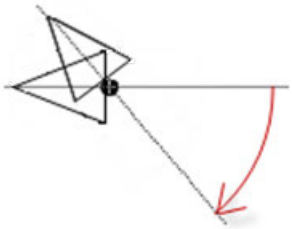
   ```
   rValue : REAL;
   ```

4. Configure the property *"Absolute movement ➔ Inner rotation"* with `PLC_PRG.rValue`.
5. Implement the clockwise rotation of the element by increasing the value of the variable:

   rValue := rValue + 0.1;

6. Compile, load and start the application.

   ⇨ The application runs. The visualization opens. The pointer rotates about its base.



See also

- ↳ *Chapter 1.4.5.18.1.1 “Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'” on page 1368*

1.4.5.8.2 Animating a text display

An animation of the text display can be configured in the property *“Font variables”*. All basic elements have this property as well as tables, scrollbars and text fields.

Example: animating the font size

☑ Requirement: A project with a visualization is open.

1. Open the visualization and add an element *“Rectangle”*.
   ⇨ The view *“Properties”* displays the configuration of the element.
2. Configure the property *“Texts ➔ Text”* with `Important`:
3. In the application in the POU `PLC_PRG`, declare a type-compliant variable:
   ⇨ `iFontHeight : INT;`
4. Configure the property *“Font variables ➔ Size”* with `PLC_PRG.iFontHeight`.
5. Implement a change of the font size.
   ⇨ iFontHeight := iFontHeight + 1) MOD 20;
6. Compile, load and start the application.
   ⇨ The application runs. The visualization opens. The rectangle is labelled with `Important`. The font size grows from 1 to 20.

See also

- ↳ *Chapter 1.4.5.18.1.1 “Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'” on page 1368*

1.4.5.8.3 Animating a color display

The colors of an element are specified in the *“Colors”* properties of the element properties. There you can select either a predefined style color from the selection list or a color in the color dialog.

The *“Color variables”* element property is used for the color animation of the element. If you pass variables to the properties, then you can program color changes in the application code or configure a user input that results in a color change. A color constant or color variable in the code has the data type `DWORD` and is encoded according to the RGB color space or RGBA extension.

Color definition in RGBA color space

> **NOTICE!**
> The *“Activate semi-transparent drawing”* option is provided in the Visualization Manager. This option is enabled by default so that the *“Transparency”* property is available for all color definitions. With programmatic color definition, the leading byte is interpreted as an alpha channel and therefore used as the transparency value of the color. When the option is cleared, the *“Transparency”* property is not available and the leading byte is ignored in color literals.

Color information in the code is specified as `DWORD` literals. The value is in the RGBA color space and is usually shown as a hexadecimal number. The value is coded with additive portions of red, green, and blue. It is appended with the alpha channel which determines the transparency of the color.

Byte order of a color literal

```
16#<TT><RR><GG><BB>

<TT> : 00 - FF     // Transparency in 256 levels
<RR> : 00 - FF     // Red in 256 levels
<GG> : 00 - FF     // Green in 256 levels
<BB> : 00 - FF     // Blue in 256 levels
```

The graduation value for transparency is `16#FF` for opaque and `16#00` for transparent. For each color portion, one byte is reserved for 256 color graduations `16#FF` to `16#00`. `16#FF` means 100% color portion and `16#00` means 0% color portion.

<TT>	Byte for the transparent graduation of 00-FF
<RR>	Byte for the red portion of 00-FF
<GG>	Byte for the green portion of 00-FF
<BB>	Byte for the blue portion of 00-FF

Example

Table 262: Color literal

16#FF0000FF	Blue, opaque
16#FF00FF00	Green, opaque
16#FFFFFF00	Yellow. opaque
16#88888888	Gray, semitransparent
16#88000000	Black, semitransparent
16#FFFF0000	Red, opaque

Example

Global declaration of color constants

```
VAR_GLOBAL CONSTANT
    c_dwBLUE : DWORD := 16#FF0000FF;          // Highly opaque
    c_dwGREEN : DWORD := 16#FF00FF00;          // Highly opaque
    c_dwYELLOW : DWORD := 16#FFFFFF00;          // Highly opaque
    c_dwGREY : DWORD :=16#88888888;          // Semitransparent
    c_dwBLACK : DWORD := 16#88000000;          // Semitransparent
    c_dwRED: DWORD := 16#FFFF0000;          // Highly opaque
END_VAR
```

Animating a visualization element in color

1. Create a standard project in CODESYS.
2. Declare global color constants in the POU tree.

   ⇨

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
    gc_dwRed : DWORD := 16#FFFF0000;
    gc_dwGreen: DWORD := 16#FF00FF00;
    gc_dwYellow: DWORD := 16#FFFFFF00;
    gc_dwBlue: DWORD := 16#FF0000FF;     // Highly opaque
    gc_dwBlack : DWORD := 16#88000000;     // Semitransparent
END_VAR
```

3. In the device tree, declare local color variables in `PLC_PRG`.

   ⇨
   ```
   VAR
       dwFillColor: DWORD := GVL.gc_dwGreen;
       dwFrameColor : DWORD := GVL.gc_dwBlack;
       dwAlarmColor : DWORD := GVL.gc_dwRed;
   END_VAR
   ```

4. Declare a control variable.

   ⇨ `bChangeColor : BOOL;`

5. Declare an input variable in `PLC_PRG`.

   ⇨ `bInput : BOOL;`

6. Enable the visualization editor.
7. Drag a *"Rectangle"* element to the visualization editor.

   ⇨ The *"Properties"* view of the element opens.

8. Configure the properties of the rectangle as follows:
   - Property *"Color variables"*, *"Normal state"*, *"Filling color"*: `PLC_PRG.dwFillColor`
   - Property *"Color variables"*, *"Normal state"*, *"Frame color"*: `PLC_PRG.dwFrameColor`
   - Property *"Color variables"*, *"Alarm state"*, *"Filling color"*: `PLC_PRG.dwAlarmColor`
   - Property *"Color variables"*, *"Toggle color"*: `<toggle/tap variable>`
   - Property *"Input configuration"*, *"Toggle"*, *"Variable"*: `PLC_PRG.bInput`
9. Program the variables as follows:

   ```
   PROGRAM PLC_PRG
   VAR
       dwFillColor: DWORD := GVL.gc_dwGreen;
       dwFrameColor : DWORD := GVL.gc_dwBlack;
       dwAlarmColor : DWORD := GVL.gc_dwRed;

       bChangeColor : BOOL;
       bInput : BOOL;
   END_VAR

   IF bChangeColor = TRUE THEN
       dwFillColor := GVL.gc_dwYellow;
       dwFrameColor := GVL.gc_dwBlue;
   ELSE
       dwFillColor:= GVL.gc_dwGreen;
       dwFrameColor := GVL.gc_dwBlack;
   END_IF
   ```

   ⇨ The colors are initialized at runtime. If the variable `bChangeColor` is then forced to `TRUE`, the color display of the rectangle changes. When the rectangle is clicked in the visualization, the rectangle is displayed in alarm colors.

See also

- *Chapter 1.4.5.8 "Animating visualization elements" on page 1293*
- *Chapter 1.4.5.19.4.2 "Object 'Visualization manager'" on page 1777*
- *Chapter 1.4.5.17 "Applying Visualization Styles" on page 1360*
- *Chapter 1.4.5.8.2 "Animating a text display" on page 1295*

1.4.5.9 Displaying data arrays in tables

A frequently required function of a user interface is the display of data arrays. CODESYS Visualization provides the element *“Table”* for this.

In the configuration of the element *“Table”*, enter an array variable in the property *“Data array”*. The array components are displayed in the rows and columns of the table.

A table for displaying data arrays can also be created in the following way. You duplicate a single element having at least one property that is described by a structured variable. The single element is configured as a "template" for this and duplicated with a command.

1.4.5.9.1 Displaying Array Variables in Tables

A frequently required function of a user interface is the display of data arrays. CODESYS Visualization provides the element *“Table”* for this.

In the configuration of the element *“Table”*, enter an array variable in the property *“Data array”*. The array components are displayed in the rows and columns of the table.

Subsequent instructions describe an example of how an array of a structure is displayed in a table. As a preparation, create the `MYSTRUCT` DUT and the declarations in the `PLC_PRG` program.

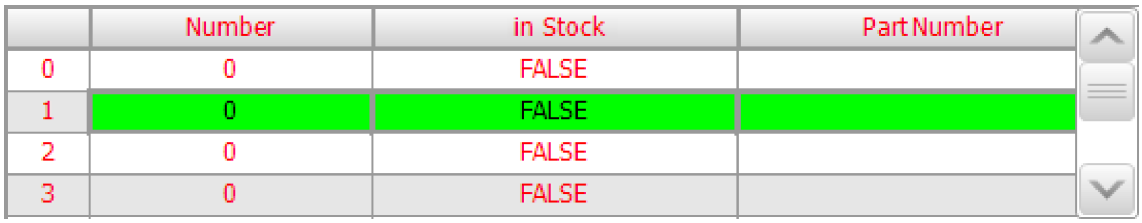
```
TYPE MYSTRUCT :
    STRUCT
        iNo : INT;
        bOnStock : BOOL;
        strPartNumber : STRING;
    END_STRUCT
END_TYPE
```

```
PROGRAM PLC_PRG
VAR
    arrStruct : ARRAY[0..6] OF MYSTRUCT;
    iSelectedColumn : INT;
END_VAR
```

1. Drag the *“Table”* visualization element to the visualization editor.
2. Assign the array variable `arrStruct` to the *“Data array”* property.
   ⇨ The structure members are displayed as column headings and the array index as row headings.
3. Change the *“Columns ➔ Column ➔ [0] ➔ Column header”* property to an informative heading (example: `Number`).
4. Change the heading of column [1] to `in stock` and column [2] to `Part number`. Adjust the column width.
5. Assign a color to the *“Selection ➔ Selection color”* property.
6. Define the *“Selection ➔ Selection type”* property as `Row selection`.

7. In the *“Selection ➔ Variable for selected row”* property, define the `PLC_PRG.iSelectedColumn` variable.

   ⇨ The following display results in online mode:



	Number	in Stock	Part Number
0	0	FALSE	
1	0	FALSE	
2	0	FALSE	
3	0	FALSE	

See also

- *Chapter 1.4.5.18.1.13 “Visualization Element 'Table'” on page 1485*

1.4.5.9.2 Configuring and Multiplying Visualization Elements as Templates

A table can also be created to display data arrays in the following way. You multiply a single element that has at least one property which is described by a structured variable. To do this, the single element is configured as a "template" and multiplied by means of a command.

You can use the *“Visualization ➔ Multiply Visu Element”* command to display array data in a visualization. The command multiplies a template element to create an element of the same type for each array component. The layout of the new elements in the visualization is one-dimensional as a row or column, or two-dimensional as a table.

To do this, drag an applicable element into the visualization editor. Then configure the properties of the element with array variables and specify the index access placeholder `$FIRSTDIM$` as component access. If you have declared a multidimensional array, then you can use the second index access placeholder `$SECONDDDIM$` for the additional dimension. Configure the remaining properties as usual with the typical values. The purpose is to create a valid template element. Then execute the *“Multiply Visu Element”* command on the template element. Now the dialog with the same name opens. There you define in detail how many elements should be created and where they should be located.

After multiplying, the visualization contains as many of the same elements as are indexed using placeholders. In doing so, the settings in the *“Multiply Visu Element”* dialog are taken into consideration. All new elements in the properties that were preset with placeholders have these replaced with precise indexes. The remaining properties have been applied and copied without changes.

For example, you can have a layout of nine buttons as 3x3 tables, which are all the same size or the same color, but vary in the labeling. The labels are declared as a string array (nine components) and are passed as a value to the *“Texts”->“Text”* property.

Valid template element:

- Declaration of array variables
  Example: `asText: ARRAY[1..3, 1..3] OF STRING;`
- Element with applicable element type

- Configuration of at least one property of the applicable element with array variables with index access placeholders
  Example: Property *“Texts”*, *“Text”* = `PLC_PRG.asText[$FIRSTDIM$, $SECONDDIM$]`
  This is possible for all properties that permit a variable as a value (for example, also properties from the "Animation" or "Input" categories. To configure multiple properties for an element with arrays and index access placeholders, all arrays must have the same structure with the same dimension. The declarations have to be compatible.
- Configuration of properties that do not vary (and are therefore the same for all generated elements) with the usual values without index access placeholders
  Example:
  `sButtonTip : STRING := 'This element is created by multiplication'`
  Property *“Texts”*, *“Tooltip”* = `%s`
  Property *“Text variables”*, *“Tooltip variable”* = `sButtonTip`

You can still use the placeholder `%` as usual for the text display of variable values in the properties in “Texts”.

Applicable visualization elements

Visualization elements that can be multiplied:

- *“Rectangle”*
- *“Rounded Rectangle”*
- *“Ellipse”*
- *“Line”*
- *“Polygon”*
- *“Polyline”*
- *“Bézier Curve”*
- *“Image”*
- *“Frame”*
- *“Button”*
- *“Pie”*
- *“Spin Box”*
- *“Text Field”*
- *“Check Box”*
- *“Image Switcher”*
- *“Lamp”*
- *“Dip Switch”*
- *“Power Switch”*
- *“Push Switch”*
- *“Push Switch LED”*
- *“Rocker Switch”*
- *“Rotary Switch”*

Configuring and multiplying lamps and buttons as templates

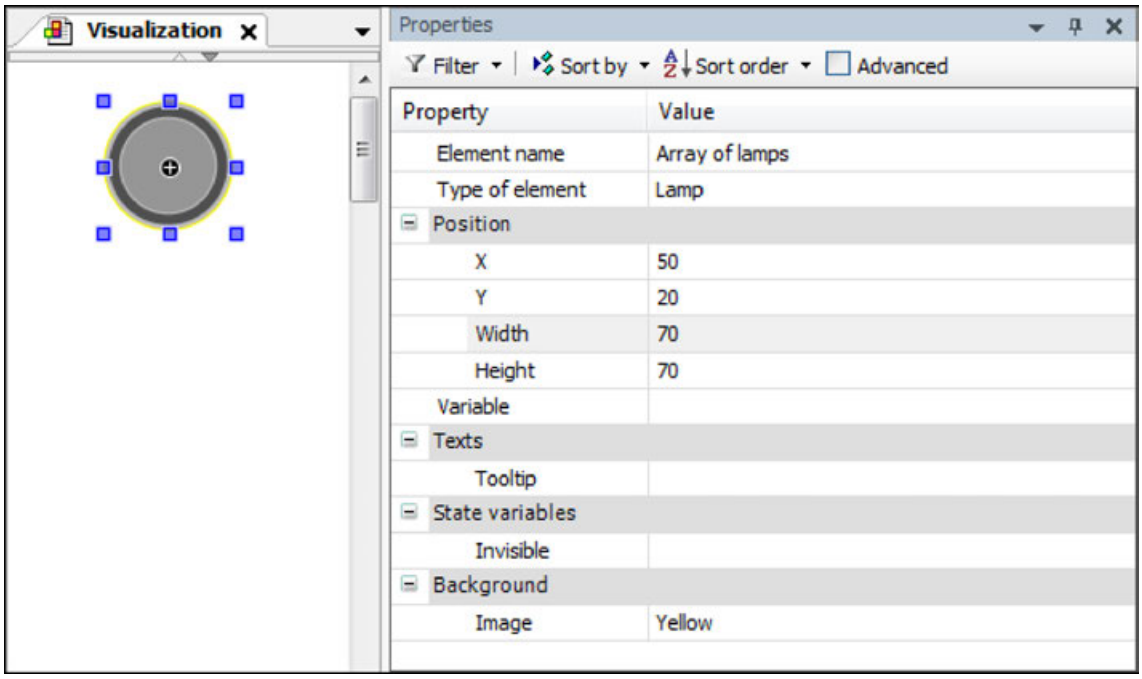
1. Create a new standard project.
   ⇨ A CODESYS Control Win V3 is configured as the device. The `MainTask` calls `PLC_PRG`. The implementation language is ST.

2. In `PLC_PRG` in the program code, declare array variables with basic data type `STRING`.

   ⇨

```
PROGRAM PLC_PRG
VAR
    axLampIsOn: ARRAY[1..2,1..3] OF BOOL;  // For lamp,
property 'variable' and button, user input
    asButtonText: ARRAY[1..2,1..3] OF STRING := // Output text
for button, property 'text variables''text variable'
    [
        '1A Lamp', '2A Lamp',
        '1B Lamp', '2B Lamp',
        '1C Lamp', '2C Lamp'
    ];
END_VAR
```

3. Select the application in the device tree and click *"Add Object ➔ Visualization"*.
4. In the *"Add Visualization"* dialog, specify the name `VisuMain` and click *"Add"* to close the dialog.
5. Drag a *"Lamp"* element from the *"Visualization Toolbox"* view to the visualization.
6. Configure the fixed property values.



7. Double-click the value field of the *"Variable"* property.

   ⇨ The line editor opens.

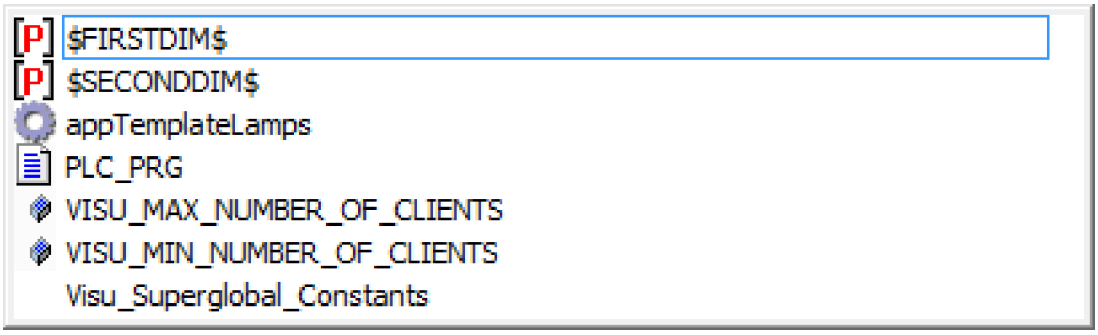
8. Click ...

   ⇨ The Input Assistant opens.

9. Select the array variable `PLC_PRG.axLampIsOn` from the variable tree.

10. Extend the string at the end, for example with "`[f`".

    ⇨ If you have activated SmartCoding (*"Options"* dialog, *"SmartCoding"* category, *"List components immediately when typing"* option), then the current variable list appears with the placeholders:



11. Select the placeholder `$FIRSTDIM$` for the first dimension and confirm the selection.
12. Extend the string at the end, for example with "`,s`".

    ⇨ The variable list appears again.
13. Select the placeholder `$SECONDDIM$` for the second dimension and confirm the selection.
14. Complete the string with a closing bracket.

    ⇨ `PLC_PRG.axLampIsOn[$FIRSTDIM$, $SECONDDIM$]`

    The lamp is configured as a template.
15. Click *"Visualization ➔ Multiply Visu Element"*.

    ⇨ The *"Multiply Visu Element"* dialog opens. The default values are derived from the array declarations.

    "Total number of elements", *"Horizontal"* = `2`

    "Total number of elements", *"Vertical"* = `3`
16. Declare the distance between the new elements.

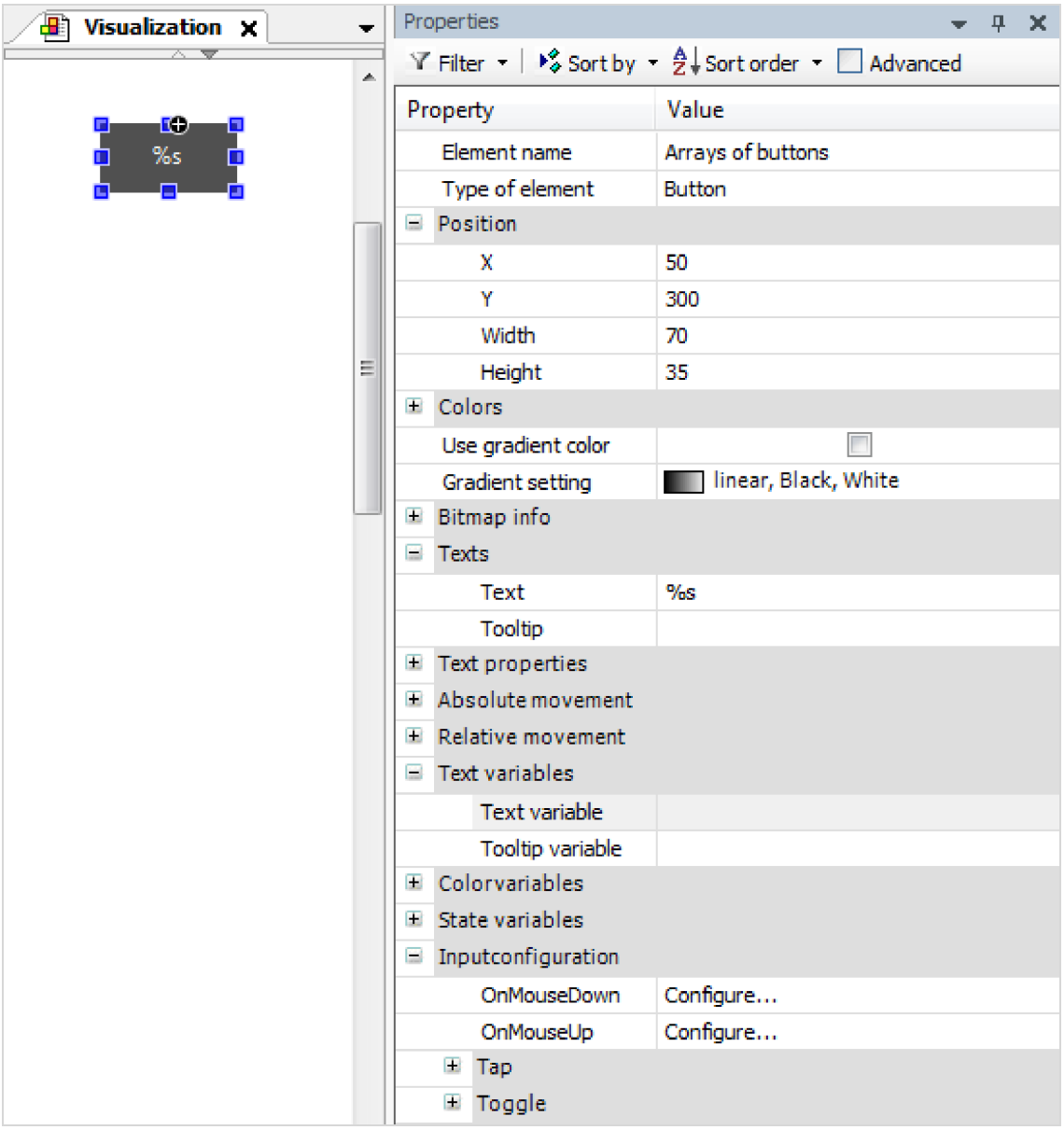
    ⇨ *"Offset between elements"*, *"Horizontal"* = `3`

    "Offset between elements", *"Vertical"* = `3`
17. Check the advanced settings.
18. Click *"OK"* to confirm the selection.

    ⇨ The new elements appear in the visualization editor. All properties are configured with a precise index and the array variables are indexed.
19. In the *"Visualization Toolbox"*, in the *"Common Controls"* category, drag the *"Button"* element to the visualization editor.

    ⇨ The *"Properties"* view of the element opens.

20. Configure the fixed property values.



21. Configure the value for the *“Text variables”->“Text variable”* property.

    ⇨ `PLC_PRG.asButtonText[$FIRSTDIM$, $SECONDDIM$]`

22. Configure the value for the *“Input configuration”->“Toggle”->“Variable”* property.

    ⇨ `PLC_PRG.axLampIsOn[$FIRSTDIM$, $SECONDDIM$]`

    The button is configured as a template.

23. Click *“Visualization ➔ Multiply Visu Element”*.

    ⇨ The *“Multiply Visu Element”* dialog opens. The default values are derived from the array declarations.

    “Total number of elements”, *“Horizontal”* = `2`

    “Total number of elements”, *“Vertical”* = `3`

24. Declare the distance between the new elements.

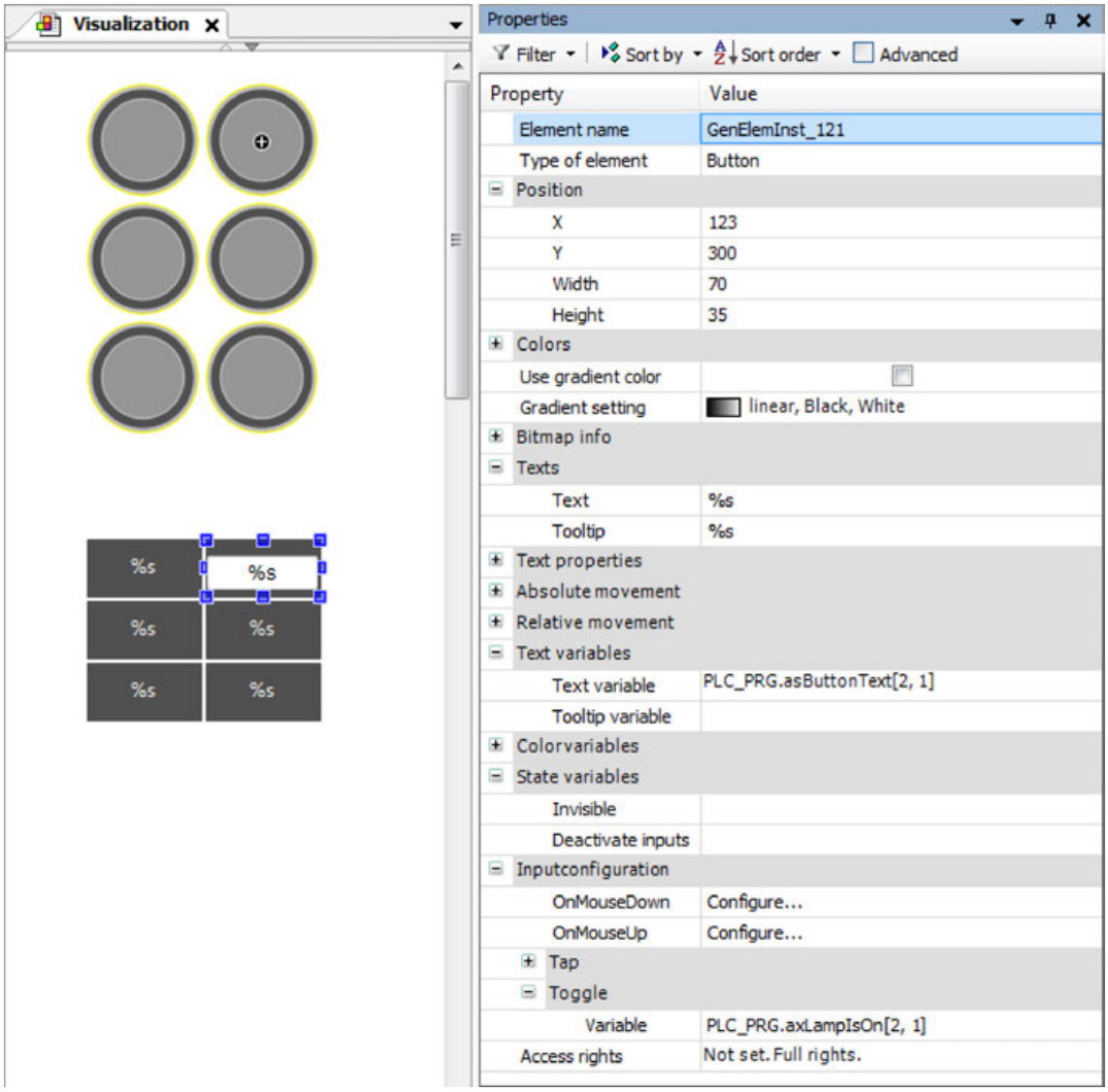
    ⇨ *“Offset between elements”*, *“Horizontal”* = `3`

    “Offset between elements”, *“Vertical”* = `3`

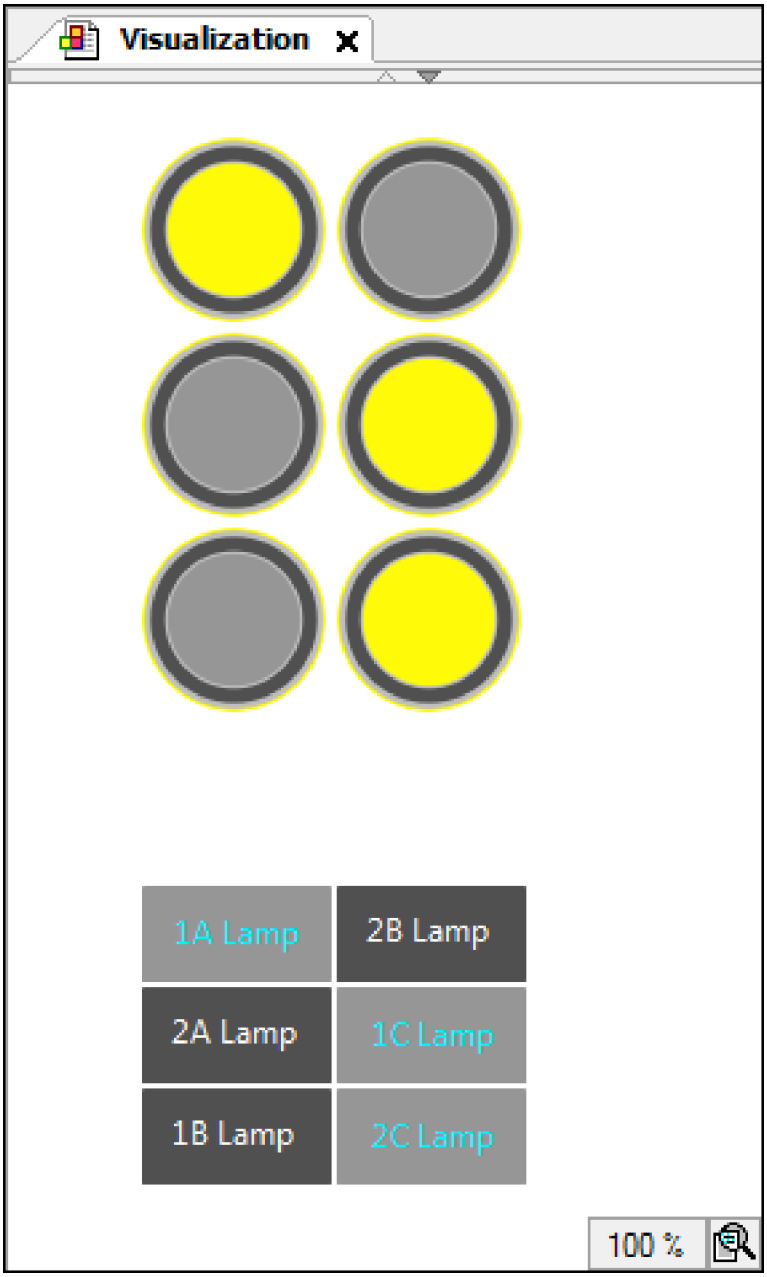
25. Check the advanced settings.

26. Click *“OK”* to confirm the selection.

    ⇨ The new elements appear in the visualization editor. All properties are configured with a precise index and the array variables are indexed.



27. Build, start, and download the application.
    ⇨ Visualization at runtime:



Array variable with more than two dimensions

You can also configure the template element with array variables that have more than two dimensions, but you can only assign placeholders to a maximum of two of the dimensions. In the additional dimensions, the indexes are fixed.

Example

Declaration

```
PROGRAM PLC_PRG
VAR
 asText: ARRAY[1..2, 1..3, 1..6, 1..2] OF STRING;
END_VAR
```

Configure the *“Text variables”*, *“Tooltip variable”* property for the template element:

```
PLC_PRG.asText[2, $FIRSTDIM$, $SECONDDIM$, 2]
```

Layout of a one-dimensional array in a table

You can configure the template element with a one-dimensional array by means of the index access placeholder `$FIRSTDIM$`. If the number of new elements to be created is greater than five, then a tabular layout is preset in the *"Multiply Visu Element"* dialog. The layout of the new elements is as quadratic as possible.

Example

```
PROGRAM PLC_PRG
VAR
 asText: ARRAY[1..100] OF STRING;
END_VAR
```

The default setting in the *"Multiply Visu Element"* dialog allows for a layout of 100 new elements in a 10x10 field.

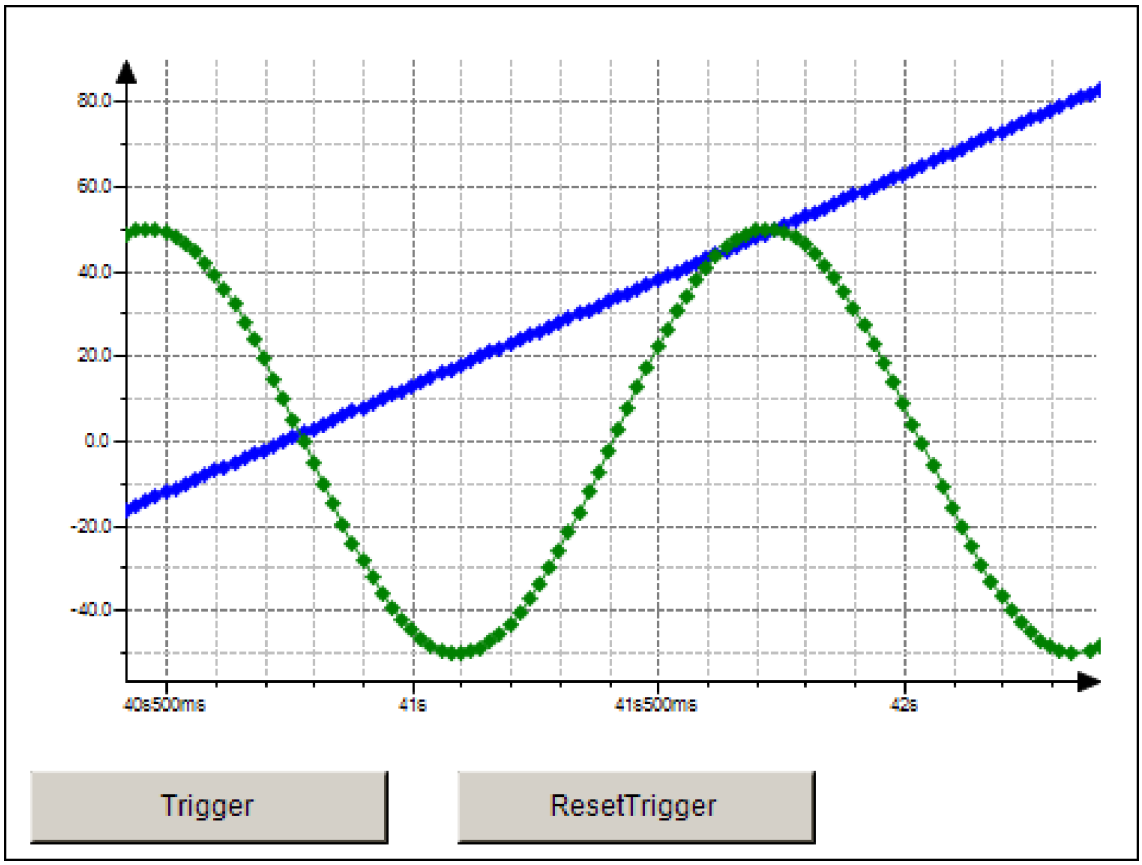
See also

- *Chapter 1.4.5.19.2.11 "Command 'Multiply Visu Element'" on page 1729*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *Options for SmartCoding*

1.4.5.10 Displaying data curve with trace

With this element, you can integrate a trace graph in the visualization that monitors and displays variable values permanently. You configure the displayed trace graph in the element properties. In addition, you can add control elements that control the trace functionality. This is done manually or by using the *"Insert Elements for Controlling Trace"* command.

Configurations for the 'Trace' visualization element can be taken from the 'Trace' object.



See also

- *Chapter 1.4.5.10.1 “Getting started with trace” on page 1307*
- *Chapter 1.4.5.19.2.15 “Command 'Insert Elements for Controlling Trace'” on page 1737*
- *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*

1.4.5.10.1 Getting started with trace

Create a project with the following program PLC_PRG:

```
PROGRAM PLC_PRG
VAR
    iVar : INT;
    rSin : REAL;
    rVar : REAL;
END_VAR

iVar := iVar + 1;
iVar := iVar MOD 33;

rVar := rVar + 0.1;
rSin := 30 * SIN(rVar);
```

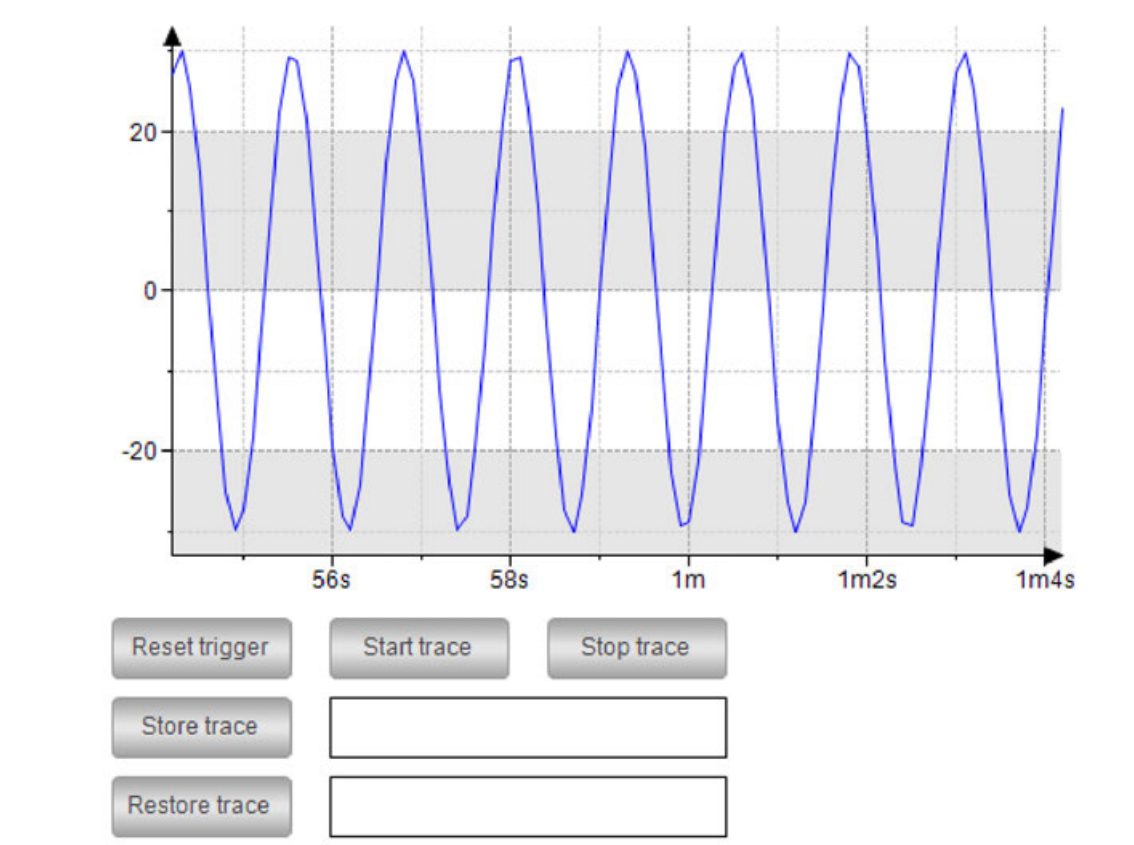
1. In the device tree, select the application and add a new visualization by clicking *“Project ➔ Add Object ➔ Visualization”*.
   ⇨ The respective visualization editor opens.
2. Add the *“Visualization”* object to the device tree below *“Application”*.
   ⇨ An empty visualization appears.
3. Open *“Toolbox ➔ Special Controls”*.
4. Drag the *“Trace”* element to the visualization editor.
   ⇨ The element properties are displayed on the right side.

5. Click the symbol in the *"Trace"* property.
   ⇨ The *"Trace Configuration"* dialog opens.
6. Click *"Add Variable"* to add an entry to the tree view of the trace configuration and select a project variable (for example, `PLC_PRG.rSin`).
7. Click the top node of the trace configuration.
   ⇨ The group *"Record Settings"* is shown on the right.
8. Select the `MainTask` option for the *"Task"* setting.
   Tip: The trace recording and the corresponding program should be executed in the same task.
9. Click *"OK"*.
   ⇨ The task configuration is applied.
10. Select the trace element and click *"Visualization ➔ Add Elements for Trace Control"*
    ⇨ The *"Trace Wizard"* dialog opens. By default, all control elements are activated there.
11. Click *"OK"* to close the dialog.
    ⇨ The control elements are added to the visualization and the control variables are declared. Then the control elements and the trace element are configured with the control variables.
12. Download the application to the controller and start it.

Example

Record the sine-shaped data of the IEC variable `PLC_PRG.rSin`

The `PLC_PRG` program is running on the PLC. When you follow the "Getting Started" instructions, the following interface is displayed:



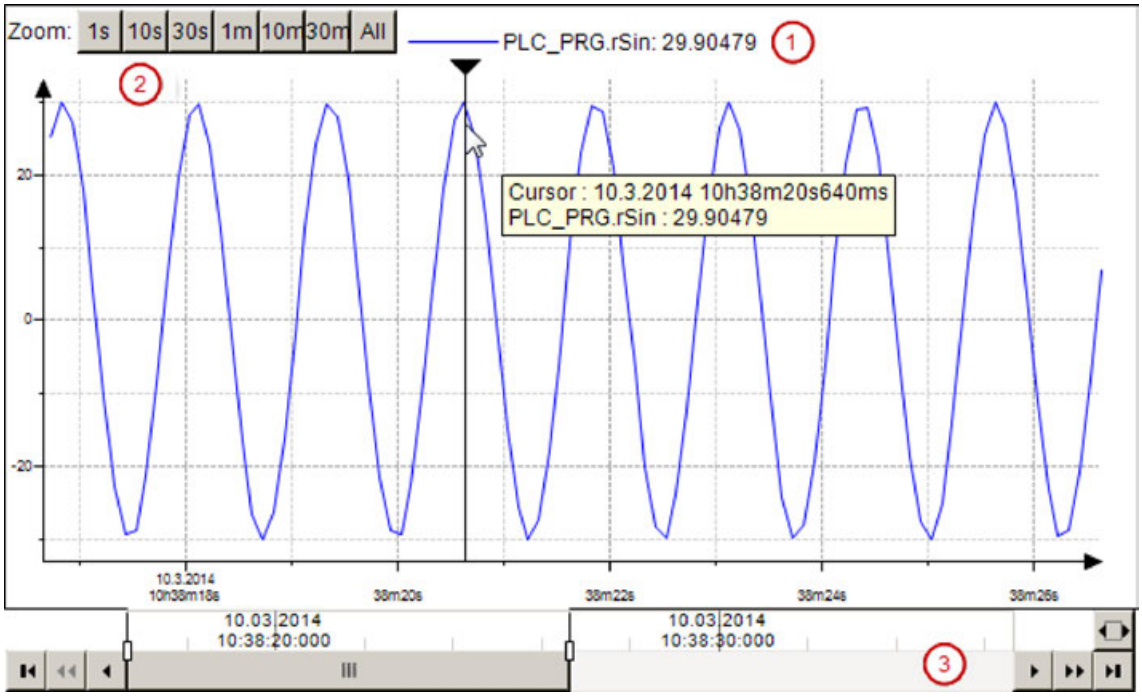
You can control the trace recording by clicking the buttons.

See also

- *Chapter 1.4.5.19.2.13 “Command 'Configure Trace'” on page 1734*
- *Chapter 1.4.5.19.3.19 “Dialog 'Display Settings'” on page 1770*
- *Chapter 1.4.5.18.1.34 “Visualization Element 'Trace'” on page 1619*

1.4.5.11 Displaying data curve with trend

A trend visualizes data that is used in the database of a trend recording. In contrast to the trace element, the trend element is particularly appropriate for long-term data recording.



The visualization of a trend encompasses the Trend element and the control elements. The three possible control elements can be seen in the illustration.

- Legend ①: Outputs the trend variables with values.
- Time range picker ②: Provides buttons for selecting predefined time ranges.
- Date range picker ③: encompasses control elements for navigation and zooming in the historical and current data on basis of the set date range.

A cursor is optionally available that enables the reading of a value at a certain time.

You can execute a trend visualization in the following clients:

- Target visualization
- Integrated visualization

See also

- *Chapter 1.4.5.11.1 “Getting Started with Trend Visualization” on page 1309*
- *Chapter 1.4.5.18.1.35 “Visualization Element 'Trend'” on page 1625*
- *Chapter 1.4.5.18.1.45 “Visualization Element 'Date Range Picker'” on page 1680*
- *Chapter 1.4.5.18.1.46 “Visualization Element 'Time Range Picker'” on page 1685*

1.4.5.11.1 Getting Started with Trend Visualization

When you execute a Trend, it is best to proceed with user guidance and the help of the trend wizard.

Development of a visualization with trend

1. Create an empty standard project and program at least one variable into `PLC_PRG`.
   ⇨ `PLC_PRG` is declared and implemented
2. Add the *"Visualization"* object to the device tree below *"Application"*.
   ⇨ An empty visualization appears.
3. Open *"Toolbox ➔ Special Control"*.
4. Drag the *"Trend"* element to the visualization
   ⇨ The *"Trend Recording"* dialog opens with the *"Recording Settings"*.
5. Select the task in which the trend recording will be executed.

   > *In general the trend recording runs in the same task as the main program, i.e. PLC_PRG.*

   Therefore, select `MainTask`.
6. Add a trend variable with *"Add Variable"* and assign an IEC variable from PLC_PRG to the trend variable.
7. Click *"OK"* to close *"Trend Configuration"*.
   ⇨ There is a newly created object of the type Trend recording under *"Trend Recording Manager"*. The active visualization contains a *"Trend"* element that is selected.
8. Click *"Visualization ➔ Insert Elements for Controlling Trend Elements"*.
   ⇨ The *"Trend Wizard"* dialog box opens.
9. By default, all three control elements are activated in the dialog. Click *"OK"* to close the dialog box.
   ⇨ The active visualization contains a *"Trend"* with control elements.
10. Set the application containing the trend objects to active.
11. Compile the application with *[F11]*.
12. Click *"Online ➔ Login"*.
13. Start the application with *[F5]*.
    ⇨ The target visualization appears. The visualization contains the trend diagram with the value curve of the variable. The control elements enable user inputs.

See also

- *Trend recording*
- *Chapter 1.4.5.11.2 "Programming a Trend Visualization" on page 1312*
- *Chapter 1.4.5.19.2.18 "Command 'Insert Elements for Controlling the Trend'" on page 1739*

Example: Visualization of the sinusoidal trend of an IEC variable.

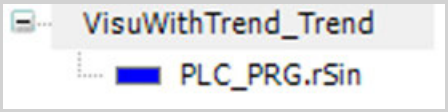
The following objects are implemented in the project:

- `PLC_PRG`
- `Visualization_Trend1`
- `VisuWithTrend`

PLC_PRG `PLC_PRG` runs as part of the application on the controller.

```
PROGRAM PLC_PRG
VAR
    iVar : INT;
    rSin : REAL;
    rVar : REAL;
END_VAR

iVar := iVar + 1;
iVar := iVar MOD 33;

rVar := rVar + 0.1;
rSin := 30 * SIN(rVar);
```

Visualization_Trend1 `Visualization_Trend1` is the object that contains the configuration of the trend recording.



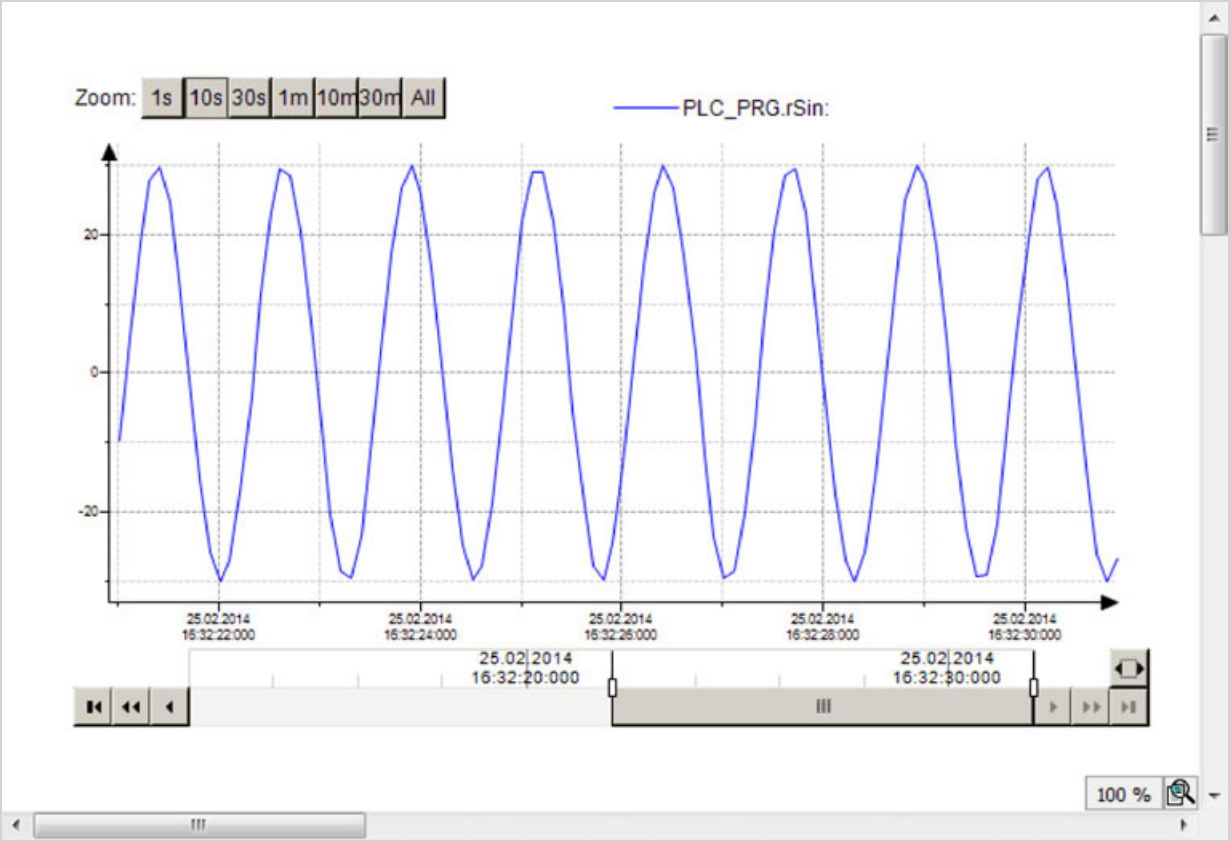
VisuWithTrend

`VisuWithTrend` is the object that visualizes the trend.

The visualization contains four elements: one *"Trend"* and three control elements. The properties of the trend are defined as follows.

Properties	Value
"Trend recording"	Visualization_Trend1
"Display cursor"	☑
"Display tool tip"	☑
"Show frame"	☑
"Date Range Picker"	Trend1DateRangeSelector
"Time Picker"	Trend1TimeSelector
"Legend"	Trend1Legend

`VisuWithTrend` at runtime



1.4.5.11.2 Programming a Trend Visualization

To display a trend recording in the visualization, you define which application provides which trend recording. You define the display by means of the *"Properties"* of the trend element and the controls used.

Defining the application and data source

The visualization task and the trend recording task generally run under the same application. If this is not the case, then the application containing the visualization task requires a data source manager.

1. Select a trend element in the active visualization editor.
   ⇨ The properties of the trend element are displayed on the right side.
2. Double-click the value field *“Properties ➔ Application”*.
3. Use the Input Assistant (...) to select the application. You can also specify the name of the application directly.

See also

- *Trend Recording*
- *Data Source Manager*

Adding a control

1. Select a trend element in the active visualization editor.
2. Click *“Visualization ➔ Insert Elements for Trend Controlling”*.
   ⇨ The *“Trend Wizard”* dialog opens.
3. Select the desired control. Examples: *“Date Range Picker”*, *“Time range Range Picker”*, *“Legend”*. Click *“OK”* to confirm.
   ⇨ The selected controls are inserted for the trend element. You can move them to any position you like. In the *“Properties”* of the trend element, the controls are shown below *“Assigned controls”*.

See also

- *Chapter 1.4.5.19.2.18 “Command 'Insert Elements for Controlling the Trend'” on page 1739*

Defining the trend recording to visualize

1. Select a trend element in the active visualization editor.
   ⇨ The properties of the trend element are displayed on the right side.
2. Click the value field of *“Properties ➔ Trend recording”*
   ⇨ *“Select trend recording”* is displayed. The trend recordings available application-wide are listed under *“Available trend recordings”*.
3. Select a trend record below *“Available trend recordings”*.
4. Click .
   ⇨ The trend recording is located under *“Selected trend recording”*.
5. Click *“OK”* to confirm the entry.
   ⇨ The selected trend recording is listed in *“Values”* in *“Properties ➔ Trend recording”*.

See also

- *Trend Recording*

Removing a control

A control that was added with the help of the "Trend Wizard" cannot be deleted via the Trend wizard dialog.

1. Select a control of a trend in the active visualization editor.
2. Press *[Del]* or *"Delete"* to delete the element.
3. Select the trend in the active visualization editor.
4. Delete the assigned value in *"Properties ➔ Assigned controls ➔ <control>"*.

NOTICE!
It is absolutely necessary to delete this reference manually. The property is not deleted automatically by deleting the control.

Configuring the coordinate system of the trend diagram

1. Select a trend in the active visualization editor.
2. Use the *"Visualization ➔ Configure Trend Display Settings"* command.
   ⇨ The *"Display Settings"* dialog opens.
3. Adapt the settings as needed.

See also

- *Chapter 1.4.5.19.2.18 "Command 'Insert Elements for Controlling the Trend'" on page 1739*

Reading a trend value at runtime

1. Open *"View ➔ Element Properties"*.
2. Select a trend element in your visualization.
   ⇨ The properties of the trend element are displayed on the right side.
3. Select the *"Properties ➔ Show cursor"* option and *"Show tooltip"*.
   ⇨ A cursor is drawn in the coordinate system.
4. Select the *"Properties ➔ Show tooltip"* option.
5. Download the application to the controller and start the application.
6. *If the diagram "runs", then the date range has been placed in such a way that its end time is the current time.*

   Select the date range so that the diagram does **not** run. If necessary, drag the scroll bar to an earlier date range.
   ⇨ A cursor is available. The tooltip of the cursor informs you of the trend values. For each trend variable, the legend displays the value at the point in time at which the cursor is positioned.

Deleting the trend recording history

You can insert an input element in the visualization which the operator can use to delete the previous value recording in the trend visualization at runtime. The curve displayed until then is removed and the display starts over.

1. In the application (example: in the program `PLC_PRG`), implement the following code:

```
itfTrendRecording : ITrendRecording;
itfTrendStorageWriter : ITrendStorageWriter;
itfTrendStorageWriter3 : ITrendStorageWriter3;
sTrendRecordingName : STRING := 'TrendRecording';
itfTrendRecording :=
GlobalInstances.g_TrendRecordingManager.FindTrendRecording(ADR(sTr
endRecordingName));
xClearHistoryTrend: BOOL;

IF xClearHistoryTrend THEN
itfTrendRecording :=
GlobalInstances.g_TrendRecordingManager.FindTrendRecording(ADR(sTr
endRecordingName));
IF itfTrendRecording <> 0 THEN
    itfTrendStorageWriter :=
itfTrendRecording.GetTrendStorageWriter();
    IF __QUERYINTERFACE(itfTrendStorageWriter,
itfTrendStorageWriter3) THEN
itfTrendStorageWriter3.ClearHistory();
    END_IF
END_IF
```

2. In the visualization of the trend recording, add a button for deleting the previous curve. Configure its *"Toggle"* property with the variable `PLC_PRG.xClearHistoryTrend`.

   ⇨ When `xClearHistoryTrend` is set to `TRUE`, the previously recorded curve is deleted. The recording immediately starts again.

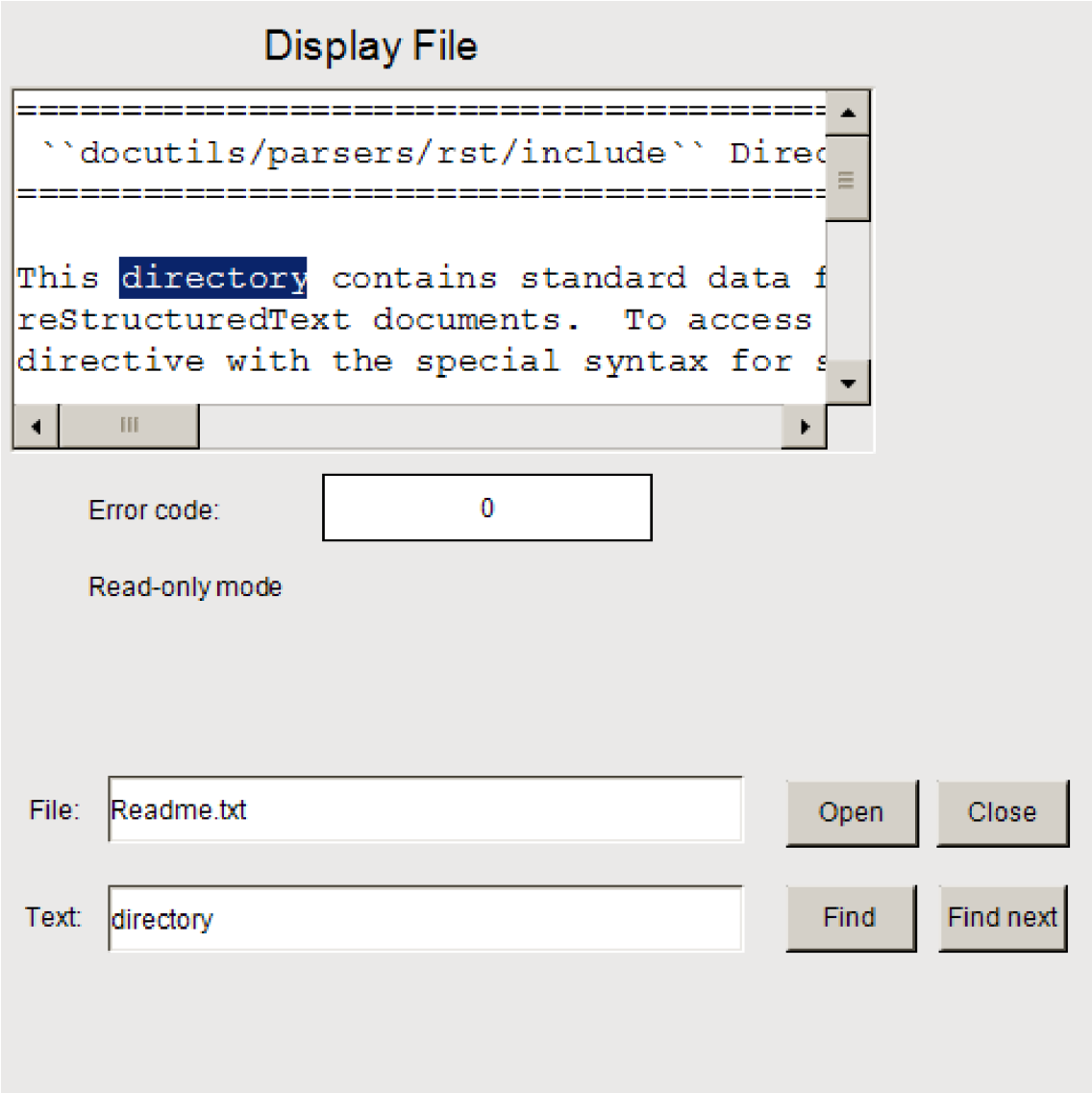
1.4.5.12 Displaying and Editing Text Files

With the help of the element *"Text Editor"* you can display a text file in the user interface and optionally also enable the user to edit the file.

1.4.5.12.1 Configuring the Display of a Text File

In order to display a text file that is located on the controller, you need not only the element *"Text Editor"*, but also control elements for selecting, opening and closing the file. Optionally a text search function can be set up in the file with further control elements.

Example:



Configuring the element *"Text Editor"*, example

1. Drag an element *"Text Editor"* into the visualization editor.
2. Declare the control variables for the element, for example as global variables in the `GVL` object.

   ⇨ Refer to the declaration of the control variables for this.
3. For the *"Text Editor"*, configure the property *"Editing mode"* with *"Read only"*.
4. Also configure the property *"Control variables"*.

   Assign the following variables there:

   - *"Control variables ➔ File ➔ Variable"* with `g_sFileName`
   - *"Control variables ➔ File ➔ Open"* with `g_bFileOpen`
   - *"Control variables ➔ File ➔ Close"* with `g_bFileClose`
   - *"Control variables ➔ File ➔ New ➔ Variable"* with `g_bFileNew`
   - *"Control variables ➔ File ➔ Save ➔ Variable"* with `g_bFileSave`
   - *"Control variables ➔ Edit ➔ Variable"* with `g_sEditSearchFor`
   - *"Control variables ➔ Edit ➔ Find"* with `g_bEditFind`
   - *"Control variables ➔ Edit ➔ Find next occurrence"* with `g_bEditFindNext`

Declaring the control variables

```
VAR_GLOBAL
                    g_sFileName: STRING := 'Readme.txt';
                    g_bFileOpen : BOOL;
                    g_bFileClose: BOOL;
                    g_bFileNew: BOOL;
                    g_bFileSave: BOOL;
                    g_sEditSearchFor : STRING;
                    g_bEditFind : BOOL;
                    g_bEditFindNext : BOOL;

                    g_usiErrorHandlingVarForErrorCode: USINT;
                    g_bVarForContentChanged : BOOL;
                    g_bVarForReadWriteMode: BOOL;
                    END_VAR
```

Configuring control elements for the file selection

1. Add an element *“Label”*.
2. Configure the property *“Texts ➔ Text”* with `File:`.
3. Add an element *“Rectangle”* next to it, in which the user can then enter the file name:
4. Configure the property *“Texts ➔ Text”* with `%s`:
5. Configure the property *“Texts ➔ Text variable”* with `g_sFileName`.
6. Configure the property *“Input configuration ➔ OnMouseclick”* with *“Write a variable”*.
   In the dialog *“Input Configuration”*, select *“Text input”* as the *“Input type”*.
   Activate the option *“Use text output variable”*.
   ⇨ The rectangle for the input of the file name is configured.
7. Add an element *“Button”* for opening the file.
8. Configure the property *“Texts ➔ Text”* with `Open`:
9. Configure the property *“Input configuration ➔ OnMouseclick”* with *“Toggle a variable”*.
   Assign `g_bFileOpen` as a variable.
   ⇨ The button `Open` is configured.
10. Add a further element *“Button”* for closing the file.
11. Configure the property *“Texts ➔ Text”* with `Close`:
12. Configure the property *“Input configuration ➔ OnMouseclick”* with *“Toggle a variable”*.
    Assign `g_bEditFile` as a variable.
    ⇨ The button `Close` is configured.

Control elements for searching for a text.

1. Add an element *“Label”*.
2. Configure the property *“Texts ➔ Text”* with `Text:`.
3. Alongside it, add an element *“Rectangle”* for the input of the text to be found.
4. Configure the property *“Texts ➔ Text”* with `%s`:
5. Configure the property *“Texts ➔ Text variable”* with `g_sEditSearchFor`.
6. Configure the property *“Input configuration ➔ OnMouseclick”* with *“Write a variable”*.
   In the dialog *“Input Configuration”*, select *“Text input”* as the *“Input type”*.
   Activate the option *“Use text output variable”*.
   ⇨ The rectangle is configured.
7. Add an element *“Button”* for starting the search.
8. Configure its property *“Texts ➔ Text”* with `Find`.

9. Configure the property *"Input configuration ➔ OnMouseclick"* with *"Toggle a variable"*.

   Assign `g_bEditFind` as a variable.

10. Also add the action *"Execute ST-Code"*.

    Program the action with: `g_bEditFindNext := FALSE;`

    ⇨ The button is configured.

11. Add a further element *"Button"*.
12. Configure the property *"Texts ➔ Text"* with `Find next`.
13. Configure the property *"Input Configuration ➔ OnMouseclick"* with *"Toggle a variable"*.

    Assign `g_bEditFind` as a variable.

14. Also add the action *"Execute ST code"*.

    Program: `g_bEditFindNext := TRUE;`

    ⇨ The button is configured.

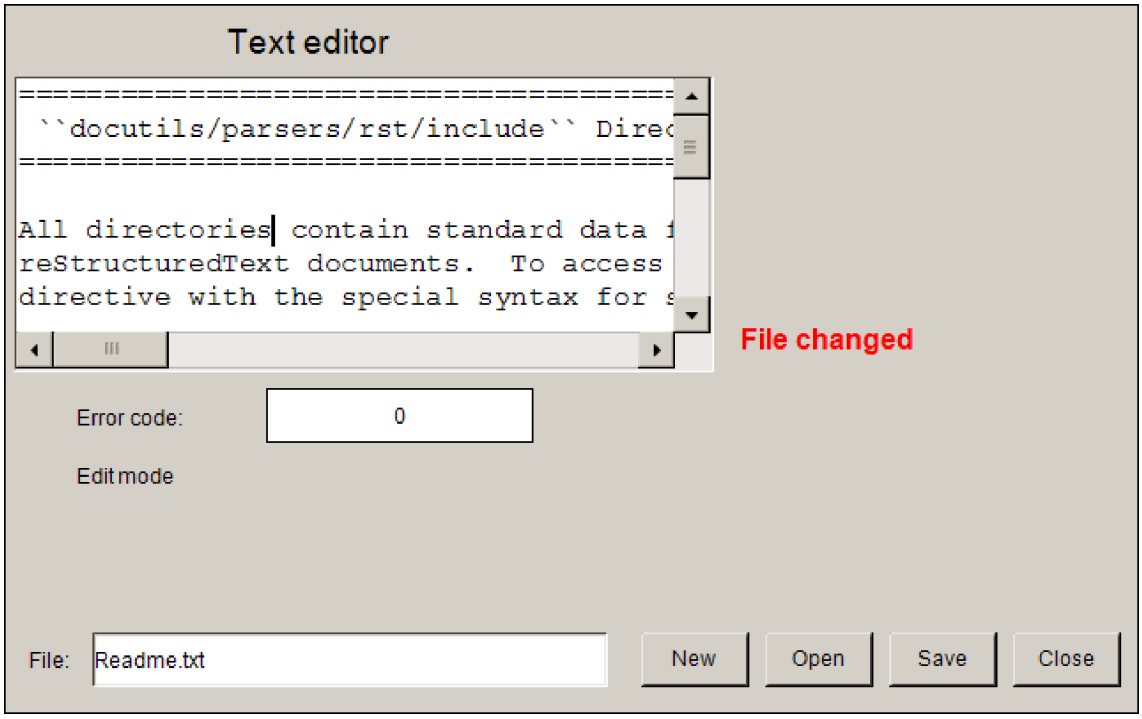
See also

- *Chapter 1.4.5.18.1.41 "Visualization Element 'Text Editor'" on page 1653*

1.4.5.12.2 Configuring the Editing of a Text File

In order to be able to create a new text file or edit an existing one on the controller with the *"Text Editor"* in the user interface, you need not only the element *"Text Editor"*, but also control elements for selecting, opening, closing, saving and creating a file.

Example:



Configuring the element *"Text Editor"*, example:

1. Drag an element *"Text Editor"* into the visualization editor.
2. Declare the control variables for the element, for example as global variables in the `GVL` object.

   ⇨ Refer below to the declaration of the control variables for this.

3. For the *"Text Editor"*, configure the property *"Editing mode"* with *"Read/Write"*.

4. Also configure the property *"Control variables"*.

   Assign the following variables there:

   - *"Control variables ➔ File ➔ Variable"* with `g_sFileName`
   - *"Control variables ➔ File ➔ Open"* with `g_bFileOpen`
   - *"Control variables ➔ File ➔ Close"* with `g_bFileClose`
   - *"Control variables ➔ File ➔ Save"* with `g_bFileSave`
   - *"Control variables ➔ File ➔ New"* with `g_FileNew`

Declaring the control variables

```
VAR_GLOBAL
                g_sFileName: STRING := 'Readme.txt';
                g_bFileOpen : BOOL;
                g_bFileClose: BOOL;
                g_bFileSave: BOOL;
                g_FileNew: BOOL;
                g_usiErrorHandlingVarForErrorCode: USINT;
                g_bVarForContentChanged : BOOL;
                g_bVarForReadWriteMode: BOOL;
                END_VAR
```

Configuring control elements for file selection

1. Add an element *"Label"*.
2. Configure it in the property *"Texts ➔ Text"* with `File:`.
3. Add an element *"Rectangle"* next to it.
4. Configure its property *"Texts ➔ Text"* with `%s`.
5. Configure its property *"Texts ➔ Text variable"* with `g_sFileName`.
6. Configure the property *"Input configuration ➔ OnMouseclick"* with *"Write a variable"*.

   In the dialog *"Input Configuration"*, select *"Text input"* as the *"Input type"*.

   Activate the option *"Use text output variable"*.

   ⇨ The rectangle for the input of the file name is configured.
7. Add an element *"Button"*.
8. Configure its property *"Texts ➔ Text"* with `New`.
9. Configure the property *"Input configuration ➔ OnMouseclick"* with *"Toggle a variable"*.

   Assign `g_bFileNew` as a variable.

   ⇨ The button `New` is configured.
10. Add a further element *"Button"*.
11. Configure the property *"Texts ➔ Text"* with `Open`:
12. Configure the property *"Input configuration ➔ OnMouseclick"* with *"Toggle a variable"*.

    Assign `g_bFileOpen` as a variable.

    ⇨ The button `Open` is configured.
13. Add a further element *"Button"*.
14. Configure its property *"Texts ➔ Text"* with `Save`.
15. Configure the property *"Input configuration ➔ OnMouseclick"* with *"Toggle a variable"*.

    Assign `g_bFileSave` as a variable.

    ⇨ The button `Save` is configured.
16. Add a further element *"Button"*.
17. Configure its property *"Texts ➔ Text"* with `Close`.

18. Configure the property *“Input configuration ➔ OnMouseclick”* with *“Toggle a variable”*.

    Assign `g_bEditFile` as a variable.

    ⇨ The button `Close` is configured.

See also

- ↳ *Chapter 1.4.5.18.1.41 “Visualization Element 'Text Editor'” on page 1653*

1.4.5.13 Configuring a variable assignment with unit conversion

A variable that was assigned in a visualization can be linked with a unit conversion. This causes the variable value to be converted according to a predefined rule and the result is edited in the visualization.

You have already configured the conversion rules in the editor of an object of the type *“Unit Conversion”*.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Linking a variable with unit conversion

☑ Requirement: A project with a visualization is open. In addition, the application contains the object `UnitConversion` with the rule `convert_A`.

1. Select an element.

   ⇨ The view *“Properties”* opens.

2. When assigning a variable, link the variable `iVar_A` with a rule of the unit conversion:

   ```
   convert_A.convert(iVar_A)
   ```

3. Compile, load and start the application.

   ⇨ The application runs. The visualization opens. The unit conversion is applied.

1.4.5.14 Using recipes in visualization elements

You can manage and use the recipes created in CODESYS by means of a visualization. For this purpose, the input configuration of a visualization element provides the following commands:

- *“Read Recipe”*
- *“Write Recipe”*
- *“Load Recipe from File”*
- *“Save Recipe to File”*
- *“Create Recipe”*
- *“Delete Recipe”*

See also

- ↳ *Chapter 1.4.1.12.2 “Changing Values with Recipes” on page 417*
- ↳ *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Example: Loading recipes by means of visualization elements

Requirement: The *“Visualization”* object is added to the project.

1. Create a recipe according to the instructions in the section "Changing Values with Recipes - Creating Recipes".

   Assign the following names:

   - Recipe definition: "Recipes"
   - Recipes: "Recipe1" and "Recipe2"
   - Variables: `iValue1` and `iValue2`

   Type in different variable values of both recipes.
2. Open the *“Visualization”* object in the editor.
3. Drag a *“Button”* element to the visualization. Label it "Load Recipe 1". You can specify the text by double-clicking the element or in the *“Texts ➔ Text”* property.
4. Click the value field of the *“Input configuration”*: *“OnMouseDown”* property.

   ⇨ The *“Input Configuration”* dialog box opens.
5. Select *“Execute command”* in the left of the left side and click the button

   ⇨ The configuration of the *“Internal command”* opens on the right side of the dialog.
6. Select the *“Write Recipe”* command from the drop-down list.
7. Click the button.

   ⇨ The *“WriteRecipe”* command is added to the list.
8. Specify the first parameter as `Recipes` and the second parameter as `Recipe1`.
9. Click *“OK”* to close the dialog box.
10. Drag a second button to the visualization, name it "Load Recipe 2", and repeat steps 4 to 8. For step 7, specify `Recipe2` as the second parameter.
11. Load the program to the controller and start it Click the *“Load Recipe 1”* and *“Load Recipe 2”*, and monitor the variables `iValue1` and `iValue2`.

The other recipe commands are assigned to visualization elements as described in this example. Refer to the help page of the input configuration for a description of the internal commands.

1.4.5.15 Creating a structured user interface

You can reference visualizations that are available or exist in the project in another visualization and thus reuse them. You obtain a structured user interface that consists of several visualizations. In principle you have the following possibilities to reference visualizations.

On the one hand you can display visualizations within a main visualization and toggle between them. The element *“Frame”* or *“Tabs”* serves here as a window area element that defines the display area for the referenced visualizations.

On the other hand you can configure a user input for a visualization that causes another visualization to open as a dialog. The requirements for this is that it has the visualization type *“Dialog”*. A dialog is used to collect inputs from the user.

In addition, you can declare an interface for a visualization that is to be referenced in order to vary the display of the visualization at runtime. A visualization is thereby instanced with different data and executed.

See also

- *Chapter 1.4.5.18.1.6 “Visualization Element 'Frame'” on page 1432*
- *Chapter 1.4.5.18.1.10 “Visualization Element 'Tabs'” on page 1463*

1.4.5.15.1 Displaying Multiple Visualizations in One Visualization

You can reference other visualizations within a main visualization either in a *“Frame”* or a *“Tabs”* element, and then display them in the window pane of the element.

In the case of the *“Frame”* element, you can freely program which of the visualizations is displayed at which time. One option is to use the switch frame variable of the *“Frame”* element, which automatically triggers a switch according to its value. You can also program additional controls which, after user input, trigger input actions that result in switching a visualization.

NOTICE!

Visualizations can be nested at any depth by means of *“Frame”* elements. In order to use the *“Switch to any visualization”* frame selection type without any problems, a *“Frame”* must not contain more than 21 referenced visualizations.

For more information, see also the description for the *“Input configuration”* of an element: Action *“Switch frame visualization”*.

Moreover, you can use the *“Tabs”* to reference visualizations. It is easy and advantageous that the *“Tabs”* element provides preconfigured control of the visualization switch.

In CODESYS Forge, you will find the sample project "Visualization Switching". There you will see a visualization that displays other visualizations in a frame area one after another at runtime. The visualization switch is controlled either by the user, programmatically, or via the FrameManager.

See also

- *Sample project in CODESYS Forge*

Switching frame visualizations by means of a variable

In the main visualization, the *“Frame”* element displays one of the referenced frame visualizations at runtime. The user can select the *“Radio Buttons”* element which is displayed in the frame.

Connecting frame visualizations with a radio buttons element

1. Create a new standard project in CODESYS.
2. Select the application in the device tree and click *“Add Object ➔ Visualization”*.
3. In the *“Add Visualization”* dialog, specify the name `VisuMain` and click *“Add”* to close the dialog.
4. Select the application in the device tree and click *“Add Object ➔ Visualization”*.
5. In the *“Add Visualization”* dialog, specify the name `Visu1` and click *“Add”* to close the dialog.
6. Select the application in the device tree and click *“Add Object ➔ Visualization”*.
7. In the *“Add Visualization”* dialog, specify the name `Visu2` and click *“Add”* to close the dialog.
8. Select the application in the device tree and click *“Add Object ➔ Visualization”*.
9. In the *“Add Visualization”* dialog, specify the name `Visu3` and click *“Add”* to close the dialog.

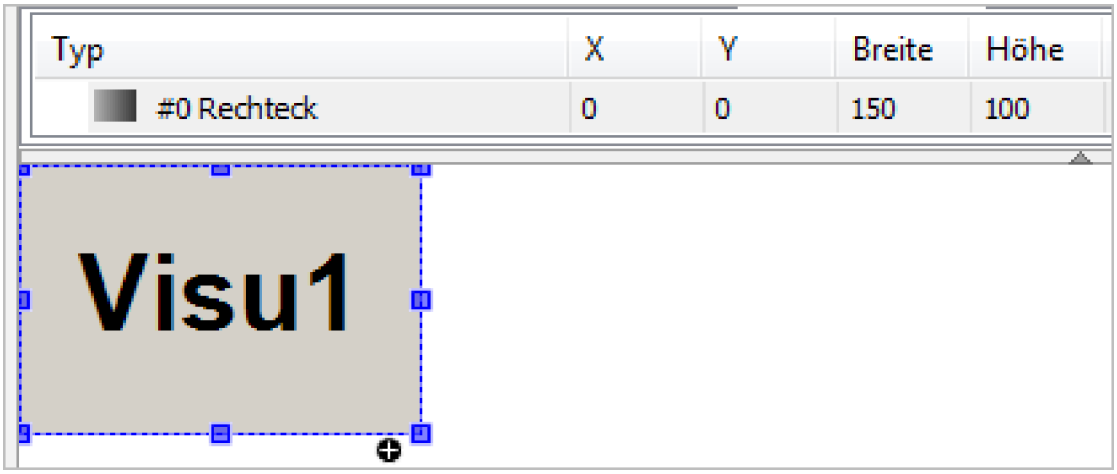
   ⇨ In addition to the main visualization, there are three more visualization objects.
10. Open the `Visu1` object.

11. In the *“Visualization Toolbox”*, in the *“Basic”* category, select and drag the *“Radio Buttons”* element to the visualization editor.

    ⇨ The *“Properties”* view of the element opens.

12. Configure the properties of the rectangle as follows:

    - Property *“Texts”*, *“Text”* = `Visu1`
    - Property *“Text properties”*, *“Font”* = *“Title”*
    - Property *“Colors”*, *“Normal state”*, *“Fill color”* = *“Light gray”*



13. Program the object `Visu2` accordingly.

    Properties of the rectangle:

    - Property *“Texts”*, *“Text”* = `Visu2`
    - Property *“Text properties”*, *“Font”* = *“Title”*
    - Property *“Colors”*, *“Normal state”*, *“Fill color”* = *“Gray”*

14. Program the object `Visu3` accordingly.

    Properties of the rectangle:

    - Property *“Texts”*, *“Text”* = `Visu3`
    - Property *“Text properties”*, *“Font”* = *“Title”*
    - Property *“Colors”*, *“Normal state”*, *“Fill color”* = *“Dark gray”*

15. Open the `VisuMain` object.

16. In the *“Visualization Toolbox”*, in the *“Basic”* category, select and drag the *“Frame”* element to the visualization editor.

    ⇨ The *“Frame Configuration”* dialog opens.

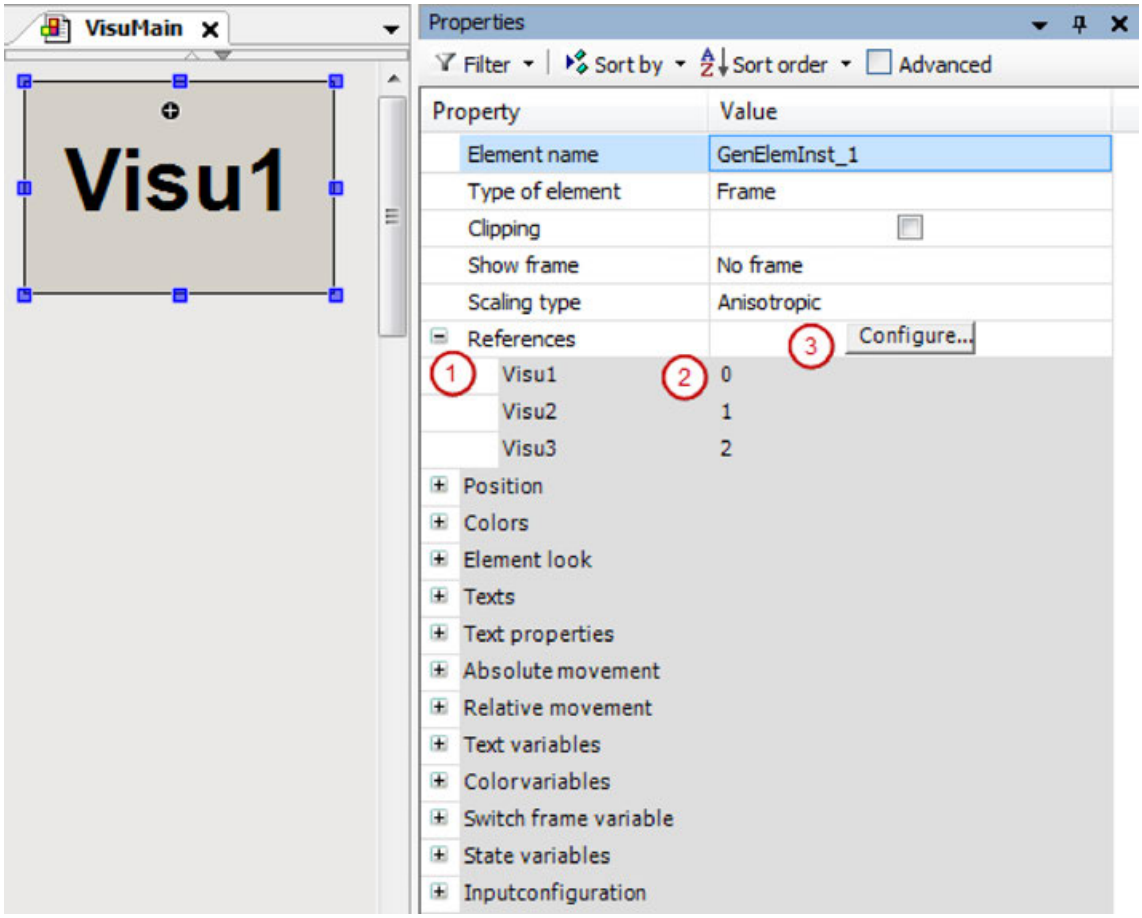
17. In the *“Available Visualizations”* window area, on the *“By Visualization Name”* tab, select the object `Visu1`. In *“Selected Visualizations”*, click *“Add”*.

18. Then select the object `Visu2` and click *“Add”* in *“Selected Visualizations”*.

19. Then select the object `Visu3` and click *“Add”* in *“Selected Visualizations”*.

20. Click *“OK”* to exit the dialog.

    ⇨ Now the *“Frame”* element references the three selected visualizations. The references (1) are listed in the *“References”* property in the element properties of the *“Frame”* element. In addition to the visualization name, the corresponding index value (2) is also displayed.



    Note: You can open the dialog when you click the *“Configure”* button in the value field of the *“References”* property. See (3). You can influence the index by means of the visualization order in the *“Selected Visualizations”* list.

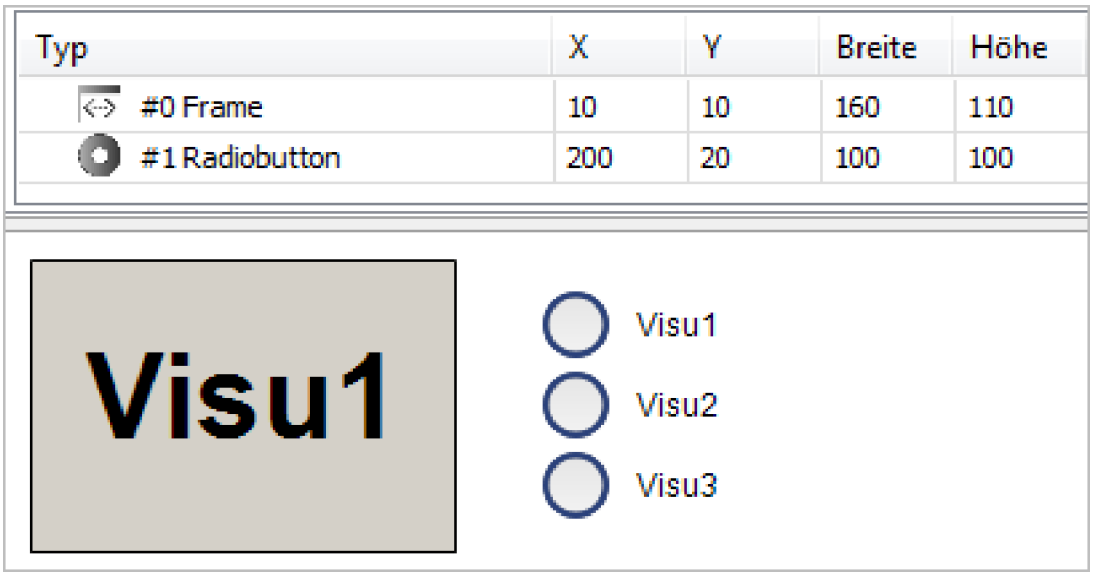
21. In the *“Visualization Toolbox”*, in the *“Common Controls”* category, drag the *“Radio Buttons”* element to the visualization editor.

    ⇨ The *“Properties”* view of the element opens.

22. In the *“Radio button settings”*, *“Radio button”*, click the *“Create new”* button.

    ⇨ This element has three switches to select from.

23. Configure the properties of the radio button as follows:
    - Property *“Radio button settings”*, *“Areas”*, *“[0]”*, *“Text”* = `Visu1`
    - Property *“Radio button settings”*, *“Areas”*, *“[1]”*, *“Text”* = `Visu2`
    - Property *“Radio button settings”*, *“Areas”*, *“[2]”*, *“Text”* = `Visu3`



24. In the `PLC_PRG` program, declare a local variable for the number of the visualization that is active.

    ⇨
    ```
    VAR
        iActiveVisu : INT; // Index of visu activated by the user
    END_VAR
    ```

25. Select the *“Radio Buttons”* element. In the value field of the *“Variable”* property, click ....

26. In the *“Input Assistant”* dialog, select the recently declared variable. Then exit the dialog.

    ⇨ Property of the *“Radio Buttons”* element:
    - Property *“Variable”* = `PLC_PRG.iActiveVisu`

27. Select the *“Frame”* element. Click in the value field of the *“Switch frame variable”*, *“Variable”* property. Specify the recently declared variable here as well.

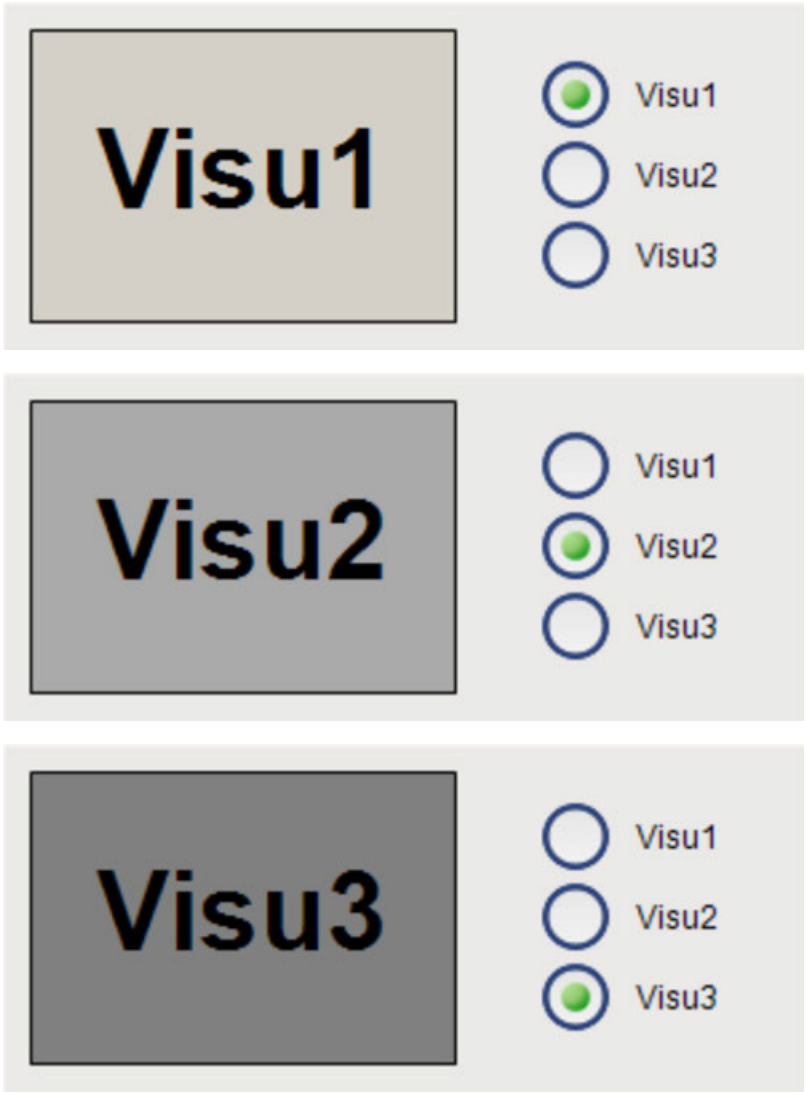
    ⇨ Property of the *“Frame”* element:
    - Property *“Switch frame variable ”*, *“Variable”* = `PLC_PRG.iActiveVisu`

    The control variable of the *“Radio Buttons”* element is also the switch frame variable of the *“Frame”* element. User input for the *“Radio Buttons”*element switches the frame visualization.

28. Click *“Build ➔ Generate Code”*.

29. Click *“Online ➔ Login”* and start the application.

    ⇨ The visualization starts. One of the referenced visualizations is running in the frame. When you click an unselected option of the *“Radio Buttons”* element, the visualization switches the contents in the frame to the desired visualization.



In the example, the switch frame variable is connected to an input variable. Instead, you can also set the switch frame variable programmatically in the IEC code.

Switching frame visualizations by means of a follow-up action

In the main visualization, the *“Frame”* element displays one of the frame visualizations at runtime. The user can use buttons to control the display in the frame. The user input triggers the *“Switch frame visualization”* input action.

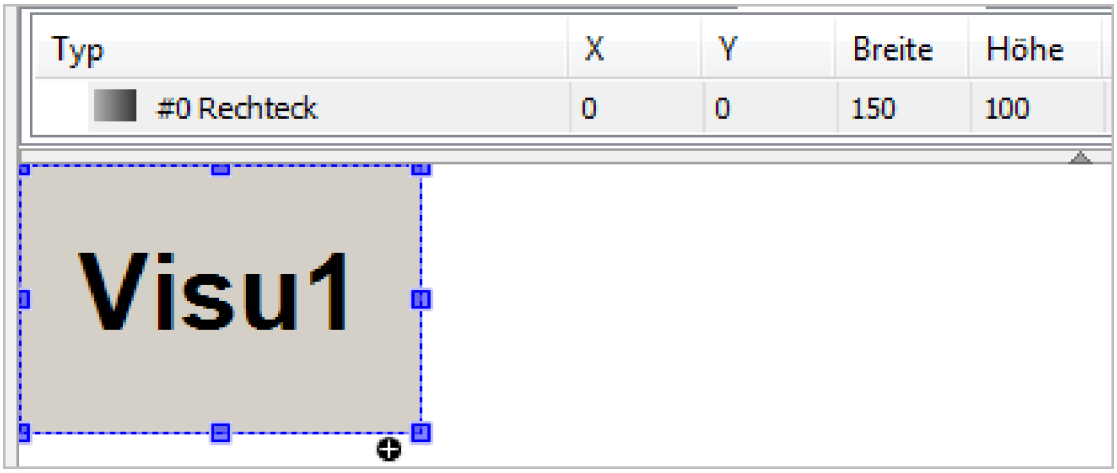
Programming a visualization

1. Create a new standard project in CODESYS.
2. Select the application in the device tree and click *“Add Object ➔ Visualization”*.
3. In the *“Add Visualization”* dialog, specify the name `VisuMain` and click *“Add”* to close the dialog.
4. Select the application in the device tree and click *“Add Object ➔ Visualization”*.
5. In the *“Add Visualization”* dialog, specify the name `Visu1` and click *“Add”* to close the dialog.
6. Select the application in the device tree and click *“Add Object ➔ Visualization”*.

7. In the *"Add Visualization"* dialog, specify the name `Visu2` and click *"Add"* to close the dialog.
8. Select the application in the device tree and click *"Add Object ➔ Visualization"*.
9. In the *"Add Visualization"* dialog, specify the name `Visu3` and click *"Add"* to close the dialog.

   ⇨ In addition to the main visualization, there are three more visualization objects.
10. Open the `Visu1` object.
11. In the *"Visualization Toolbox"*, in the *"Basic"* category, select and drag the *"Rectangle"* element to the visualization editor.

    ⇨ The *"Properties"* view of the element opens.
12. Configure the properties of the rectangle as follows:
    - Property *"Texts"*, *"Text"* = `Visu1`
    - Property *"Text properties"*, *"Font"* = *"Title"*
    - Property *"Colors"*, *"Normal state"*, *"Fill color"* = *"Light gray"*



13. Program the object `Visu2` accordingly.

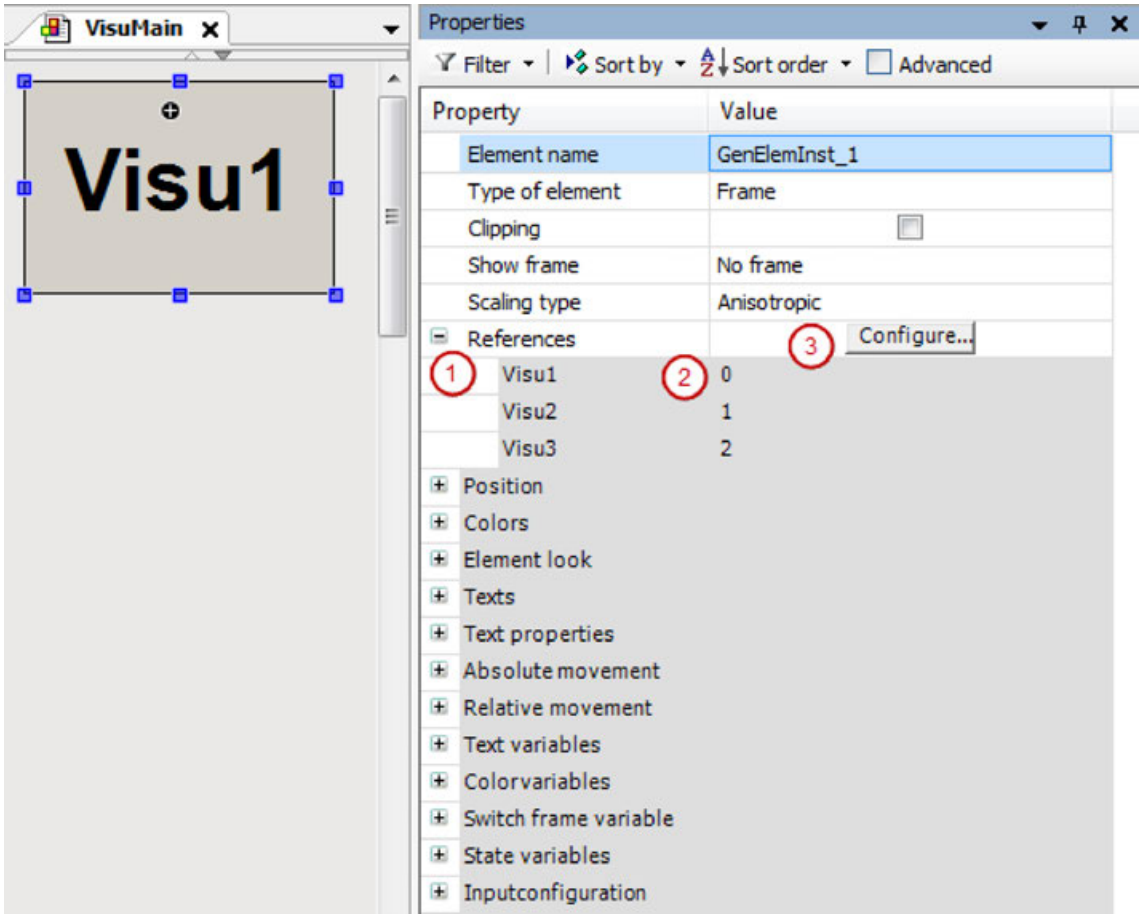
    ⇨ Properties of the rectangle:
    - Property *"Texts"*, *"Text"* = `Visu2`
    - Property *"Text properties"*, *"Font"* = *"Title"*
    - Property *"Colors"*, *"Normal state"*, *"Fill color"* = *"Gray"*
14. Program the object `Visu3` accordingly.

    ⇨ Properties of the rectangle:
    - Property *"Texts"*, *"Text"* = `Visu3`
    - Property *"Text properties"*, *"Font"* = *"Title"*
    - Property *"Colors"*, *"Normal state"*, *"Fill color"* = *"Dark gray"*
15. Open the `VisuMain` object.
16. In the *"Visualization Toolbox"*, in the *"Basic"* category, select and drag the *"Frame"* element to the visualization editor.

    ⇨ The *"Frame Configuration"* dialog opens.
17. In the *"Available Visualizations"* window area, on the *"By Visualization Name"* tab, select the object `Visu1`. In *"Selected Visualizations"*, click *"Add"*.
18. Then select the object `Visu2` and click *"Add"* in *"Selected Visualizations"*.
19. Then select the object `Visu3` and click *"Add"* in *"Selected Visualizations"*.

20. Click *“OK”* to exit the dialog.

    ⇨ Now the *“Frame”* element references the three selected visualizations. The references (1) are listed in the *“References”* property in the element properties of the *“Frame”* element. In addition to the visualization name, the corresponding index value (2) is also displayed.



    Note: You can open the dialog independently when you click the *“Configure”* button in the value field of the *“References”* property. See (3). You can influence the index by means of the visualization order in the *“Selected Visualizations”* list.

21. In the *“Visualization Toolbox”*, in the *“Common Controls”* category, drag the *“Button”* element to the visualization editor.

    ⇨ The element is selected and its properties are visible in the *“Properties”* view.

22. Configure the *“Texts”*, *“Text”* property with `Visu1`.

23. In the *“Input configuration”**“OnMouseDown”* property, click *“Configure”*.

    ⇨ The *“Input Configuration”* dialog opens.

24. Select the *“Switch frame visualization”* action and click .

    ⇨ The action is displayed in the window on the right.

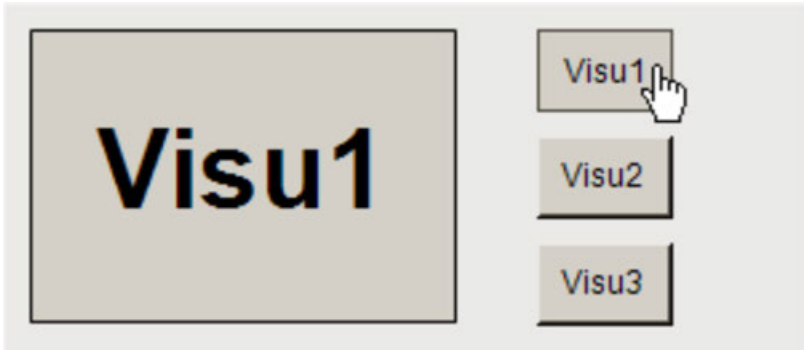
25. Configure the action:

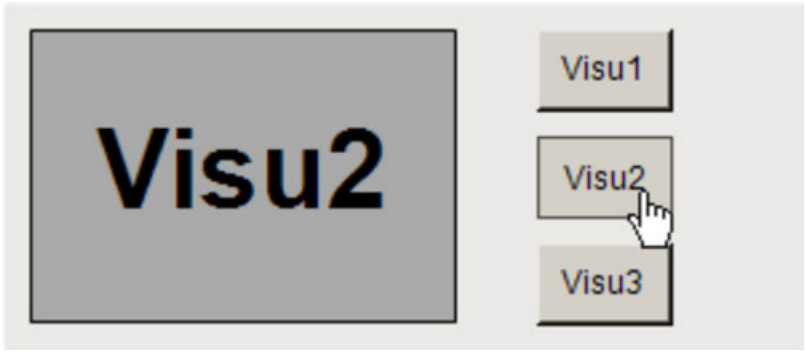
    - Select the *“Switch local visualization”* option.
    - Set the *“Visualization selection”* to `Visu1`.
    - Click *“OK”* to exit the dialog.

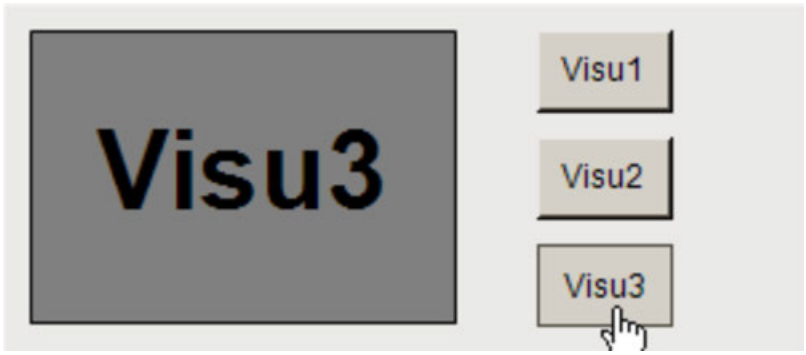
    ⇨ The follow-up action is configured in the *“Input configuration”* property.

    Property *“Input configuration”*, *“OnMouseDown”*, *“Switch frame visualization”* = `0`

26. Drag another *“Button”* element to the visualization editor. Configure the button accordingly.
    ⇨ Properties of the button:
    - Property *“Texts”*, *“Text”* = `Visu2`
    - Property *“Input configuration”*, *“OnMouseDown”*, *“Switch frame visualization”* = `1`
27. Drag another *“Button”* element to the visualization editor. Configure the button accordingly.
    ⇨ Properties of the button:
    - Property *“Texts”*, *“Text”* = `Visu3`
    - Property *“Input configuration”*, *“OnMouseDown”*, *“Switch frame visualization”* = `2`
28. Click *“Build ➔ Generate Code”*.
29. Click *“Online ➔ Login”* for the device and start the application.
    ⇨ The visualization starts. One of the referenced visualizations is running in the frame. When you click one of the buttons, the visualization switches the contents in the frame to the respective visualization.







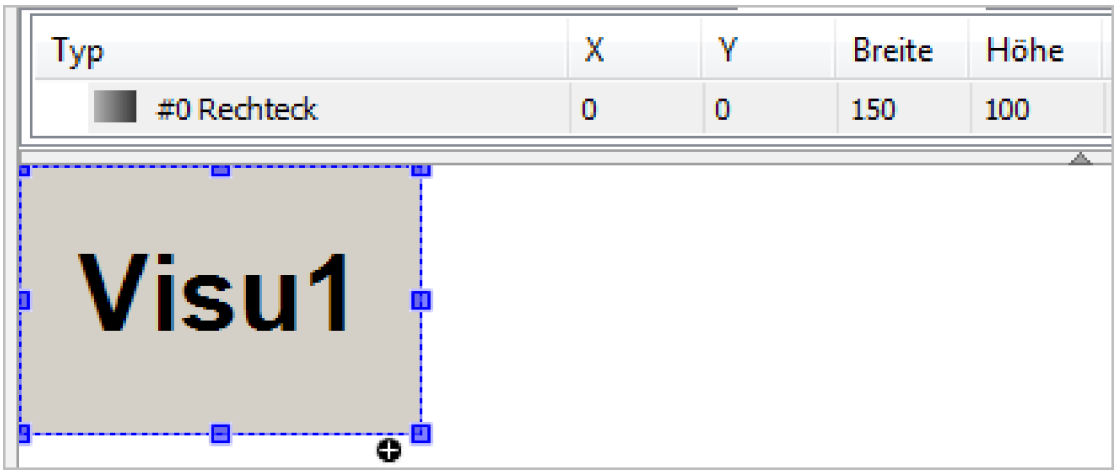
Displaying visualizations on a tabs element

For the *“Tabs”*, the navigation of the referenced visualizations is provided automatically. The first of the referenced visualizations is in the foreground, while the others are hidden behind it. The user can navigate between them by means of the tabs which are provided automatically.

Configuring a tabs element

1. Create a new standard project in CODESYS.
2. Select the application in the device tree and click *“Add Object ➔ Visualization”*.

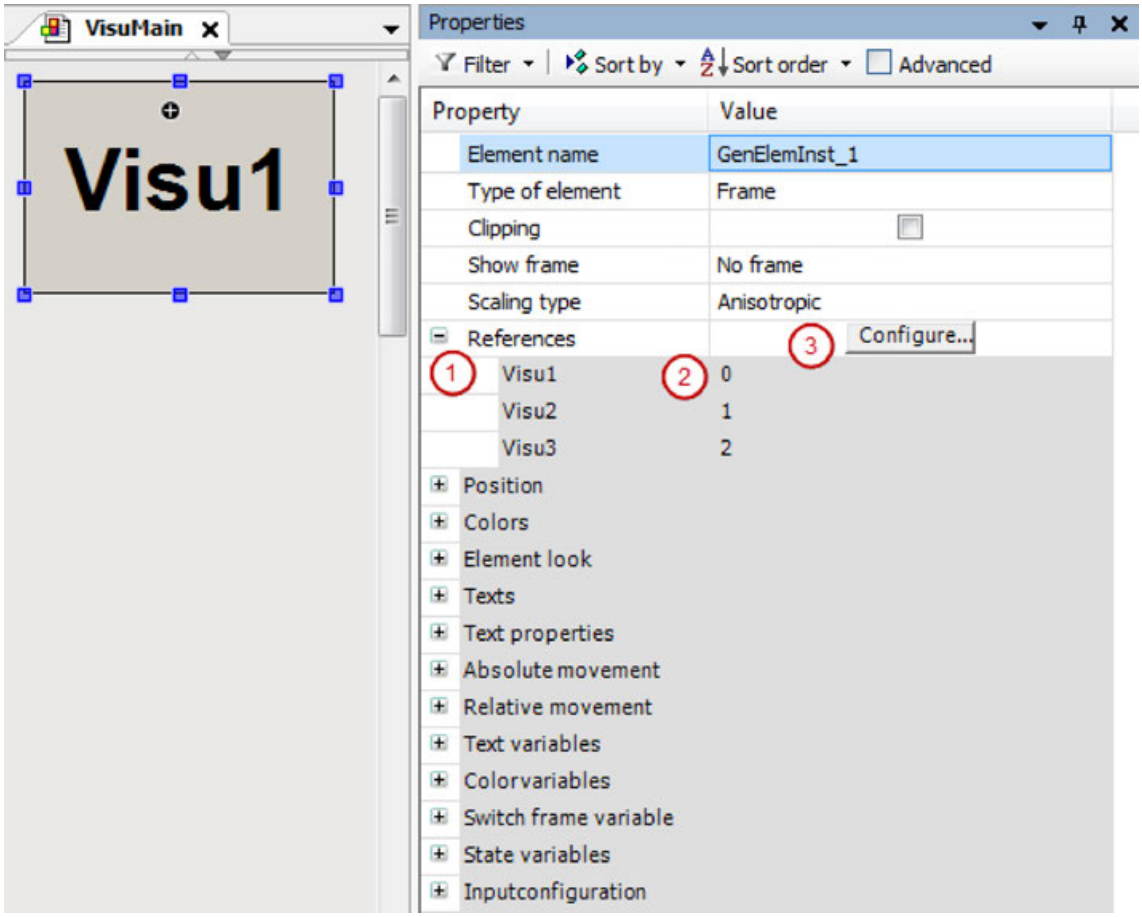
3. In the *"Add Visualization"* dialog, specify the name `VisuMain` and click *"Add"* to close the dialog.
4. Select the application in the device tree and click *"Add Object ➔ Visualization"*.
5. In the *"Add Visualization"* dialog, specify the name `Visu1` and click *"Add"* to close the dialog.
6. Select the application in the device tree and click *"Add Object ➔ Visualization"*.
7. In the *"Add Visualization"* dialog, specify the name `Visu2` and click *"Add"* to close the dialog.
8. Select the application in the device tree and click *"Add Object ➔ Visualization"*.
9. In the *"Add Visualization"* dialog, specify the name `Visu3` and click *"Add"* to close the dialog.
   ⇨ In addition to the main visualization, there are three more visualization objects.
10. Open the `Visu1` object.
11. Drag a *"Rectangle"* element to the visualization editor.
    ⇨ The *"Properties"* view of the element opens.
12. Configure the properties of the rectangle as follows:
    - Property *"Texts"*, *"Text"* = `Visu1`
    - Property *"Text properties"*, *"Font"* = *"Title"*
    - Property *"Colors"*, *"Normal state"*, *"Fill color"* = *"Light gray"*



13. Program the object `Visu2` accordingly.
    ⇨ Properties of the rectangle:
    - Property *"Texts"*, *"Text"* = `Visu2`
    - Property *"Text properties"*, *"Font"* = *"Title"*
    - Property *"Colors"*, *"Normal state"*, *"Fill color"* = *"Gray"*
14. Program the object `Visu3` accordingly.
    ⇨ Properties of the rectangle:
    - Property *"Texts"*, *"Text"* = `Visu3`
    - Property *"Text properties"*, *"Font"* = *"Title"*
    - Property *"Colors"*, *"Normal state"*, *"Fill color"* = *"Dark gray"*
15. Open the `VisuMain` object.
16. In the *"Visualization Toolbox"*, in the *"Basic"* category, select and drag the *"Frame"* element to the visualization editor.
    ⇨ The *"Frame Configuration"* dialog opens.

17. In the *“Available Visualizations”* window area, on the *“By Visualization Name”* tab, select the object `Visu1`. In *“Selected Visualizations”*, click *“Add”*.
18. Then select the object `Visu2` and click *“Add”* in *“Selected Visualizations”*.
19. Then select the object `Visu3` and click *“Add”* in *“Selected Visualizations”*.
20. Click *“OK”* to exit the dialog.

    ⇨ Now the *“Tabs”* element references the three selected visualizations. The references (1) are listed in the *“References”* property in the element properties of the *“Frame”* element. In addition to the visualization name, the corresponding index value (2) is also displayed.

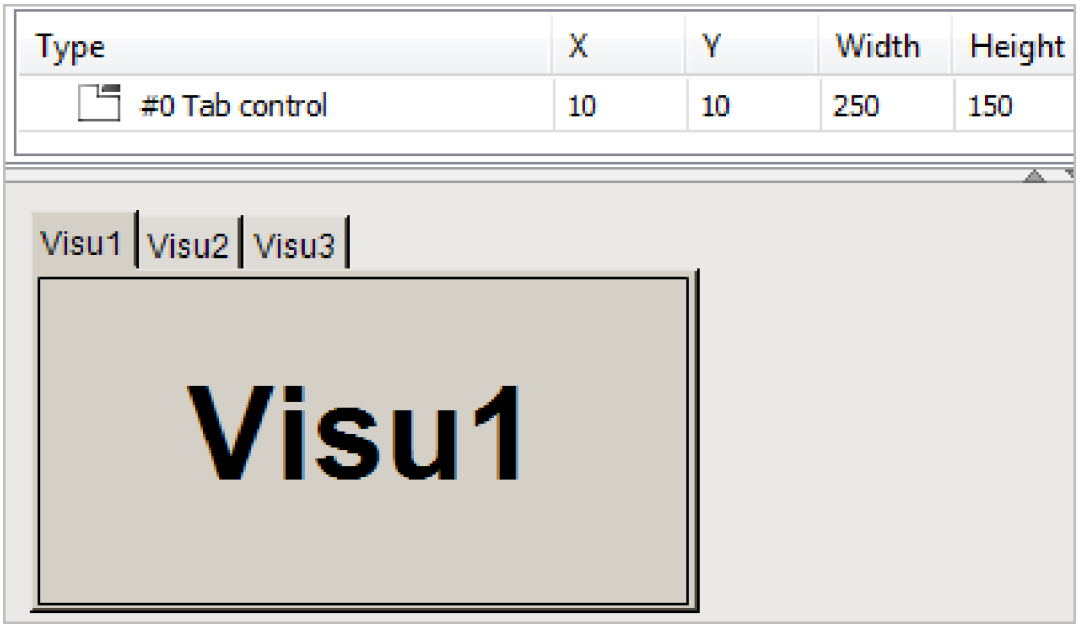


Note: You can open the dialog *“Frame Configuration”* dialog independently when you click the *“Configure”* button in the value field of the *“References”* property. See (3). You can influence the index by means of the visualization order in the *“Selected Visualizations”* list.

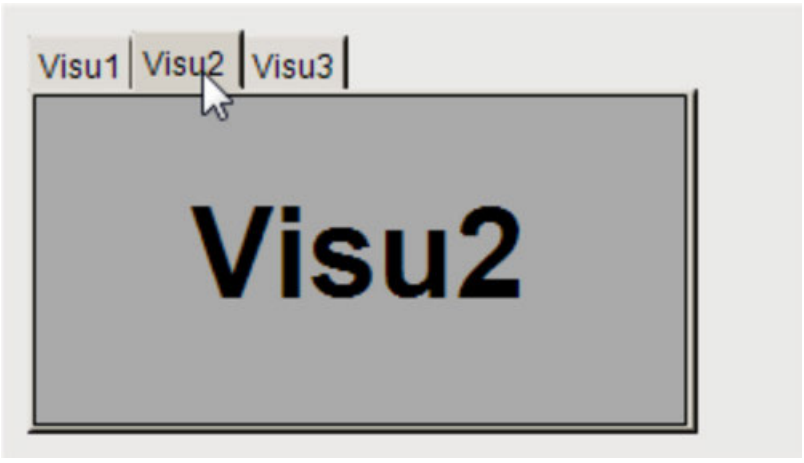
21. In the *“Visualization Toolbox”*, in the *“Common Controls”* category, drag the *“Tabs”* element to the visualization editor.

    ⇨ The *“Properties”* view of the element opens.

22. Configure the properties of the tab as follows:
    - Property *“Tab width”*: `40`
    - Property *“References”*, `Visu1`, *“Header”* = `Visu1`
    - Property *“References”*, `Visu2`, *“Header”* = `Visu2`
    - Property *“References”*, `Visu3`, *“Header”* = `Visu3`



23. Click *“Build ➔ Generate Code”*.
24. Click *“Online ➔ Login”* for the device and start the application.
    ⇨ The visualization starts. One of the referenced visualizations is running in the *“Tabs”* element. Click the tab to switch to the respective visualization.



See also

- ↳ *“Dialog 'Frame Configuration'” on page 1727*
- ↳ *Chapter 1.4.5.18.1.6 “Visualization Element 'Frame'” on page 1432*
- ↳ *Chapter 1.4.5.18.1.10 “Visualization Element 'Tabs'” on page 1463*

1.4.5.15.2 Calling a Visualization with an Interface

You can declare an interface for parameters for a visualization that is to be referenced. The actual parameters are passed to the interface (similar as in the case of a function block) when the visualization is called at runtime.

First of all, declare the interface variables in the visualization interface editor. Then configure the parameters that are transferred to the interface by assigning a data-type-compliant application variable to each interface variable. The assignment is configured in the *"References"* property in the case of a *"Frame"* or a *"Tabs"*.

Depending on the display variant, the parameter transfer of local variables (with the `VAR` scope) is limited. If you execute the visualization as an integrated visualization, you can only transfer local variables having a basic data type as parameters. If the visualization is called as CODESYS TargetVisu or CODESYS WebVisu, then you can also transfer parameters with a user-defined data type.

User-controlled update of the transfer parameters

If you have configured visualization references and then save a change to the variable declaration for one of these visualizations in an interface editor, then the *"Updating the Frame Parameters"* dialog appears automatically. The dialog prompts you to edit the references. A list of all the visualizations affected is displayed there, so that the parameter transfers can be reassigned at the changed interface.

When the dialog is closed, the changes are accepted and the elements affected are displayed in the *"References"* property.

Calling visualization with interface (`VAR_IN_OUT`)

☑ Requirement: The project contains a visualization and a main visualization. The main visualization contains an element that the visualization references.

1. Open the visualization.
2. Click *"Visualization ➔ Interface Editor"*.
3. Declare a variable in the interface editor.
   ⇨ The visualization has an interface and the *"Updating the Frame Parameters"* dialog appears.
4. Assign a type-compliant transfer parameter to the interface variables in all calls by entering an application variable in *"Value"*. Close the dialog.
   ⇨ A transfer parameter is assigned at the points where the visualization is to be referenced. These now appear in the main visualization in the *"References"* property.

Example

The `visPie` visualization contains an animated, colored pie. The `visMain` main visualization calls the `visPie` visualization multiple times in a *"Tabs"* control. Color information, angle information, and label are transferred via the `pieToDisplay` interface variable. The pies vary at runtime.

Visualization `visPie`:

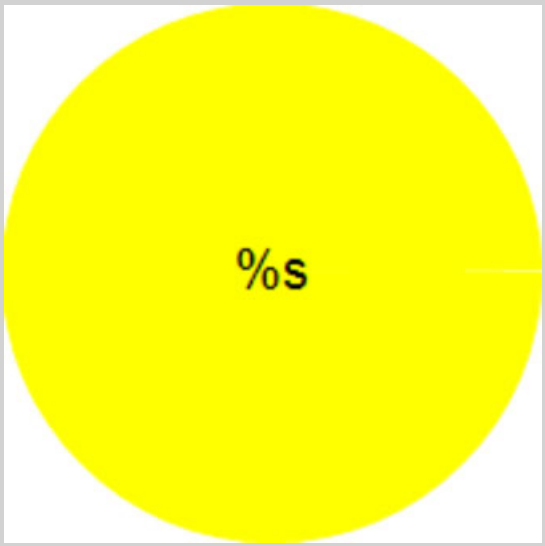


Table 263: Properties of the "Pie" element:

"Variable for begin"	`pieToDisplay.iStart`
"Variable for end"	`pieToDisplay.iEnd`
"Texts ➔ Text"	%s
"Text variables ➔ Text variable"	`pieToDisplay.sLabel`
"Color variable ➔ Normal state"	`pieToDisplay.dwColor`

Interface of the visualization `visPie`:

```
VAR_IN_OUT
    pieToDisplay : DATAPIE;
END_VAR
```

Main visualization `visMain`:

Table 264: Properties of the "Tabs" element:

"References"	
"visPie"	
"Heading"	`A`
`pieToDisplay`	`PLC_PRG.pieA`
"visPie"	
"Heading"	`B`
`pieToDisplay`	`PLC_PRG.pieB`
"visPie"	
"Heading"	`C`
`pieToDisplay`	`PLC_PRG.pieC`

DATAPIE (STRUCT)

```
TYPE DATAPIE : // Parameter type used in visPie
STRUCT
    dwColor : DWORD; // Color data
    iStart : INT; // Angle data
    iEnd : INT;
    sLabel : STRING;
END_STRUCT
END_TYPE
```

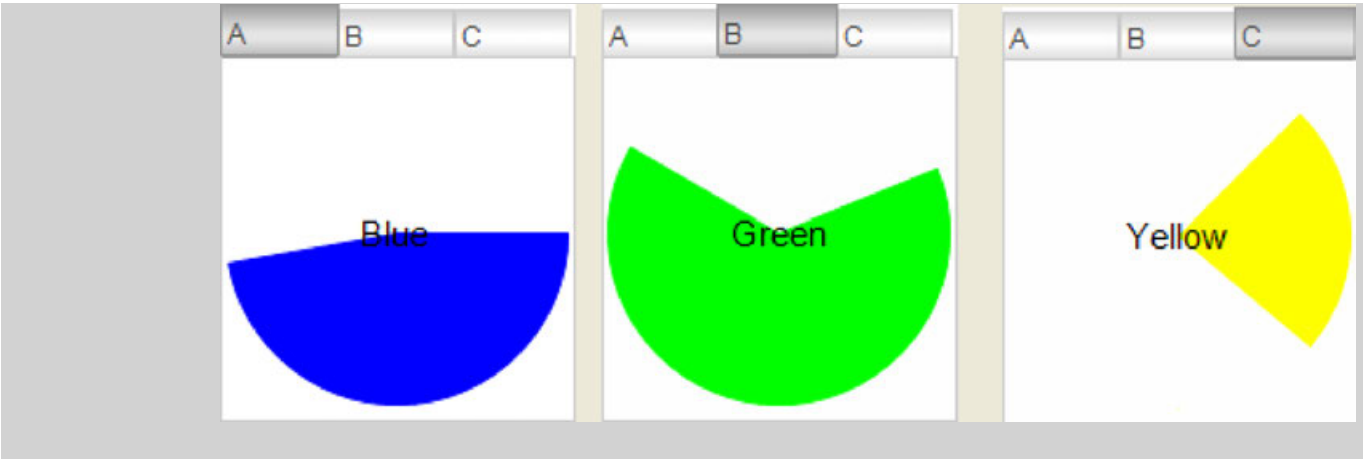
GVL

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
    c_dwBLUE : DWORD := 16#FF0000FF; // Highly opaque
    c_dwGREEN : DWORD := 16#FF00FF00; // Highly opaque
    c_dwYELLOW : DWORD := 16#FFFFFF00; // Highly opaque
    c_dwGREY : DWORD :=16#88888888; // Semitransparent
    c_dwBLACK : DWORD := 16#88000000; // Semitransparent
    c_dwRED: DWORD := 16#FFFF0000;  // Highly opaque
END_VAR
```

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    iInit: BOOL := TRUE;

    pieA : DATAPIE; // Used as argument when visPie is called
    pieB : DATAPIE;
    pieC : DATAPIE;

    iDegree : INT; // Variable center angle for the pie element
used for animation
END_VAR

IF iInit = TRUE THEN
    pieA.dwColor := GVL.c_dwBLUE;
    pieA.iStart := 0;
    pieA.sLabel := 'Blue';

    pieB.dwColor := GVL.c_dwGREEN;
    pieB.iStart := 22;
    pieB.sLabel := 'Green';

    pieC.dwColor := GVL.c_dwYELLOW;
    pieC.iStart := 45;
    pieC.sLabel := 'Yellow';

    iInit := FALSE;
END_IF

iDegree := (iDegree + 1) MOD 360;

pieA.iEnd := iDegree;
pieB.iEnd := iDegree;
pieC.iEnd := iDegree;
```

Main visualization `visMain` at runtime:



Printing the instance name of a transfer parameter

In order to obtain and output the instance name of a transfer parameter, you can implement an interface variable (data type `STRING`) with the pragma `{attribute 'parameterstringof'}` in the `VAR_INPUT` scope.

☑ The project contains a visualization and a main visualization. The main visualization contains elements that the visualization references.

1. Open the visualization.
2. Click *"Visualization ➔ Interface Editor"*.
3. Declare an interface variable (`VAR_IN_OUT`).

   ⇨ `pieToDisplay : DATAPIE;`
4. In the interface editor, declare a variable (`VAR_INPUT`) with attribute `{attribute 'parameterstringof'}`.

   ⇨ `{attribute 'parameterstringof' := 'pieToDisplay'}`

   `sNameToDisplay : STRING;`
5. Save the changes.

   ⇨ The *"Updating the Frame Parameters"* dialog does not open.
6. Insert a *"Text Field"* element.
7. In the *"Texts"*, *"Text"* property, assign an output text to the text field.

   ⇨ `Visualization of %s`
8. In the *"Text variables""Text variable"* property, assign the interface variable to the text field.

   ⇨ `sNameToDisplay`

   `visPie` has a heading.

Example

The `visPie` visualization consists of one pie until now. The `visMain` main visualization calls `visPie` in a *"Tabs"* control three times with different transfer parameters.

The `visPie` is extended with a text field that outputs the name of the parameters actually passed to the visualization. For this, the interface of `visPie` is extended with a string variable that contains the instance name of the specified transfer parameter. At runtime, each pie is overwritten.

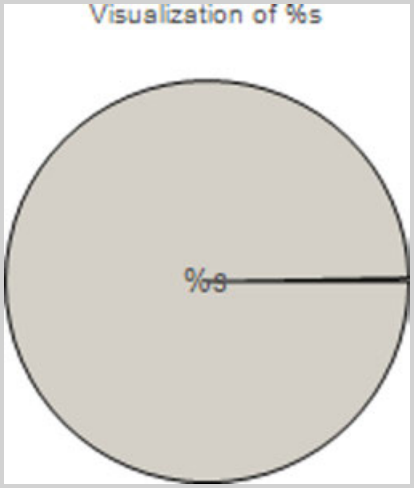


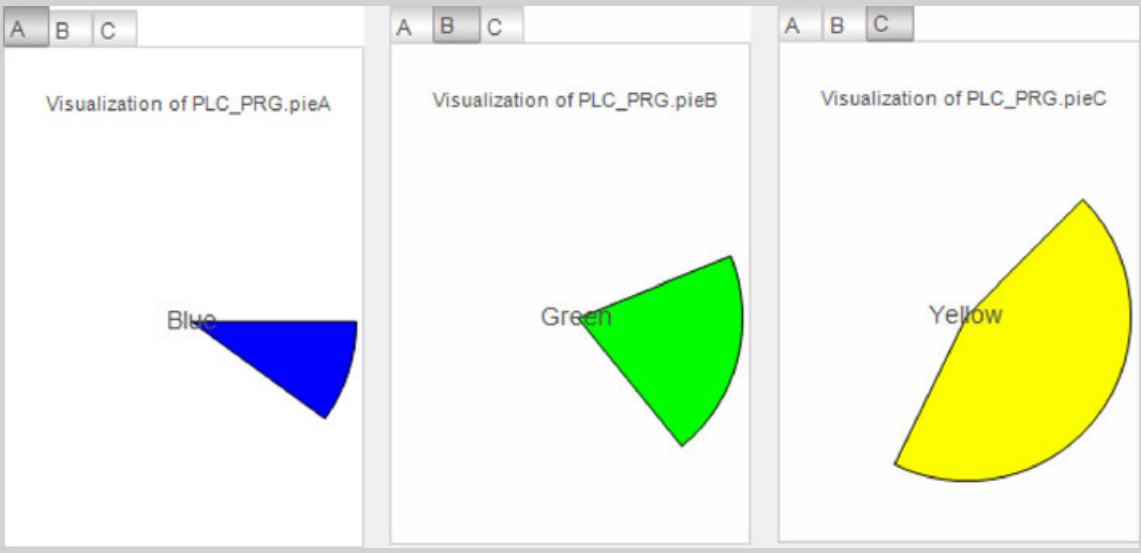
Table 265: Properties of the "Text field" element:

"Texts", *"Text"*	`Visualization of %s`
"Text variables", *"Text variable"*	`sNameToDisplay`

Interface of the 'visPie' visualization:

```
VAR_INPUT
    {attribute 'parameterstringof' := 'pieToDisplay'}
    sNameToDisplay : STRING;
END_VAR
VAR_IN_OUT
    pieToDisplay : DATAPIE;
END_VAR
```

Main visualization `visMain` at runtime:



See also

- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*
- *Chapter 1.4.5.18.1.6 “Visualization Element 'Frame'” on page 1432*
- *Chapter 1.4.5.18.1.10 “Visualization Element 'Tabs'” on page 1463*
- *Chapter 1.4.5.19.3.3 “Dialog 'Update Frame Parameters'” on page 1746*

1.4.5.15.3 Calling a dialog in a visualization

You can configure a user input for a visualization that causes a referenced visualization to open as a dialog. For example, a user clicks on a button, whereupon a dialog opens requesting the input of values. A dialog is used to collect user inputs and, if it is modal, it can lead to inputs outside the dialog being blocked.

Only visualizations with the visualization type *“Dialog”* can be opened as dialog. The visualization type is configured in the dialog *“Properties”* of a visualization object.

Basic procedure:

☑ Requirement: The project contains a main visualization and a dialog.

1. Configure a user input for the main visualization with the action *“OpenDialog”* for the dialog.

   ⇨ The opening of the dialog is configured.

2. Configure a user input for an element of the dialog with the action *“CloseDialog”*.

   Hint: in the case of non-modal dialogs you can also configure the user input for closing outside the dialog.

   ⇨ The closing of the dialog is configured.

You can also use dialogs from the library instead of self-made dialogs. For example, if the library `VisuDialogs` is integrated in the project, you can use the dialogs `VisuDialogs.Login` or `VisuDialogs.FileOpenSave` contained in it.

See also

- *Chapter 1.4.5.4 “Configuring user inputs” on page 1267*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*
- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Configuring a visualization object as a dialog

1. Select the object in the view *“Devices”*, open the context menu and select the command *“Properties”*.
2. Select the tab *“Visualization”*.
3. Activate the option *“Dialog”* and close the dialog with *“OK”*.

   ⇨ The visualization has the visualization type *“Dialog”* and can be called as such.

Configuring a dialog call

When calling a dialog, a user normally clicks on a button, whereupon a dialog opens requesting an input.

In the following example, a dialog representing a calendar enables a date to be entered.

☑ Requirement: The project contains the visualizations `visMain` and `dlgCalender`.

1. Set the visualization type of `dlgCalender` to Dialog.
2. Drag a rectangle into the visualization `visMain`.
3. Configure the property *"Texts ➔ Text"* with the text `Due Date: %t[yyyy-MM-dd]`.

   Configure the property *"Text variables ➔ Text variable"* with `PLC_PRG.dateDue`.
4. Drag a button into the visualization.
5. Configure the property *"Texts ➔ Text"* with the text `Open dialog`.

   Configure the property *"Input configuration ➔ OnMouseClick"* for the action *"Open Dialog"* with `dlgCalender`.

   ⇨ The user input for the opening of the dialog is configured.
6. Double-click on the dialog `dlgCalender`.
7. Drag the element *"Date picker"* into the visualization editor.
8. Configure the property *"Texts ➔ Text"* with `Due Date: %t[yyyy-MM-dd]`.

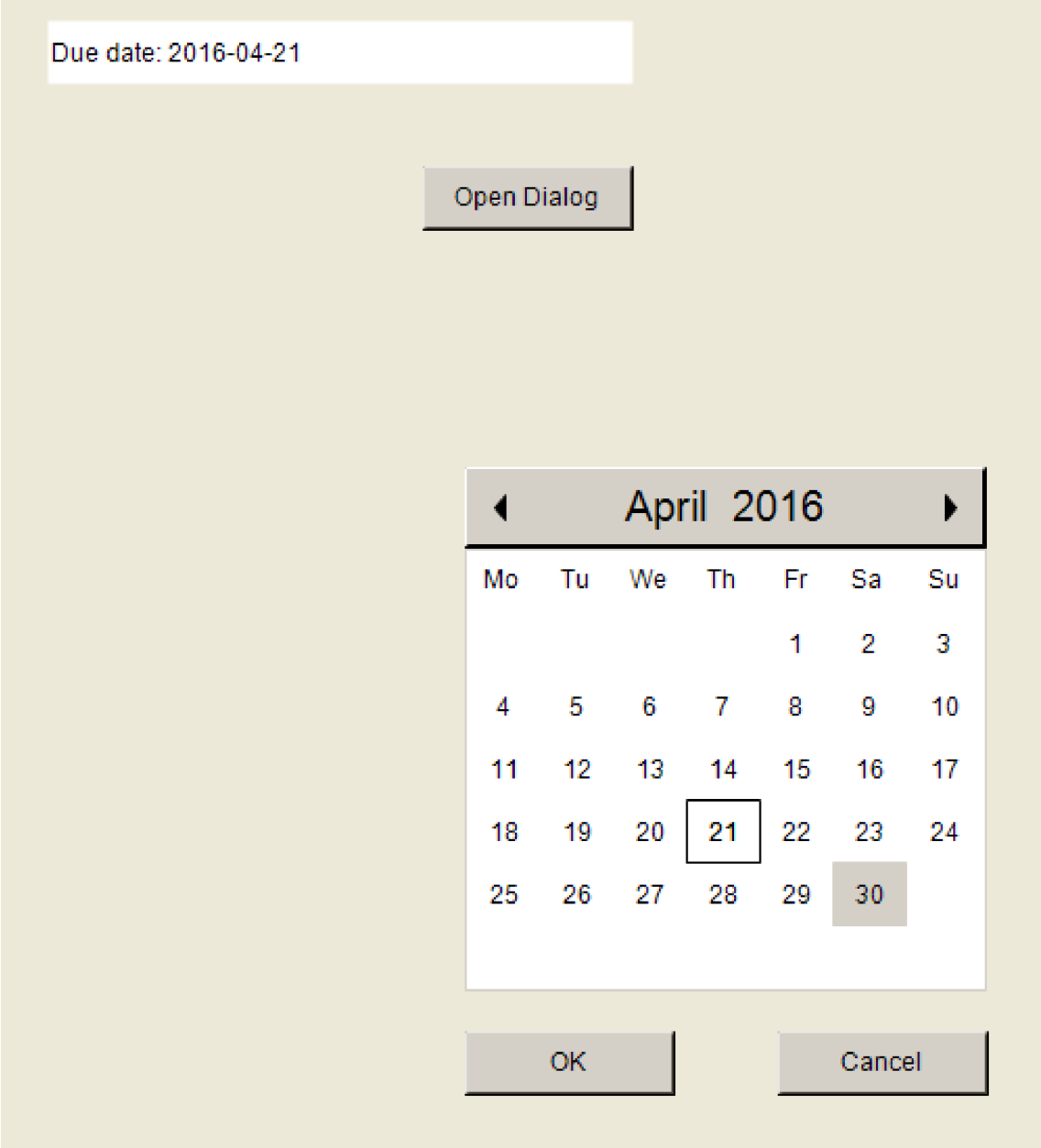
   Configure the property *"Variable"* with `PLC_PRG.dateCalender`.

   ⇨ The element is configured.
9. Drag a button into the visualization editor.
10. Configure the property *"Texts ➔ Text"* with `OK`:
11. Configure the property *"Input configuration ➔ OnMouseClick"* for the action *"Close Dialog "* with `dlgCalender, Result: OK`.
12. Configure a further property *"Input configuration ➔ OnMouseClick"* for the action *"Execute ST-Code"* with `PLC_PRG.dateDue := PLC_PRG.dateCalendar;`.

    ⇨ The user input for the closing of the dialog is configured.
13. Drag a further button into the visualization editor.
14. Configure the property *"Texts ➔ Text"* with `Cancel`:
15. Configure the property *"Input configuration ➔ OnMouseClick"* for the action *"Close Dialog "* with `dlgCalender, Result: Cancel`.

    ⇨ The user input for the cancellation of the dialog is configured.

17. Compile, load and start the application.



Variable declaration:

```
PROGRAM PLC_PRG
                VAR
                dateDue : DATE := DATE#2000-01-01;
                dateCalendar : DATE;
                END_VAR
```

Opening a dialog globally

Normally a dialog appears only on the display variant on which the user has executed the triggering event.

However, you can configure the opening of the dialog in such a way that the dialog appears simultaneously on all active display variants configured under the visualization manager. This way, for example, an input request can appear simultaneously on all display variants although a user only entered something on the CODESYS TargetVisu.

If a user closes the dialog on a CODESYS TargetVisu display variant, it will be closed on all display variants.

You can open and close a global dialog with the functions `OpenDialog3` and `CloseDialog2` from the library `VisuElems`.

Implementing an application access to a dialog

In the application code you can implement the access to a dialog that is managed in the dialog manager. The dialog manager automatically instances and manages all visualizations of the type *"Dialog"*. The access takes place via the internal visualization manager.

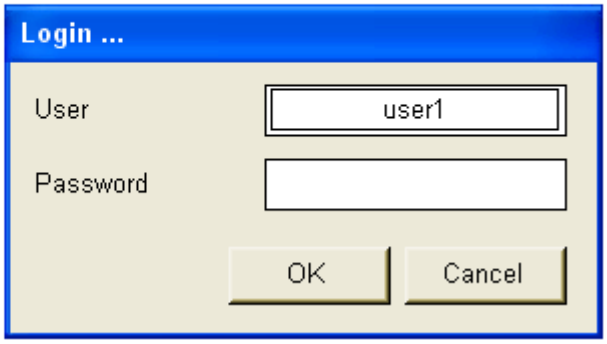
First of all, implement the access to the dialog manager by calling the `GetDialogManager()` method of the internal visualization manager. You can then use the methods of the dialog manager to program the program sequence of a dialog.

In the following example a button is configured so that it opens the preconfigured dialog `Login` when clicked on. The user can enter a name and a password in the dialog. The dialog `Login` is contained in the library `VisuDialogs`. You can also call a self-made dialog in the same way.

Implementing an application access to the dialog `Login` from the library `VisuDialogs`:

☑ Requirement: The library `VisuDialogs` is integrated in the project.

1. Insert a new visualization `visMain` under the application.
   ⇨ The visualization editor opens.
2. Drag a button into the visualization editor.
3. Enter in its property *"Text"* `Login`.
   ⇨ The button is labelled.
4. Click on *"Configure "*in the property *"Input configuration ➔ OnMouseDown"*.
5. Select the input action *"Execute ST-Code"* and click on ▫.
6. Enter the following function call in the ST editor: `OpenLoginDialog(pClientData);`
   ⇨ The main visualization contains a button. If a user clicks on the button, the dialog `Login` opens and the function `OpenLoginDialog()` is called.
7. Click on *"Configure "*in the property *"Input configuration ➔ OnDialogClosed"*.
8. Select the input action *"Execute ST-Code"* and click on ▫.
9. Enter the following function call in the ST editor:
   `OnLoginDialogClosed(pClientData);`
   ⇨ If a user closes the dialog, the function `OnLoginDialogClosed()` is called.



Implementation of the function OpenLogin-Dialog():

```
FUNCTION OpenLoginDialog : BOOL
                VAR_INPUT
                pClientData : POINTER TO VisuStructClientData;
                END_VAR

                VAR
                dialogMan : IDialogManager;
```

```
                loginDialog : IVisualisationDialog;
                pLoginInfo : POINTER TO Login_VISU_STRUCT; //
Login_VISU_STRUCT contains the parameters defined in the interface of
visualization "Login".
                result : Visu_DialogResult;
                stTitle : STRING := 'Login ...';
                stPasswordLabelText: STRING;
                stUserLabelText: STRING;
                stUsername: STRING;
                END_VAR

                dialogMan := g_VisuManager.GetDialogManager(); // The
DialogManager is provided via the implicitly available VisuManager
                IF dialogMan <> 0 AND pClientData <> 0 THEN
                loginDialog :=
dialogMan.GetDialog('VisuDialogs.Login'); // Dialog to be opened is
specified
                IF loginDialog <> 0 THEN
                pLoginInfo :=
dialogMan.GetClientInterface(loginDialog, pClientData);
                IF pLoginInfo <> 0 THEN      // In the following the
parameters of the login dialog in the Login_VISU_STRUCT will be read
                pLoginInfo^.stTitle := stTitle;
                pLoginInfo^.stPasswordLabelTxt := stPasswordLabelText;
                pLoginInfo^.stUserLabelTxt := stUserLabelText;
                dialogMan.OpenDialog(loginDialog, pClientData, TRUE,
0);
                END_IF
                END_IF
                END_IF
```

`OnLoginDialogClosed()` defines the reaction to the closing of a dialog.

Implementation of the function `OnLoginDialogClosed()`:

```
 FUNCTION OnLoginDialogClosed : BOOL
                VAR_INPUT
                pClientData : POINTER TO VisuStructClientData;
                END_VAR

                VAR
                dialogMan : IDialogManager;
                loginDialog : IVisualisationDialog;
                pLoginInfo : POINTER TO Login_VISU_STRUCT;
                result : Visu_DialogResult;
                stPassword: STRING;
                stUsername: STRING;
                END_VAR

                dialogMan := g_VisuManager.GetDialogManager(); // The
DialogManager is provided via the implicitly available VisuManager
                IF dialogMan <> 0 AND pVisuClient <> 0 THEN
                loginDialog :=
dialogMan.GetDialog('VisuDialogs.Login'); // Gets the login dialog
                IF loginDialog <> 0 THEN
                result := loginDialog.GetResult(); // Gets the result
(OK, Cancel) of the dialog
                IF result = Visu_DialogResult.OK THEN
                loginDialog.SetResult(Visu_DialogResult.None); //
Reset to default (none)
                pLoginInfo :=
dialogMan.GetClientInterface(loginDialog, pVisuClient); // Structure
Login_VISU_STRUCT gets read;
                // In the following the structure parameters can be
set
                IF pLoginInfo <> 0 THEN
                stPassword :=
```

```
pLoginInfo^.stPasswordpLoginInfo^.stPassword := ''; // Reset the
passwword
                stUsername := pLoginInfo^.stUsername;
                END_IF
                ELSIF result = Visu_DialogResult.Cancel THEN
                loginDialog.SetResult(Visu_DialogResult.None); //
React on 'Cancel'
                ELSE
                // nothing to do
                END_IF
                END_IF
                END_IF
```

See also

- *Chapter 1.4.5.18.3 “Methods of the Dialog Manager” on page 1714*

1.4.5.15.4 Calling a Dialog with an Interface

You can define an interface for a visualization that is called as a dialog.

Create a visualization for this with visualization type *“Dialog”* and declare an interface for the dialog. The reference the visualization in a primary visualization by means of a user input and transfer the parameters to the interface.

If you call the visualization as an integrated visualization, then the parameter that are transferred must be variables of a basic data type. If the visualization is called as CODESYS TargetVisu or CODESYS WebVisu, then the parameters can have user-defined data types as well.

See also

- *“Scopes” on page 1719*
- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Main procedure

1. Set the visualization types of the visualization to dialog.
2. Declare variables in the interface editor of the dialog.
   ⇨ The dialog has an interface. You can transfer parameters when calling the dialog.
3. Configure the elements of the dialog and use the interface variables.
4. Select an element in another visualization (usually the main visualization) for configuring how the dialog opens.
5. Click *“Configure”* in the property *“Input configuration ➔ OnMouseDown”*.
   ⇨ The *“Input Configuration”* dialog box opens.
6. Select *“Open dialog”* in the list of selected input actions.
7. Select one from the *“Dialog”* drop-down list.
   ⇨ If the selected dialog has an interface, then the interface variables are listed below.
8. Assign a transfer parameter to the interface variables in the *“Value”* column.
9. Select the result for which the parameters were updated in the list *“Update”* *“and”* *“Parameter in case of results”*.
10. Activate the option *“Open dialog modal”*. Click *“OK”* to close the dialog box.
    ⇨ The dialog opening is configured.

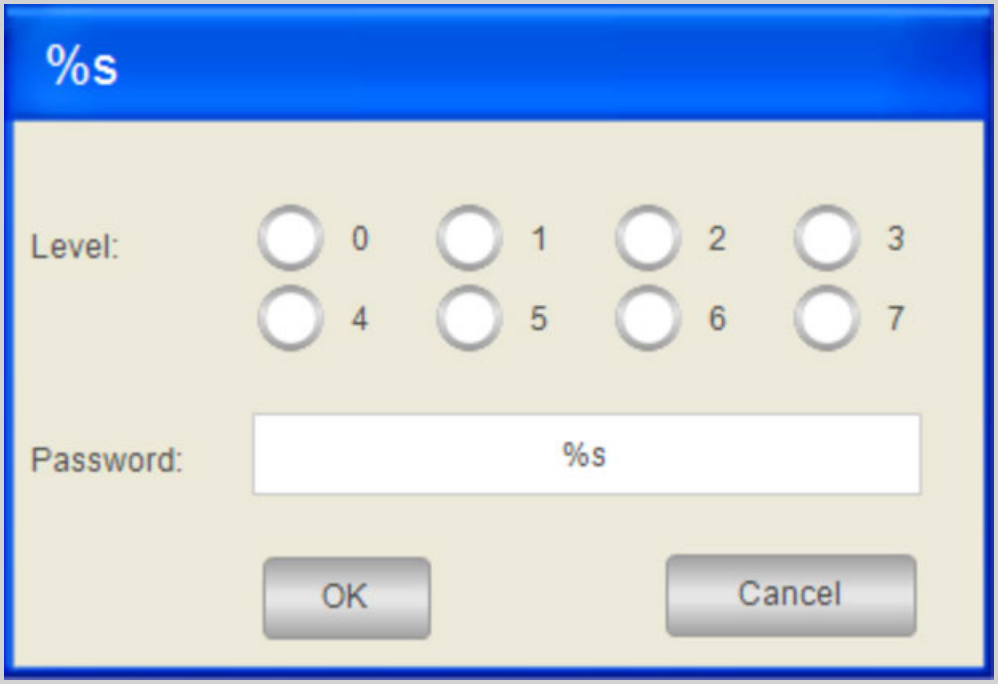
Executing a dialog several times at the same time requires multiple instances of the dialog. These must have already been downloaded to the visualization device when downloading the application. For this purpose, set the number of instances to download in the visualization manager (“Visualizations” tab).

See also

- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*
- *Chapter 1.4.5.15.4 “Calling a Dialog with an Interface” on page 1343*

Example the following application calls the *“Change User Level”* dialog and prompts the user to select a level and specify a password. If the password agrees, then the *“OK”* button is enabled. Then the user can close the dialog. The input of the level is also applied.

Dialog `visChangeUserLevel`:



Declaration of the interface of dialog `visChangeUserLevel`:

```
VAR_INPUT
    sTitle: STRING; // titel of the dialog box
    sItfLevel0: STRING; //password level 0
    sItfLevel1: STRING; //password level 1
    sItfLevel2: STRING; //password level 2
    sItfLevel3: STRING; //password level 3
    sItfLevel4: STRING; //password level 4
    sItfLevel5: STRING; //password level 5
    sItfLevel6: STRING; //password level 6
    sItfLevel7: STRING; //password level 7
END_VAR
VAR_IN_OUT
    iItfLevel: INT; // user input: level
    sItfPwd: STRING; //user input: password
END_VAR
```

Table 266: Element list of the `visChangeUserLevel` dialog box:

Type	Name	Element properties	Description
`#0 Image`	`Background`	*"Static ID"*: `VisuDialogs.ImagePoolDialogs.Login`	The property assigns the image of a blank dialog with a gray background and a blank blue caption bar to the element. The image is included in the *"VisuDialogs"* library.
`#1 Box`	`Title`	*"Texts ➔ Text"*: `%s`	Output with placeholder for text variable
		"Text variables ➔ Text variable": `sItfTitle`	Assignment of interface variable `sItfTitle` for which a parameter is transferred at call time.
`#2 Radio Buttons`	`Input level`	*"Variable"*: `iItfLevel`	Assignment of interface variable `iItfLevel` for which a parameter is transferred at call time. Includes the user input at runtime.
		"Number of columns": `4`	

Type	Name	Element properties	Description
		"Radio button order": *"Left to right"*	Display
		"Radio button settings ➔ Radio button ➔ Areas": [0] bis [7]*""* *"[<n>] ➔ Text"*: <n>	Label of eight radio buttons with numbers from 0 to 7
#3 Text Field	Input password	*"Texts ➔ Text"*: `%s`	Output with placeholder for text variable
		"Text variables ➔ Text variable": `sItfPwd`	Assignment of interface variable `sItfPwd` for which a parameter is transferred at call time. Includes the user input at runtime.
		"Input configuration ➔ OnMouseDown ➔ Write variable": `Variable:,InputType:Edit, Use text output variable : TRUE`	In the *"Input configuration"* dialog, *"Text input"* is selected for the *"Input type"* drop-down list and the option *"Use text output variable"* is activated.
#4 Text Field	Label for level	*"Texts ➔ Text"*: `Level:`	Label
#5 Text Field	Label for password	*"Texts ➔ Text"*: `Password`	Label
#6 Button	OK	*"Texts ➔ Text"*: `OK`	Label
		"Colors ➔ Color": `Element base color` *"Colors ➔ Alarm color"*: `Alarm fill color`	Configuration of the display in state-dependent colors. You can switch between colors.
		"Color variables ➔ Toggle color": `sItfPwd <> MUX(iItfLevel, sItfLevel0, sItfLevel1, sItfLevel2, sItfLevel3, sItfLevel4, sItfLevel5, sItfLevel6, sItfLevel7);`	If the password and the user input do not agree, then the expression is `TRUE`. Then the button is displayed in the alarm color.
		"State variables ➔ Deactivate inputs": `sItfPwd <> MUX(iItfLevel, sItfLevel0, sItfLevel1, sItfLevel2, sItfLevel3, sItfLevel4, sItfLevel5, sItfLevel6, sItfLevel7);`	If the password and the user input do not agree, then the expression is `TRUE`. The button is deactivated. If the password agrees, then the button is enabled.
		"Input configuration ➔ OnMouseDown ➔ Close dialog": `Close Dialog: visChangeUserLevel, Result : OK`	If a user clicks the *"OK"* button, then the `visChangeUserLevel` dialog is closed and the parameters are updated.
#7 Button	Cancel	*"Texts ➔ Text"*: `Cancel`	Label

Type	Name	Element properties	Description
		"Colors ➔ Color": `Element base color`	Display
		"Input configuration ➔ OnMouseDown ➔ Close dialog": `Close Dialog: visChangeUserLevel, Result : Cancel`	If a user clicks the *"Cancel"* button, then the `visChangeUserLevel` dialog is closed.

Main visualization `visMain`:

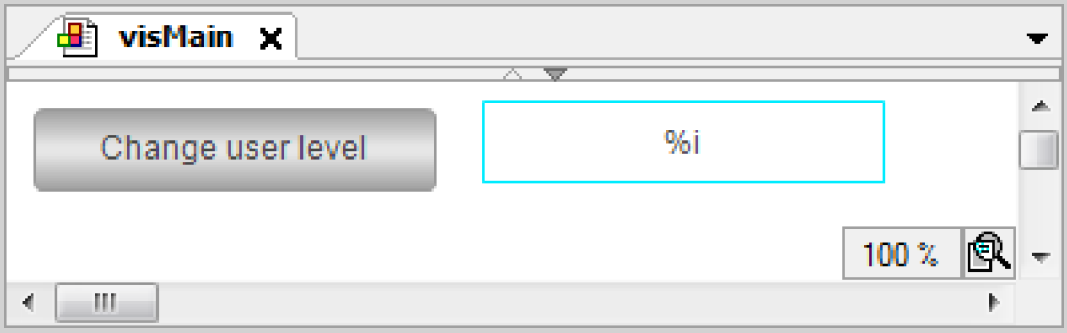


Table 267: Element list of the `visMain` visualization:

Type	Name	Element properties	Description
`#5 Text Field`	`Button for change user level`	*"Texts ➔ Text"*: `%s`	Output with placeholder
		"Text variables ➔ Text variable": `PLC_PRG.iLevel`	Assignment of the `PLC_PRG.iLevel` variables to the placeholder. Includes the level number.
`#6 Button`	`Title`	*"Texts ➔ Text"*: `Change user level`	
		"Input configuration ➔ OnMouseDown ➔ Open dialog": `Open Dialog: visChangeUserLevel`	If a user clicks the `Change user level` button, then the `visChangeUserLevel` dialog opens with the parameter list stored here. Tip: Click *"Configure"* to view the stored configuration in the *"Input Configuration"* dialog (input action *"Open dialog"*).

Table 268: Configuration of the call of dialog `visChangeUserLevel`:

Parameter	Type	Value	Description
The parameter list is stored in the *"Input Configuration"* dialog (input action *"Open dialog"*).			
`sItfTitle`	`STRING`	`'ChangeUse user level'`	Transfer of a string for the title.
`sItfLevel0`	`STRING`	`'pwd0'`	Transfer of a string as password for Level0.
`sItfLevel1`	`STRING`	`'pwd1'`	Transfer of a string as password for Level1.
`sItfLevel2`	`STRING`	`'pwd2'`	Transfer of a string as password for Level2.

Parameter	Type	Value	Description
sItfLevel3	STRING	'pwd3'	Transfer of a string as password for Level3.
sItfLevel4	STRING	'pwd4'	Transfer of a string as password for Level4.
sItfLevel5	STRING	'pwd5'	Transfer of a string as password for Level5.
sItfLevel6	STRING	'pwd6'	Transfer of a string as password for Level6.
sItfLevel7	STRING	'pwd7'	Transfer of a string as password for Level7.
iItfLevel	INT	PLC_PRG.iLevel	Transfer of a variable for the level specified by the user.
sItfPwd	STRING	PLC_PRG.sPwd	Transfer of a variable for the password specified by the user.

Table 269: List “Update” and “Parameter in case of result”

“Value”	**Description**
“OK”	activated

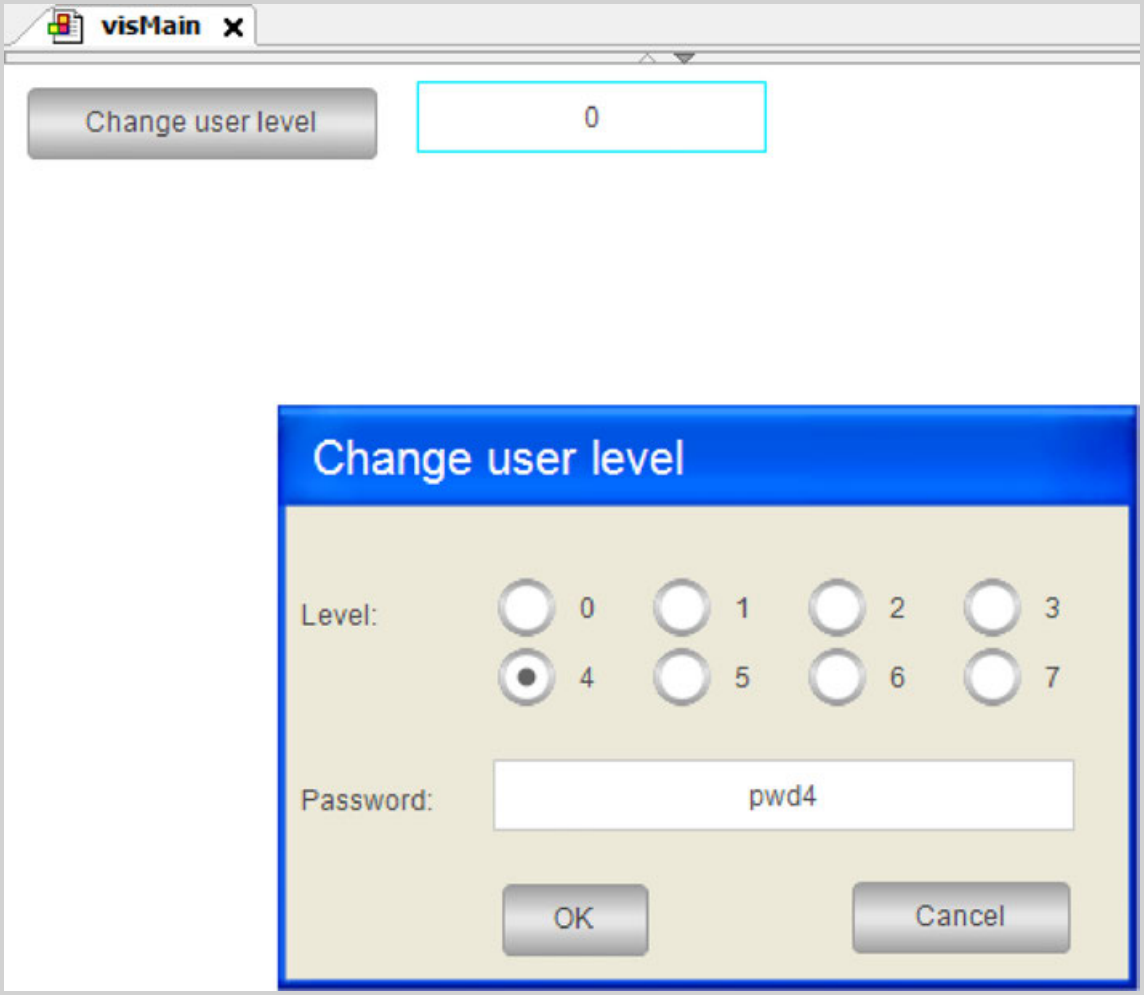
“Open in dialog mode”	activated Input outside of the dialog is not possible.

Application code `PLC_PRG`:

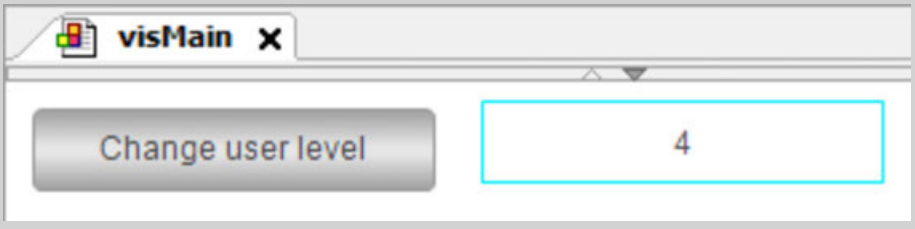
```
PROGRAM PLC_PRG
VAR
    iLevel: INT;
    sPwd : STRING;
END_VAR
```

Visualization at runtime

After clicking the button, the dialog opens and permits input. If the specified text agrees with the stored text, then *"OK"* is enabled:



After clicking *"OK"*, the selection is applied.



The example shows the procedure for multiple return values. However, the password can be returned more easily with a local variable in the dialog.

Accessing parameters programmatically

The variables declared in the interface of a visualization are available automatically as structure variables. They are identified by `<Name of visualization>_VISU_STRUCT`. Therefore, you can access the interface variables of visualizations that appear as a dialog. Normally you use the structure in the application code of a function that is called by a user input.

Passing pointers as parameters

To pass a complex data structure, you can flag an interface variable of type `VAR_IN_OUT` with the pragma attribute `VAR_IN_OUT_AS_POINTER` and pass a pointer or reference to it as a parameter.

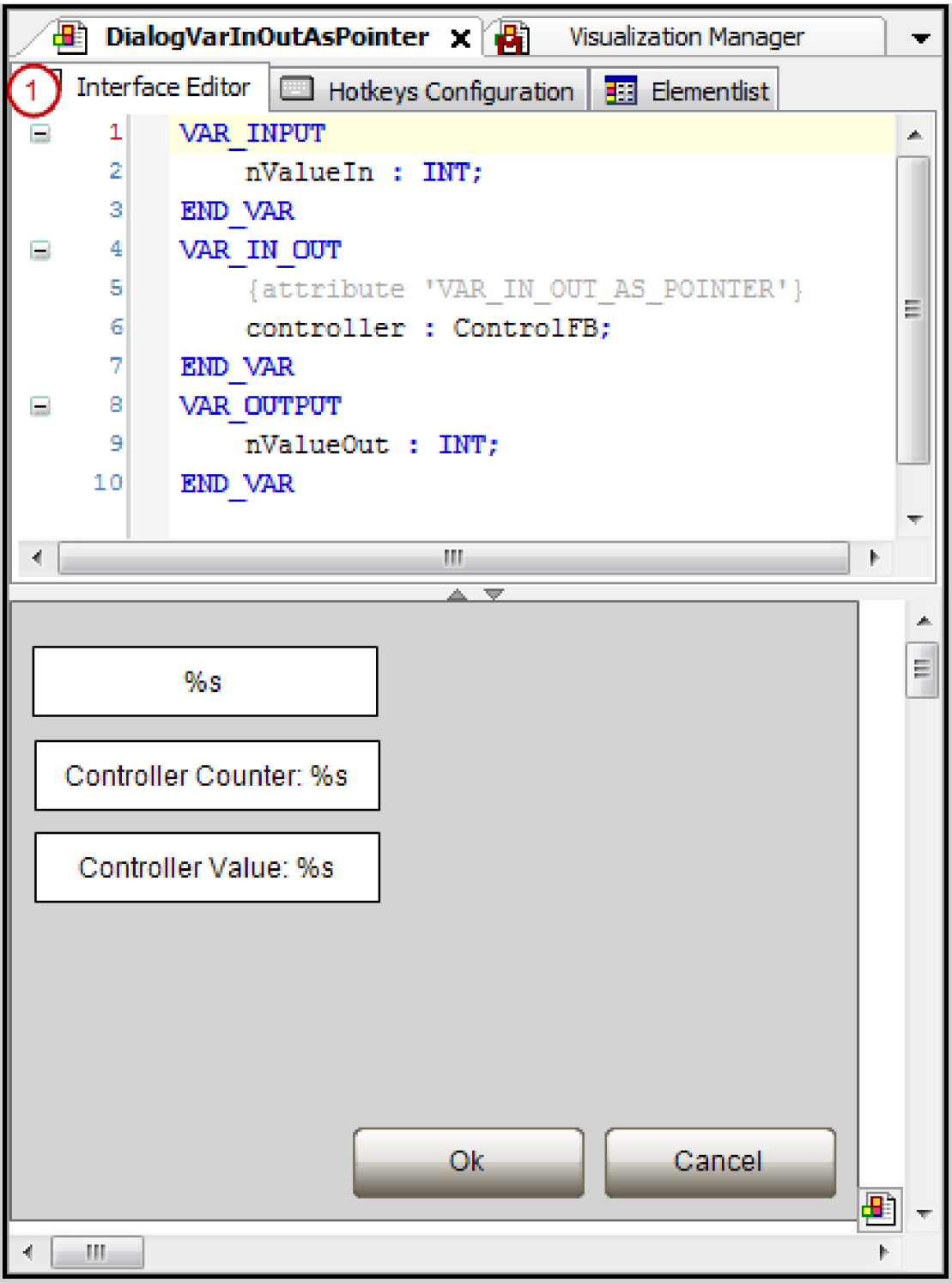
Procedure for using references

1. Declare the user data object (`DUT`).
2. In the interface editor of a dialog, declare an interface variable (`VAR_IN_OUT`) as a reference to the data object by assigning the attribute `'VAR_IN_OUT_AS_POINTER'` to the variable.
3. Program the user interface: use the dialog in a visualization or assign the dialog in the input configuration of a visualization element. Then access to the referenced data is possible.

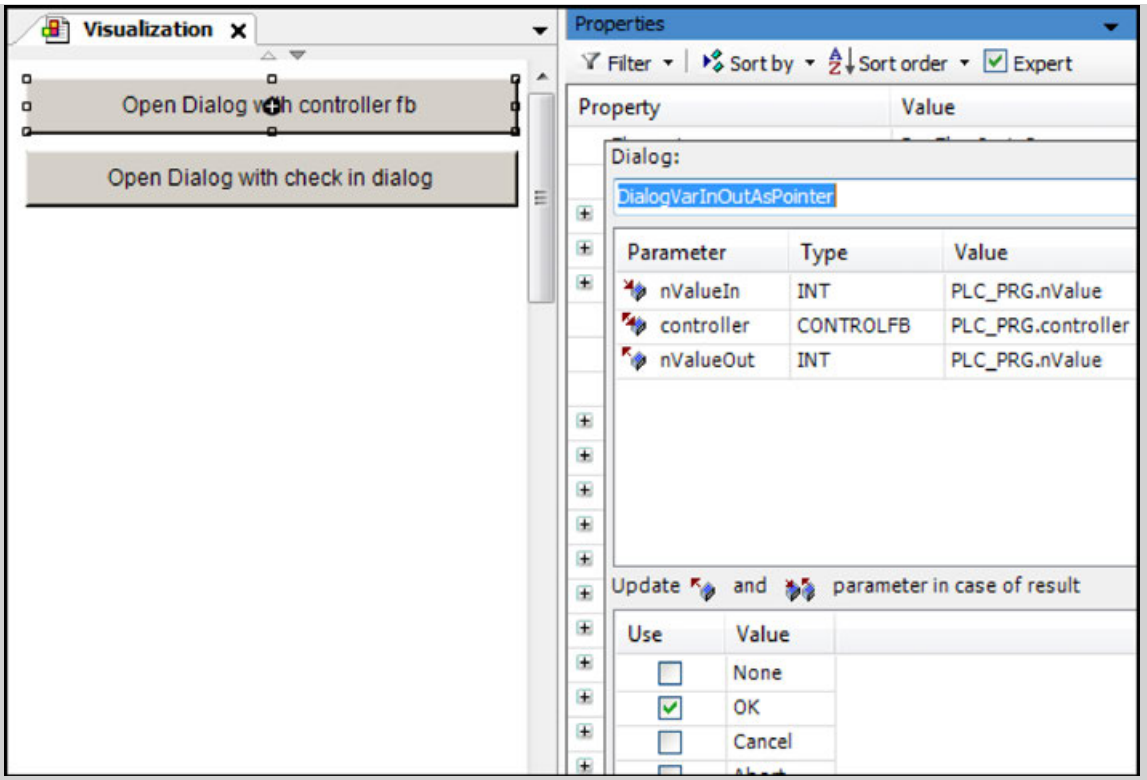
Example: Using an interface with the pragma 'VAR_IN_OUT_AS_POINTER'

```
FUNCTION_BLOCK ControlFB
VAR
    bOk : BOOL := TRUE;
    nCounter : INT;
    nValue : INT;
END_VAR
nCounter := nCounter + 1;
```

Declaration of an interface variable with `VAR_IN_OUT_AS_POINTER`



User interface: dialog opens:



See also

- *Chapter 1.4.5.18.4 “Attribute 'VAR_IN_OUT_AS_POINTER'” on page 1716*

1.4.5.16 Configuring and executing display variants

You can select from different variants for displaying your visualization created in CODESYS. An advantage is that you can have not only one, but multiple display variants running at the same time. During this time, the contents of the visualization are the same for all variants. This also applies to the integrated visualization: when the visualization editor is open in CODESYS, the visualization is also displayed there with the same active contents.

The following object types are available:

- *“TargetVisu”*
  The display as a variant of CODESYS TargetVisu is possible one time. You can insert an object of this type below the Visualization Manager.
- *“WebVisu”*
  The display as a variant of CODESYS WebVisu is possible any number of times. You can insert any number of objects.
- *“Remote TargetVisu”*
  The display as a variant of **[ERROR: Missing definition for variable "tvVisuDeviceRemoteTarget"!]** is possible any number of times. You can insert any number of objects.

When you insert a variant below the Visualization Manager, the task configuration is extended by the visualization task `VISU_TASK` (the flow unit of the visualizations). The task is automatically deleted when no more objects exist below the Visualization Manager or the objects below are excluded from compiling. You can set this in the *“Properties”* dialog of an object, on the *“Compile”* tab.

If no object is inserted below the Visualization Manager, then the visualization created there is displayed automatically as an integrated visualization when the application starts.

Exception handling at runtime

If an error or an exception occurs in a visualization at runtime, the execution of the visualization is stopped without stopping the execution of the application. An error screen appears informing you of this. In addition, the error screen (usually) enables you to restart the visualization. This exception handling takes place from visualization profile `CODESYS V3.5 SP7`, compiler version `3.5.7.0` and a runtime system from version `3.5.7.0`.

Select the command *"Stop Execution at Handled Exceptions"* in order to investigate the cause of the occurrence of exceptions and the error position.

See also

- *Chapter 1.4.1.20.3.6.19 "Command 'Stop Execution on Handled Exceptions'" on page 1043*

Identifying display variants

In order to programmatically identify a display variant, the `VisuFbClientTagDataHelper` library module from the `VisuElemBase` library is available to you. The library itself is referenced in `VisuElems`. The library module is typically called with `VisuElems.VisuFbClientTagDataHelper`.

Further information on this library module can be found in its documentation in the library manager.

See also

- *Chapter 1.4.1.8.7 "Using Library POUs" on page 265*

1.4.5.16.1 Executing as CODESYS WebVisu

NOTICE!

Recommendations for data security

In order to minimize the risk of breaches of data security, we recommend the following organizational and technical measures for the system on which your applications run:

As far as possible, avoid exposing the PLC and control networks to open networks and the Internet. For protection, use additional data-link layers such as a VPN for remote access and install firewall mechanisms. Limit access to authorized persons, change any existing standard passwords during the initial commissioning and continue to change them regularly.

If you nevertheless wish to publish your web visualization, it is **urgently** recommended that you provide it at least with simple **password protection** in order to prevent anyone accessing your control functionality over the Internet. (See an example in the project `SimpleWebvisuLogin.project`, which is provided with the standard installation of the development system).

Use the latest versions of the gateway server and the web server.

You can execute a visualization as CODESYS WebVisu.

The requirement for this is that the runtime system contains a web server with WebVisu support. This enables communication between target system and web browser. The web server on the target system is started as soon as an application with WebVisu configuration is started and runs until all applications with WebVisu are ended. The device can then display visualizations in connected HTML5-capable web browsers.

The web-based display variant of the CODESYS Visualization enables remote access to a plant as well as its remote monitoring, service and diagnosis over the Internet. A web browser communicates by Java Script (optionally with SSL encryption) with the web server in the controller and displays the visualization by means of HTML5. This technology is supported by virtually all browsers and is thus also available on terminal devices with iOS or Android.

Configuring and starting display variants

☑ An executable visualization `visMain` exists in the project.

1. Select the object *"Visualization manager"* and select the command *"Add object"*.
2. Select the object *"WebVisu"* and enter the name `WebVisu_A`.

   ⇨ There is a new object in the device tree underneath the object *"Visualization Manager"*. The associated editor opens.

   The visualization task `VISU_TASK` is automatically added under the task configuration.
3. Select the visualization `visMain` in the *"Start Visualization"*.
4. In *"Name of .htm file"*, enter the name `webvisuA`.
5. Click on *"Show used visualizations"* and check whether the selected visualization is activated for a download to the associated device.

   ⇨ The visualization is configured. The settings under *"Scaling options"* determine the window size and the scaling.
6. Start a suitable runtime system with web server and WebVisu support.

   Configure the communication settings for your system.

   ⇨ The runtime system runs.
7. Compile, load and start the application.

   ⇨ The application and the web server run.
8. Start a web browser with the following address: `http://localhost:8080/webvisuA.htm`

   ⇨ The page is displayed and you can see the data of the application and operate the application.

See also

- ↳ *Chapter 1.4.5.19.4.7 "Object 'TargetVisu'" on page 1787*

Calling a page in the web server

☑ Requirement: A visualization with WebVisu is started.

1. Start a current browser with JavaScript and support of HTML5-Canvas, e.g. Firefox, Chrome, IE>=9.
2. Enter the following address in the web browser:

   `http://localhost:8080/webvisu.htm`

   Formal: `http://<IP address of webserver>:<port of webserver>/<name of HTM-file>`

   `<name of HTM-file>` is the HTML start page of the visualization defined in the object *"WebVisu"*.

   ⇨ The page is displayed and you can see the data of the application and operate the application.

Identifying WebVisu

In order to be able to identify a WebVisu with the help of the library block `VisuFbClientTagDataHelper`, the WebVisu needs a name. In order to be able to specifically address it in the application, expand the URL call by the parameter `ClientName=<Name>`.

Example: `http://localhost:8080/webvisu.htm?ClientName=VisClientxy`.

See also

- *Chapter 1.4.5.19.4.8 “Object 'WebVisu'” on page 1788*

1.4.5.16.2 Executing as an Integrated Visualization

You can execute the visualization as an integrated visualization. In this case a display variant of the visualization runs on the development system **without** the visualization code being loaded to the controller.

Use the integrated visualization for the testing and diagnosis of your application, or for the service and commissioning of a plant.

The requirement for this is that there are no objects under the visualization manager. Alternatively, any objects located there can be excluded from compilation. You can configure an individual object accordingly in its dialog *“Properties”* on the tab *“Compile”*.

See also

- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Configuring and starting display variants

☑ A visualization project is open.

1. Remove all objects from underneath the visualization manager or exclude the objects from compilation.
   ⇨ The `VISU_TASK` has been removed from under the task configuration.
2. Load the application to the controller.
   ⇨ Now no visualization code will be transferred on loading the application.
3. Start the application.
   ⇨ The visualization in the visualization editor is being executed. You can operate your application.

Use the command “Activate keyboard usage” in order to toggle between the keyboard usage of the integrated visualization and the keyboard usage of CODESYS.

See also

- *Chapter 1.4.5.19.2.4 “Command 'Activate Keyboard Usage'” on page 1722*
- *Chapter 1.4.5.19.1 “Keyboard Shortcuts for Default Keyboard Action” on page 1717*

Restrictions in the variable output

Numerical variable values, which are output within a text in an integrated visualization, are displayed according to the current display format. You can select the display format with the command *“Debug ➔ Display”*.

See also

- *Chapter 1.4.1.20.3.7.24 “Command 'Display Mode' - 'Binary', 'Decimal', 'Hexadecimal'” on page 1058*

Data server restrictions

A variable value that is transferred via the data server is **not** output. The integrated visualization only outputs the initialization or the last transferred value.

The integrated visualization thus only enables a passive observation of the application.

Restrictions in variable types

`VAR_INPUT` variables behave like integrated visualizations such as `VAR_IN_OUT` variables during execution.

Restrictions in expressions and monitoring

Only the following expressions, which are also used in the monitoring mechanism of the development system, are supported in an integrated visualization.

Variable access:

- Example: `PLC_PRG.myPou.nCounter`

Array access:

- Access to an array of scalar data types, where a variable is used as an index
  Example: `a[i]`
- Access to an array of complex data types (structure, function block, array), where a variable is used as an index
  Example: `a[i].x`
- Access to a multidimensional array of all kinds of data types with one or more variable indices
  Example: `a[i, 1, j].x`
- Access to an array with constant index
  Example: `a[3]`
- Accesses like those described above in which simple operators are used for the calculations inside the index brackets.
  Example: `a[i+3]`
- Nested combinations of the complex expressions listed above
  Example: `a[i + 4 * j].aInner[j * 3].x`

Operators in index calculations:

- `+`, `-`, `*`, `/`, `MOD`

Pointer monitoring:

- Example: `p^.x`

Methods and function calls are not supported **with the exception** of the following:

- Standard string functions
- Type conversion functions
  Example: `INT_TO_DWORD`
- Operators such as `SEL`, `MIN`, etc.

Restrictions in the input action *“Execute ST-Code”*

When the input action *“Execute ST-Code”* is called, only a list of assignments is supported.

If a list of assignments is used, the value of the left-hand side is not assigned until the next cycle. Processing in the next row immediately afterwards is not possible.

Example

```
PLC_PRG.n := 20 * PLC_PRG.m; // Don't use this!
                IF PLC_PRG.n < MAX_COUNT THEN
                PLC_PRG.n := PLC_PRG.n + 1;
                END_IF
                //Use the following!
                PLC_PRG.n := MIN(MAX_COUNT, PLC_PRG.n + 1);
```

Restrictions in the interface of a visualization

No interface (`INTERFACE`) may be declared in the interface editor of a visualization.

1.4.5.16.3 Configure File Transfer Mode

When downloading, usually files required by the visualization for displaying, are transmitted to the respective display unit. These are especially image files or text list files.

Alternatively, you can configure,that the visualization accesses local files. So no files are transferred with a download

The following configuration is required to allow the visualization access to local files:

- The file paths for image files or text files lists are relative.
- The link type for image files is *“Link to file”*.

Using local visualization files

☑ Requirement: You have opened a visualization project with a image pool.

1. Open the image pool.
2. Select for each image under *“Link Type”* the setting *“Link to file”*.
   ⇨ The image is linked.
3. Select the command *“Project ➔ Project Setting”* and select the category *“Visualization”*.
4. Insert in tab *“General”* in *“Image files”* the local paths of the image files with relative path names.
   ⇨ Example: `.\;.\images\`

   Note: When no path is specifiet, the setting in dialog box *“Options”*, category *“Visualization”*, tab *“File Options”* setting *“Image files”* is usesd.
5. Open the visualization manager.
6. Activate under *“Extended settings”* the option *“Visible”*.
7. Activate under *“File Transfer Mode”* the option *“Use local visualization files”*.
   ⇨ When downloading, no files are transferred. When displaying the visualization, the local files are used.

See also

- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*
- *Chapter 1.4.1.20.2.13 “Object 'Image Pool'” on page 873*
- *Chapter 1.4.5.19.3.13 “Dialog 'Project Settings' - 'Visualization'” on page 1766*
- *Chapter 1.4.5.19.3.9 “Dialog Box 'Options' - 'Visualization'” on page 1763*

1.4.5.17 Applying Visualization Styles

A visualization style is a collection of colors, fonts, images, and any values that are defined as style properties. When designing a visualization element, you can use these style properties only. The you have a uniform, style-dependent appearance.

An element that applies style colors and style fonts behaves according to the selected style design in each selected style. In this way, a style property, such as `Element basic color`, can be blue In one style and gray in another style. In contrast, if the color of an element has a fixed value, this color is fixed even when the style is switched.

All applicable styles are consistent because they define a fixed set of style properties. Therefore, you can switch smoothly between styles in order to customize your visualization. You can preview a style to get an impression of how it behaves.

CODESYS provides different styles, for example the styles `Flat style` and `White Style`.

These provided styles are installed in the visualization style repository.

The selected style that applies to all visualizations in the application is set in the *"Visualization Manager"* object (*"Settings"* tab, *"Style Settings"* group, *"Selected style"*). In addition, the *"Properties"* view provides its style properties when designing an element. For each element, you can assign these styles instead of fixed values.

The style is applied to all visualizations that are below an application. The settings of the *"Options - Visualization Styles"* dialog are also considered for a library visualization or a visualization in the POUs view.

See also

- I ↳ *Chapter 1.4.5.19.3.11 "Dialog 'Project Environment' - 'Visualization Styles'" on page 1765*
- ↳ *Chapter 1.4.5.19.3.7 "Dialog 'Options' - 'Visualization Styles'" on page 1761*
- ↳ *Chapter 1.4.5.19.4.2 "Object 'Visualization manager'" on page 1777*

Designing visualization elements with style properties

The set style includes style properties. These are provided in the *"Properties"* view of an element in the drop-down list of the *"Value"* column. It is checked which style properties are appropriate for which property. For example, only style properties with color definitions are available for a color assignment.

A style can have directly defined visualization element properties. If this style is used in the project, then these properties are not configurable anymore.

☑ Requirement: A project is open with a visualization.

1. Double-click the visualization.
2. Select an element.
3. Choose *"View ➔ Element Properties"*.
4. Click in the input field of a color in the window *"Properties"* (category *"Colors"*).
   ⇨ The list box opens with style properties. The style colors are based on the currently selected style.
5. Select a style property.
   ⇨ The visualization shows the element according to the style.

Example

A visualization uses the style `CompanyStyle8`, which defines the colors `CompanyRed`, `CompanyBlue`, and `CompanyGreen`. An element is selected in the visualization. You can configure the element in the *"Properties"* view. By clicking into the value field of the *"Color"* property, you receive a drop-down list with the entries `CompanyRed`, `CompanyBlue`, and `CompanyGreen`.

Switching visualization styles

When setting a style in the visualization manager, all complete styles in the repository are available for selection. It does not matter and it is not evident if a style have been derived from another style.

You can preview a style to get an impression of how it behaves.

How a visualization implements a style at runtime also depends on the display variant. For example, if a font that is defined in the style is not available, the display variant shows the visualization with a preset font.

☑ Requirement: A project is open with a visualization.

1. Double-click the *"Visualization Manager"* object in the device tree.
   ⇨ The editor opens.
2. Click in the input field of *"Selected style"* (*"Settings"* tab, *"Style Settings"* group).
   ⇨ All styles that are installed in the repository are listed.
3. Mouse over a style.
   ⇨ A preview of how the style is displayed appears in a new window.
4. Select a style.
   ⇨ The style is applied. The preview in *"Style Settings"* shows the new setting.
5. Double-click a visualization.
   ⇨ The visualization appears in the new style.

Updating versions

☑ Requirement: A project is open with a visualization.

1. Click *"Project ➔ Project Environment"*, *"Visualization Styles"* tab.
   ⇨ CODESYS lists all new versions of the currently used styles.
2. Click *"Set All to Newest"*.
   ⇨ The style is updated. Visualizations and their elements apply the new style.

1.4.5.17.1 Editing visualization styles in the visualization style editor

A style is an XML file with the file extension `*.visustyle.xml`. It contains a specific set of style properties. CODESYS checks the style properties in the consistency check.

You can create a new style or customize an existing style. The visualization style editor is available for this.

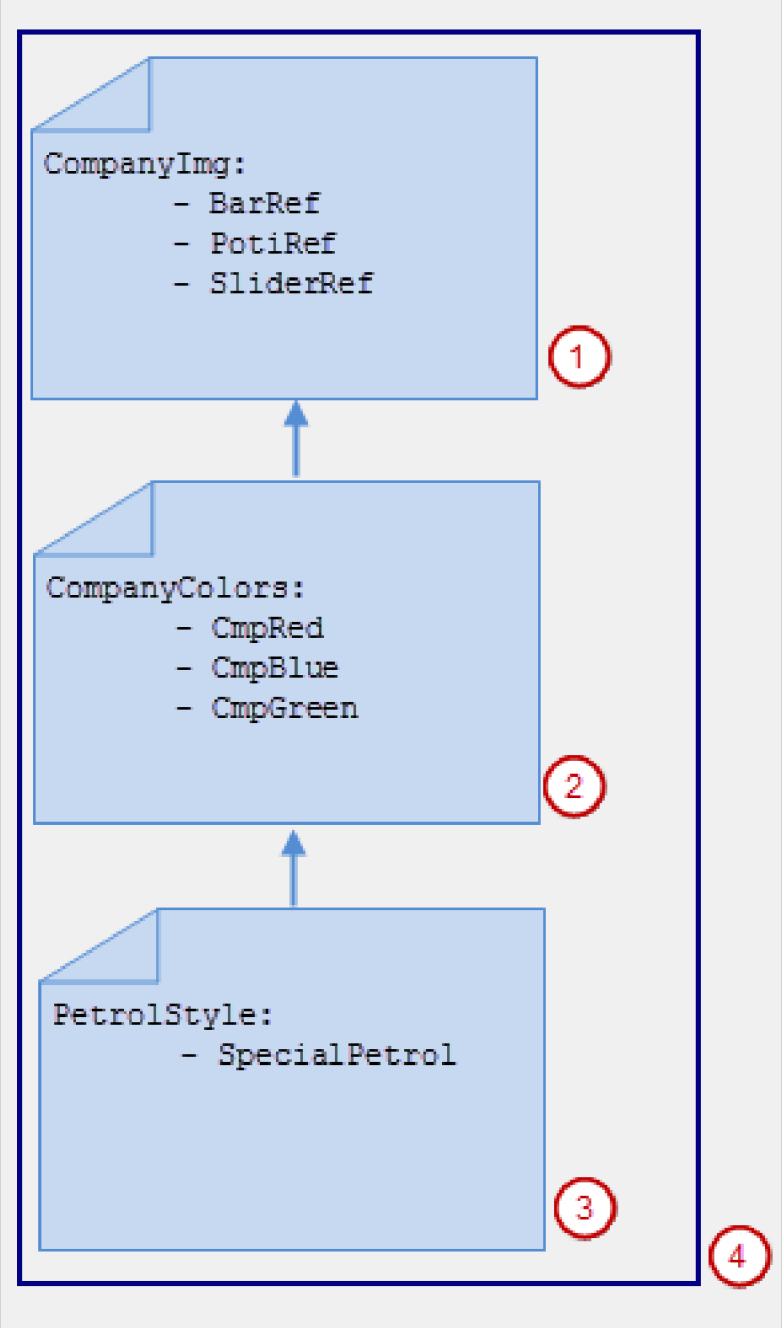
If you customize a style, then a new style is created as a hierarchy of styles. A hierarchy consists of at least two styles based on each other. The nesting depth is unrestricted. A hierarchy is identified simply with its top derived style. You can derive multiple different styles from one base style by extending the styles by differing style properties. This save memory and therefore should be your **preferred method**.

A base style does not have to be consistent for itself. Instead, you must identify it as an incomplete style. Only the top derived style must be consistent.

Example of a style hierarchy

Style `Petrostyle`

In a partial style, you can combine any style properties to form efficient hierarchies without having to worry about consistency. For example, you can collect all image references into one partial style. Then you derive the style and define more style properties for colors. This style is also incomplete. You derive the style again and define more style properties for its fonts. The top style is now completely.



- (1): `CompanyImg` is a partial style defining image references.
- (2): `CompanyColor` is a partial, derived style based on `CompanyImg` and also defines colors.
- (3): `PetrolStyle` is a complete, derived style based on `CompanyColor` and also defines a special color.
- (4): The hierarchy of styles comprises `PetrolStyle`, `CompanyColor`, and `CompanyImg`.

In the visualization style editor, you can open a style, define its style properties, and localize its name. If the style is consistent, then you can install it in the visualization style repository. The editor is not integrated in CODESYS. However, you can start the editor in CODESYS.

Names for style properties

A style property is an entry for a specific color, a specific font, or a specific image reference.

If this name contains a dash, then the Visualization Style Editor can sort the style properties by the prefixed terms before the dash and display them in a hierarchy. Otherwise the names can be sorted in alphabetical order or sequential order or in sequential order (as saved in the XML file). CODESYS displays the style properties in the order of names actually saved in the XML file for the style.

Example: `Element-Alarm-Fill-Color`

See also

- *Chapter 1.4.5.20.3 “Editor 'Visualization Style Editor'” on page 2128*

Starting the editor in CODESYS

1. Double-click the *“Visualization Manager”* object.
   ⇨ The editor opens.
2. Click the symbol (*“Settings”* tab, *“Style Settings”* group).
3. Click *“Open Style Editor”* from the drop-down list.
   ⇨ The *“Visualization Style Editor”* opens.

Starting the editor independent of CODESYS

▷ Choose *“Visualization Styles Editor”* from the CODESYS install folder in the Start menu. If you have a standard installation, then this link is located in `CODESYS` (the program folder for CODESYS).
   ⇨ The visualization style editor opens.

Deriving visualization styles

This is the recommended way to create a style that combines existing style properties with new ones.

Starting the editor in CODESYS and deriving styles

☑ Requirement: CODESYS is open with a project containing a visualization.

1. Double-click the *“Visualization Manager”* object in the device tree.
   ⇨ The editor opens.
2. Click the symbol (*“Settings”* tab, *“Style Settings”* group).
   ⇨ A list of commands opens.
3. Choose *“Create and Edit Derived Style”*.
   ⇨ The visualization style editor starts and the *“Create a New Visualization Style”* dialog box opens.
4. Type a name.

5. Select a directory.
6. Select a base style. The default style is set in CODESYS. You can also select a style from the repository.
   ⇨ The new style appears in the visualization style editor. The style properties from the base style are displayed yellow.
7. Add a new style property or modify an existing value.
8. Provide a version for the style and click *"File ➔ Save and Install"*.
   ⇨ The style is installed in the repository. The memory requirement is low because only the style property added in step 7 is saved.

Copying visualization styles

☑ Requirement: CODESYS is open with a project containing a visualization.

1. Double-click the *"Visualization Manager"* object in the device tree.
   ⇨ The editor opens.
2. Click the symbol (*"Settings"* tab, *"Style Settings"* group).
   ⇨ A list of commands opens.
3. Choose *"Copy and Edit Style"*.
   ⇨ The visualization style editor starts and the *"Open Existing Style as a Copy"* dialog box opens.
4. Select which style should be copied (*"Style"*).
5. Type a directory in "Destination" and click *"OK"*.
   ⇨ The new style appears in the visualization style editor. All style properties are identical to those in the copied style.
6. Type a name.
7. Add a new style property or modify the value of an existing style property.
8. Provide a version for the style and click *"File ➔ Save and Install"*.
   ⇨ The style is installed in the repository and the style properties are identical to the added style property, except for the style property added in step 8. The memory requirement is high because the common style properties are defined in both styles.

Creating new visualization styles

☑ Requirement: CODESYS is open with a project containing a visualization.

1. Double-click the *"Visualization Manager"* object in the device tree.
   ⇨ The editor opens.
2. Click the symbol (*"Settings"* tab, *"Style Settings"* group).
   ⇨ A list of commands opens.
3. Click *"Open Style Editor"*.
   ⇨ The visualization style editor opens.
4. Click *"File ➔ New Style"*.
   ⇨ The *"Create a New Visualization Style"* dialog box opens.
5. Type a name. Specify a base style.

6. Specify a directory and click *“OK”* to close the dialog box.
   ⇨ The new style appears in the visualization style editor.
7. Add a new style property.
8. Provide a version for the style and click *“File ➔ Save and Install”*.
   ⇨ The style is installed in the Visualization Styles Repository with the version number.

Adding a style property

Using the visualization style editor, you can edit a style to save and install as a new version.

☑ Requirement: The visualization style editor is open with a style.

1. Select a style property and click *“Styles ➔ New Style (Afterwards)”* in the *“Style Properties”* tab.
   ⇨ A new style property is added.
2. In the *“General”* tab, type a new version number in the *“Version”* setting.
3. Choose *“File ➔ Save and Install”*.
   ⇨ The changes are saved and the style is installed in the repository as the new version.

Localizing style properties

You can assign a language-dependent name to a style property. CODESYS displays a style property by its localized name, depending on the language settings in category *“International Settings”* (menu *“Tools ➔ Options”*).

☑ Requirement: The visualization style editor is open with a style.

1. Translate the name of the style property into the localized language in the *“Localization”* tab.
2. Provide a version for the file in the *“General”* tab.
3. Choose *“Save and Install”*.
   ⇨ The edited style is installed in the repository currently selected in CODESYS.
4. Update the style.
5. Set the language settings in CODESYS to the localized language.
6. Open a visualization and select an element. The style settings in its properties are displayed in the localized language.

1.4.5.17.2 Managing visualization styles in repositories

The styles that are listed in CODESYS in the drop-down lists of different dialogs and editors are all checked for consistency and installed in the visualization style repository. For derived styles, the hierarchy is checked completely and all styles of the hierarchy are installed. The repository is a version control system within the development system.

You can open a style as write-protected from the visualization style repository in the visualization style editor. The *“Save”* and *“Save and Install”* commands are not available there for read-only files. However, you can derive it as the basis for a new style or as a copy.

See also

- *Chapter 1.4.5.19.2.20 “Command 'Visualization Style Repository'” on page 1742*
- *Chapter 1.4.5.20.3 “Editor 'Visualization Style Editor'” on page 2128*

Installing styles to repositories

1. Click *“Tools ➔ Visualization Style Repository”*.
   ⇨ The *“Visualization Styles”* dialog box opens.
2. Select the *“System”* repository in the drop-down list of *“Storage location”*.
   ⇨ All versions of the installed styles are listed in *“Installed Visualization Styles”*.
3. Click on the *“Install”* button.
   ⇨ The *“Select Visualization style(s)”* dialog box opens.
4. Select a style file and click *“Open”* to close the dialog box.
   ⇨ The style is installed in the *“System”* repository. It appears now in the tree view below *“Installed Visualization Styles”*.

Uninstalling styles

1. Click *“Tools ➔ Visualization Style Repository”*.
   ⇨ The *“Visualization Styles”* dialog box opens.
2. Select a repository in the drop-down list of *“Storage location”*.
   ⇨ All versions of the installed styles are listed in *“Installed Visualization Styles”*.
3. Select a style there.
4. Click the *“Uninstall”* button.
   ⇨ The *“Select Visualization Style(s)”* dialog box opens.

Managing repositories

1. Click *“Tools ➔ Visualization Style Repository”*.
   ⇨ The *“Visualization Styles”* dialog box opens.
2. Click on the *“Edit Locations”* button.
   ⇨ The dialog makes it possible to manage other repositories.

1.4.5.18 Reference, Programming

1.4.5.18.1 Visualization Elements

Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'

Symbol:

Category: *“Basic”*

The *“Rectangle”*, *“Rounded Rectangle”*, and *“Ellipse”* are the same type of element. They can be converted into another element type by changing the *“Element type”* property.

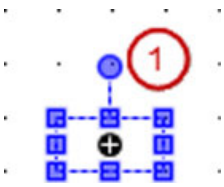
Element properties

“Element name”	Optional Example: `Werkstueck_3` Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Rectangle”*, *“Rounded Rectangle”*, *“Ellipse”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Y”	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Width”	Specified in pixels Example: `150`
“Height”	Specified in pixels Example: `30`
	Tip: You can change the values in *“X”*, *“Y”*, *“Width”*, and *“Height”* by dragging the corresponding symbols to another position in the editor.

"Angle"	Static angle of rotation (in degrees) Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Radius setting

Visible only when *"Rounded Rectangle"* is selected in the *"Type of element"* property.

"Radius"	Rounding of the corners. *"From style"* *"Relative to the element size"* *"Explicit"*: Allows for specifying a custom value in the *"Value"* setting.
"Value"	Radius of the rounded corners (in pixels) Example: `5` Requirement: *"Explicit"* is selected in the *"Radius"* setting.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

"Normal state"	The normal state is in effect if the variable in *"Color variables ➔ Toggle color"* is not defined or it has the value `FALSE`.
"Frame color"	Frame and fill color for the corresponding state of the variable.
"Fill color"	
"Transparency"	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
"Alarm state"	The alarm state is in effect if the variable in *"Color variables ➔ Toggle color"* has the value `TRUE`.
"Use gradient color"	☑: The element is displayed with a gradient of two colors.
"Gradient setting"	The *"Gradient editor"* dialog box opens.

See also

- ↳ *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Fill attributes"	The way in which the element is filled. ● *"Filled"*:The element is filled with the color from property *"Colors ➔ Fill color"*. ● *"Invisible"*: The fill color is invisible.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- ↳ *" Element property 'Appearance variables'" on page 1430*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1373*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

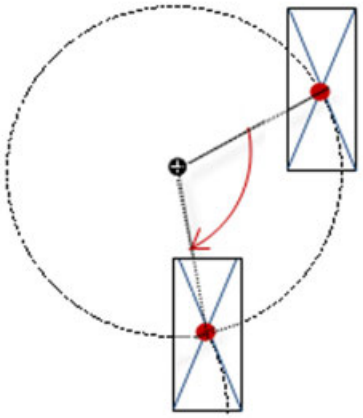
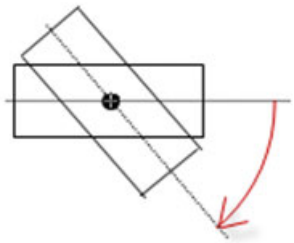
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 1371*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1370*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

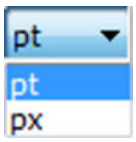
“Text list”	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
“Text index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
“Tooltip index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1371*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

“Toggle color”	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *“Color”* property. ● `TRUE`: The element is displayed with the color specified in the *“Alarm color”* property. Assignment options: ● Placeholder for the user input variable – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
“Normal state” *“Alarm state”*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *“Color variables”*, *“Toggle color”* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *“Colorvariables”*, *“Toggle color”* has the value `TRUE`.
“Frame color”	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
“Filling color”	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the “Activate semi-transparent drawing” option of the visualization manager is selected.

Select the “Advanced” option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Element property 'Appearance variables'

The properties contain IEC variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels).
“Fill attributes”	Variable (DWORD). Controls whether the fill color of the element is visible. ● Variable value = `0`: Filled ● Variable value > `0`: Invisible; no fill color
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible; no line

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1382*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *“Configure”* button opens the *“Input Configuration”* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *“Execute ST Code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog.
“OnMouseClick”	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the mouse button.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

“Hotkey”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Events”* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
“Key”	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
“Events”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the “Keyboard Configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Line'

Symbol:

Category: *“Basic”*

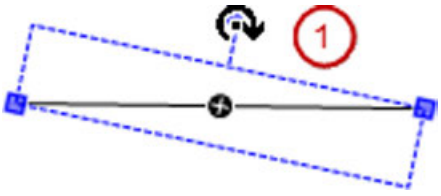
The element draws a simple line.

Element properties

“Element name”	Optional. Example: `Separator_Header` Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Line”*

Element property 'Position'

The following properties define the position and length of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“Dots”	*“[0]”*: Coordinates of the starting point *“[1]”*: Coordinate of the end point You can also change the values by dragging the box symbols () to other positions in the editor.
“Angle”	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow  , you can rotate the element about its center as a handle. Example:  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *“Absolute movement ➔ Internal rotation”*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the  symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color of the line in normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color of the line in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- ↳ *" Element property 'Appearance variables'" on page 1430*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1385*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

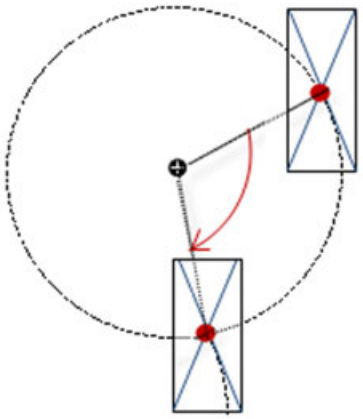
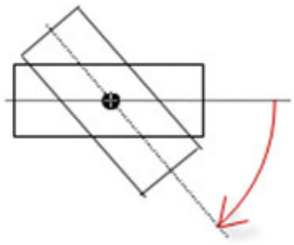
Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement point[0]” ● *“X”* ● *“Y”*	Variable (numeric data type). It contains the number (in pixels) that the starting point of the line is moved. Incrementing the X value moves the element to the right. Incrementing the Y value moves the element to the down.
“Movement point[1]” ● *“X”* ● *“Y”*	Variable (numeric data type). It contains the number (in pixels) that the end point of the line is moved. Incrementing the X value moves the element to the right. Incrementing the Y value moves the element to the down.

See also

- *“Element property 'Absolute movement'” on page 1383*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1383*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

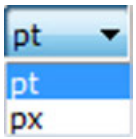
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1383*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable in the alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*
- *Chapter 1.4.5.19.4.2 "Object 'Visualization manager'" on page 1777*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

Element property 'Line width variable'

Dynamic definition of the weight of a line element using a variable.

"Integer value "	Variable (integer data type). Defines the line weight of the element (in pixels). This overwrites the fixed value that is defined in *"Appearance ➔ Line weight"*. Note: The value 0 codes the same as 1 and sets the line weight to one pixel.

Element property 'Line style variable'

"Integer value "	Variable (integer data type). Defines the appearance of the line at runtime. ● 1: Solid ● 2: Dashes ● 3: Dots ● 4: Dash Dot ● 5: Dash Dot Dot ● 6: Invisible: The line is not drawn.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Polygon', 'Polyline', 'Bézier Curve'

Symbol:

Category: *“Basic”*

The *“Polygon”*, *“Polyline”*, and *“Bézier Curve”* are the same element type. They can be converted into another type by changing the *“Element type”* property.

Elements can be dragged to the editor. The element is then drawn with five points: [0] to [4].

Other positions are added as follows: Move the mouse pointer over a corner point; the mouse pointer changes shape. Now if you press and hold *[Ctrl]* and click the left mouse button, another point is created. You can delete a point by pressing and holding *[Shift]+[Ctrl]* and click the selected point.

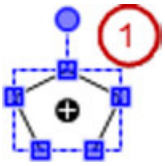
As an alternative, you can select the element in the toolbox area and in the editor click multiple times. At the same time, a connecting line is drawn from one point to the other. End by double-clicking the element or right-clicking it one time.

Element properties

“Element name”	Optional. Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Werkstueck_1`
“Type of element”	● *“Polygon”* ● *“Polyline”* ● *“Bézier Curve”*

Element property 'Position'

The following properties define the position of the corner points in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"Dots"	[0]..[n]: Coordinates of the corner points Specified in pixels You can also change the values by dragging the box symbols (⊞) to other positions in the editor.
"Angle"	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow ↻, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

"Normal state"	The normal state is in effect if the variable in *"Color variables ➔ Toggle color"* is not defined or it has the value `FALSE`.
"Frame color"	Frame and fill color for the corresponding state of the variable.
"Fill color"	
"Transparency"	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
"Alarm state"	The alarm state is in effect if the variable in *"Color variables ➔ Toggle color"* has the value `TRUE`.
"Use gradient color"	☑: The element is displayed with a gradient of two colors.
"Gradient setting"	The *"Gradient editor"* dialog box opens.

See also

- *Chapter 1.4.5.19.3.5 “Dialog 'Gradient Editor'” on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

“Line width”	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Fill attributes”	The way in which the element is filled. ● *“Filled”*:The element is filled with the color from property *“Colors ➔ Fill color”*. ● *“Invisible”*: The fill color is invisible.
“Line style”	Type of line representation ● *“Solid”* ● *“Dashes”* ● *“Dots”* ● *“Dash Dot”* ● *“Dash Dot Dot”* ● *“not visible”*

You can assign variables in the “Appearance variables” property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- *“ Element property 'Appearance variables'” on page 1430*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Text”*.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Tooltip”*.

See also

- *“Element property 'Text variables'” on page 1398*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

Element property 'Text properties'

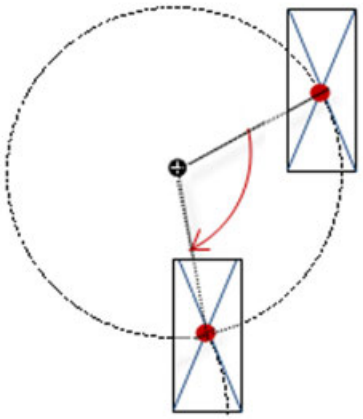
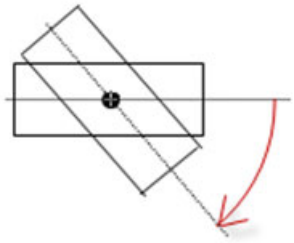
The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Dynamic points'

“Array of points”	Variable (`POINTER TO`). Points to an array of the structure `VisuElems.VisuStructPoint`. The elements `iX` and `iY` of `VisuStructPoint` contain the xy-coordinates of a point The current number of array elements implicitly contains the variable in the property *“Number of points”*. The variable that is assigned to the property *“Number of points”* contains the number of array elements and therefore the number of corner points. Example: `pPoints : POINTER TO ARRAY[0..100] OF VisuElems.VisuStructPoint;`
“Number of points”	Variable (integer data type): Contains the number of array elements and therefore the number of corner points for displaying the element. Example: `PLC_PRG.iNumberOfPoints := 24;` In the example, the element has 24 points. This definition is necessary because the individual points are defined by a pointer and this does not allow control over the number of points. Note: In this way, it is possible to adapt the display of the element dynamically by updating the number of corner points.

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1395*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

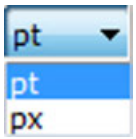
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in "Text properties".

See also

- *"Element property 'Text properties'" on page 1396*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assignment options: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Normal state" *"Alarm state"*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *"Color variables"*, *"Toggle color"* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *"Colorvariables"*, *"Toggle color"* has the value `TRUE`.
"Frame color"	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
"Filling color"	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Appearance variables'

The properties contain IEC variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels).
“Fill attributes”	Variable (DWORD). Controls whether the fill color of the element is visible. ● Variable value = 0: Filled ● Variable value > 0: Invisible; no fill color
“Line style”	Variable (DWORD). Controls the line style. Coding: ● 0: Solid line ● 1: Dashed line ● 2: Dotted line ● 3: Line type "Dash Dot" ● 3: Line type "Dash Dot Dot" ● 8: Invisible; no line

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1423*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (BOOL). Toggles the visibility of the element. TRUE: The element is not visible at runtime. Example: bIsVisible with VAR bIsVisible : BOOL := FALSE; END_VAR
“Deactivate inputs”	Variable (BOOL). Toggles the operability of the element. TRUE: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Pie'

Symbol:

Category: *“Basic”*

The element draws a pie of any angle.

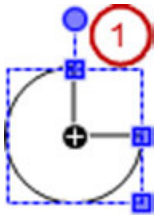
Element properties

“Element name”	Example: `Error_rate_part_1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Pie”*

Element property 'Position'

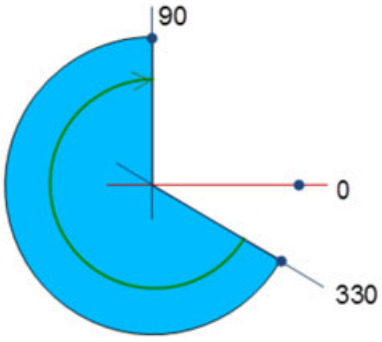
The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Y”	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Width”	Specified in pixels Example: `150`
“Height”	Specified in pixels Example: `30`
	Tip: You can change the values in *“X”*, *“Y”*, *“Width”*, and *“Height”* by dragging the corresponding symbols to another position in the editor.

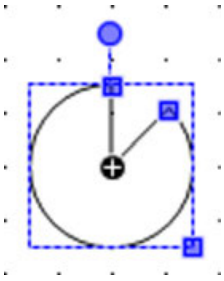
“Angle”	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *“Absolute movement ➔ Internal rotation”*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Begin”	Start angle of the pie. If you also define a variable for the start, then the start angle is calculated from the sum of the values for *“Begin”* and *“Variable for begin”*.	Example: • *“Begin”*: 330 • *“End”*: 90 
“End”	End angle of the pie. If you also define a variable for the end, then the end angle is calculated from the sum of the values for *“End”* and *“Variable for end”*. The pie is drawn clockwise from the start angle to the end angle.	
“Variable for begin”	The start of the sector is defined dynamically by a variable.	
“Variable for end”	The end of the sector is defined dynamically by a variable.	
“Only show circle line”	☑: The pie is drawn without the radius line or filling color.	

Element property 'Center'

"X" *"Y"*	Display of the center coordinates. You cannot modify these values here in the properties. If the Pie is selected in the editor, then the center of the Pie (as well as the center of the enveloping box) is visualized with the symbol ⊕. Moreover, the element is decorated with a position, begin, and end boxes that you can move.  The center coordinates change when you move the center symbol ⊕ in the editor. This also changes the size of the Pie so that the position box retains its position and the center remains in the middle of the element.

Element property 'Colors'

"Normal state"	The normal state is in effect if the variable in *"Color variables ➔ Toggle color"* is not defined or it has the value `FALSE`.
"Frame color" *"Fill color"*	Frame and fill color for the corresponding state of the variable.
"Transparency"	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
"Alarm state"	The alarm state is in effect if the variable in *"Color variables ➔ Toggle color"* has the value `TRUE`.
"Use gradient color"	☑: The element is displayed with a gradient of two colors.
"Gradient setting"	The *"Gradient editor"* dialog box opens.

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Fill attributes"	The way in which the element is filled. ● *"Filled"*:The element is filled with the color from property *"Colors ➔ Fill color"*. ● *"Invisible"*: The fill color is invisible.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- ↳ *" Element property 'Appearance variables'" on page 1430*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- ↳ *"Element property 'Text variables'" on page 1411*
- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- ↳ *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

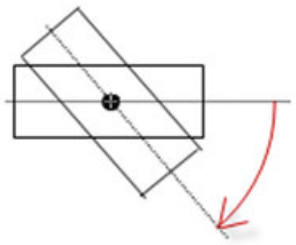
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (integer data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (integer data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.

"Interior rotation"	Variable (integer data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. If a static angle of rotation is specified in *"Position ➔ Angle"*, then the static angle of rotation and the angle of rotation are added.	

You can link the variables to a unit conversion.

The "X", "Y", and "Interior rotation" properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1409*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

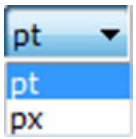
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1409*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assignment options: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Normal state" *"Alarm state"*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *"Color variables"*, *"Toggle color"* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *"Colorvariables"*, *"Toggle color"* has the value `TRUE`.
"Frame color"	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
"Filling color"	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Appearance variables'

The properties contain IEC variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels).
“Fill attributes”	Variable (DWORD). Controls whether the fill color of the element is visible. ● Variable value = `0`: Filled ● Variable value > `0`: Invisible; no fill color
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible; no line

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1423*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Image'

Symbol:

Category: *"Basic"*

The element adds an image to the visualization. The displayed image is managed in the image pool and referenced in the visualization element by means of a static ID. You can also change the displayed image dynamically by using a variable instead of the static ID.

With the "Background" command, you can define a background for the entire visualization.

Directories that contain the images for use in visualizations can be defined in the project settings (category "Visualization").

Element properties

"Element name"	Example: `Status bar` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Image"*
"Static ID"	Identifier of the image file for a static assignment ID of the image file on, as it is defined in the corresponding image pool. If the image is not included in the global image pool in the POU view, then the instance path must be specified. Then the name of the image pool is preceded to make the entry unique. Example: `imagepool2.button_image`. When a new ID is specified, a file selection dialog opens. The selected file is saved to the *"GlobalImagePool"*. See also: Help for the *"Image Pool"* object.
"Show frame"	☑: The image file is displayed with a frame.
"Clipping"	Requirement: The *"Scaling type"* property is *"Fixed"*. ☑: Only part of the visualization is displayed that fits in the element frame.
"Transparent"	☑: The image pixels that have the *"Transparent color"* are displayed as transparent.
"Transparent color"	Effective only if the *"Transparent"* option is activated. The … button opens the color selection dialog. This is where you select the transparent color.

"Scaling type"	Definition of how an image fits in the element frame. ● *"Isotropic"*: The entire image is displayed in the element frame, either larger or smaller. As a result, the proportion of height and width are retained. If the alignment of the elements to each other should also be retained within a scaled frame element, then note the following. Unwanted horizontal or vertical offsets can be prevented by setting the properties *"Horizontal alignment"* and *"Vertical alignment"* to *"Centered"*. The alignment of the elements is retained and there are no resulting horizontal or vertical offsets. Example: A lamp is centered above a switch. The lamp should remain in the horizontally centered position, even if the frame is resized. ● *"Anisotropic"*: The image resizes automatically to the dimensions of the element frame, filling the entire element frame. As a result, the proportions are not retained. ● *"Fixed"*: The image retains its original size, even if the element frame is resized. Note also that the *"Clipping"* option is selected. For each reassignment of an image ID, the element size is adapted automatically to the image size.
"Horizontal alignment"	Horizontal alignment of the element within the element frame: ● *"Left"* ● *"Centered"* ● *"Right"* Requirement: The scaling type of the image is *"Isotropic"* or *"Fixed"*. Note: If the visualization is referenced, then the horizontal alignment takes effect within the frame position. : The *"Variable"* property is shown below this.
"Variable"	Enumeration variable (`ENUM VisuElemBase.VisuEnumVerticalAlignment`). Contains the horizontal alignment. Example: `PLC_PRG.eHorizontalAlignment`
"Vertical alignment"	Vertical alignment of the element within the element frame: ● *"Top"* ● *"Centered"* ● *"Bottom"* Requirement: The scaling type of the image is *"Isotropic"* or *"Fixed"*. Note: If the visualization is referenced, then the horizontal alignment takes effect within the frame position. : The *"Variable"* property is shown below this.
"Variable"	Enumeration variable (`ENUM VisuElemBase.VisuEnumVerticalAlignment`). Contains the vertical alignment. Example: `PLC_PRG.eVerticalAlignment`

Example A valid declaration is required for the variables used as an example in the table above.

Enumeration

```
TYPE VisuElemBase.VisuEnumHorizontalAlignment
    LEFT
    HCENTER
    RIGHT
END_TYPE

TYPE VisuElemBase.VisuEnumVerticalAlignment
    DOWN
    VCENTER
    BOTTOM
END_TYPE
```

Declaration

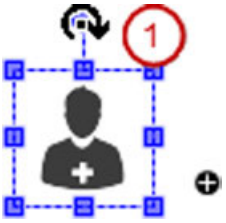
```
PROGRAM PLC_PRG
VAR
    eHorizontalAlignment :
VisuElemBase.VisuEnumHorizontalAlignment :=
VisuElemBase.VisuEnumHorizontalAlignment.HCENTER;
    eVerticalAlignment : VisuElemBase.VisuEnumVerticalAlignment :=
VisuElemBase.VisuEnumVerticalAlignment.VCENTER;
END_VAR
```

See also

- *Object 'Image Pool'*

Element property 'Position' The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Y"	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Width"	Specified in pixels Example: `150`
"Height"	Specified in pixels Example: `30`
	Tip: You can change the values in *"X"*, *"Y"*, *"Width"*, and *"Height"* by dragging the corresponding symbols to another position in the editor.

"Angle"	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the frame Requirement: *"Show frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the frame in alarm state Requirement: *"Show frame"* property is activated. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- ↳ *" Element property 'Appearance variables'" on page 1430*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- ↳ *"Element property 'Text variables'" on page 1426*
- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- ↳ *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Image ID variable'

"Image ID"	Variable (`STRING`). Contains the image ID. The contents of the string corresponds to the description of the *"Static ID"* property. Example: `PLC_PRG.stImageID := 'ImagePool_A.Image3';`

See also

- *Chapter 1.4.5.19.5.5 "Visualization Element 'Image'" on page 1842*
- *Chapter 1.4.1.20.2.13 "Object 'Image Pool'" on page 873*

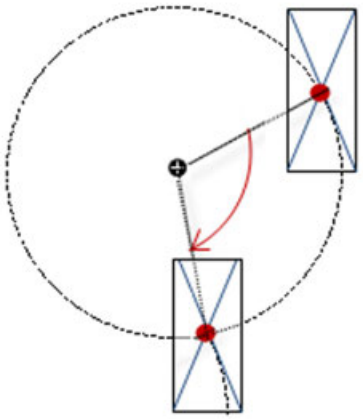
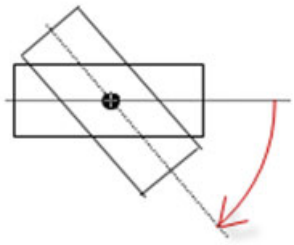
Element property 'Dynamic image'

You can use this element property for animating a series of image files.

"Bitmap version"	Variable (integer data type). Contains the version of the image. If the variable changes, then the visualization re-reads the image referenced in the *"Image ID"* property and displays it. The visualization displays animations when the image file on the controller is updated continuously, thus incrementing the version variable. The application must be programmed for this. Possible applications ● Displaying graphics that are generated by the application ● Displaying images that are refreshed by a camera

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed. 

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 1396*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1423*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

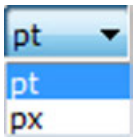
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1423*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	Color variable for the frame ● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Requirement: *"Show frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable for the frame in alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Element property 'Appearance variables'

The properties contain variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels). Note: The values 0 and 1 both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible: The line is not drawn.

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1423*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. User input is a user event from the perspective of the element.

The *"Configure"* button opens the *"Input configuration"* dialog box for creating or modifying a user input configuration.

A configuration contains one or more input actions for the respective input event. Existing input actions are displayed below it.

Example: *"Execute ST code"*: `PLC_PRG.i_x := 0;`

"OnDialogClosed"	Input event: The user closes the dialog box.
"OnMouseClick"	Input event: A user clicks the element completely. The mouse button is clicked and released.
"OnMouseDown"	Input event: A user clicks down on the element only.
"OnMouseEnter"	Input event: A user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: A user drags the mouse pointer away from the element.
"OnMouseMove"	Input event: A user moves the mouse pointer over the element area.
"OnMouseUp"	Input event: The user releases the mouse button over the element area.

See also

● *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

“Hotkeys”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Event(s)”* property are triggered.
“Key”	Key pressed for input action. Example: *[T]*
“Event(s)”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed in the “Keyboard configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Project Settings - Visualization*
- *Chapter 1.4.5.19.2.10 “Command 'Background'” on page 1728*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Frame'

Symbol:

Category: *“Basic”*

The element serves as a frame in which to display one or more already existing visualizations. You get a structured user interface. The size of the frame can be fixed or scaled. The display area of the referenced visualization then adapts itself to the frame size.

Element properties

“Element name”	Example: `refVisUserInfo`
“Type of element”	*“Frame”*
“Clipping”	☑: Fixed size. Only that part of the referenced visualization that fits inside the frame is displayed. Requirement: The *“Scaling type”* property is *“Fixed”*.
“Show frame”	Displays the frame ● *“No frame”*: The displayed area of the frame does not have borders. ● *“Frame”*: The displayed area of the frame has borders. ● *“No frame with offset”*: The displayed area of the frame does not have a border and the displayed area of the referenced visualization is reduced inwards by one pixel as compared to the frame area. The gap prevents the referenced visualization from touching any adjacent elements.

“Scaling type”	The method with which the height and width of the **referenced** visualization are scaled. ● *“Isotropic”*: The visualization is scaled to the size of the element. The visualization retains its proportions with a fixed height/width ratio. ● *“Anisotropic”*: The visualization is scaled to the size of the element. The height and width are adapted to the element independently of each other. ● *“Fixed”*: the visualization is displayed in its original size without taking into account the size of the element. ● *“Fixed and scrollable”*: The visualization is displayed fixed in the element. If it is larger than the element, the element will be provided with scrollbars. Please note: assign variables to the properties *“Scroll position variable horizontal”* or *“Scroll position variable vertical”*. You can then edit the data of the scrollbar position in the application.

Element properties 'Scrollbar settings'

The properties contain variables for the position of the scrollboxes in the scrollbars. You can then edit the data of the scrollbox position in the application.

Requirement: the property *“Scaling type”* is *“fixed and scrollable”*.	

“Scroll position variable horizontal”
“Scroll position variable vertical”
	Variable (integer data type, also as array). Contains the position of the horizontal or vertical scrollbox. The array contains the position for every display variant. If the visualization runs on several display variants, then the position changes are decoupled from each other.
	Example:
	`PLC_PRG.iScrollHor[CURRENTCLIENTID]`
	`PLC_PRG.iScrollVer[CURRENTCLIENTID]`
	The variable is declared as an array in the example.
	`iScrollHor: ARRAY[0..20] OF INT;`
	`iScrollVer: ARRAY[0..20] OF INT;`
	`CURRENTCLIENTID` indexes the current display variant.

You can combine the variables with a unit conversion.

See also

- *Unit conversion*

“Deactivation of the background character”	☐: The background is drawn. The non-animated element of the referenced visualization is drawn as a background bitmap in order to optimize the performance of the visualization. Consequence: Elements can be displayed in an unexpected order at runtime. For example, an animated element can push itself behind the Frame at runtime. ☑: Background character is deactivated in order to avoid the behavior described above.

Element property 'References' Contains the currently configured visualization references as a subnode

“References”	Clicking *“Configure”* opens the *“Frame Configuration”* dialog. This is used to manage the referenced visualizations. Caution: Visualizations can be nested at any depth by means of Frame elements. In order to use the *“Switch to any visualization”* Frame selection type without any problems, a Frame must not contain more than 21 referenced visualizations. For more information, see also the description for the *“Input configuration”* of an element: Action *“Switch Frame visualization”*.
List of the currently referenced visualizations	Visualizations that have a button also have this displayed as a subnode. Each interface variable is listed with the currently assigned transfer parameters. Example: `vis_FormA` • `iDataToDisplay_1 : PLC_PRG.iVar1` • `iDataToDisplay_2 : PLC_PRG.iVar2` Hint: You can change the assignment of the variables to an interface variable here and edit the value field. Or click the *“Configure”* button instead.

See also

- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*
- *Chapter 1.4.5.15 “Creating a structured user interface” on page 1321*
- *“Input action 'Switch Frame Visualization'” on page 1756*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for the colors.

"Color"	Color of the frame ▼: Selection list with style colors appears ...: Standard dialog *"Color"* opens for selecting a color. Please note: the normal state is when the boolean variable in the property *"Color variables ➔ Toggle color"* is not defined or its value is `FALSE`.
"Alarm color"	Color with which the element is filled during the alarm state. Please note: Alarm state is when the value of the boolean variable in the property *"Color variables ➔ Toggle color"* is `FALSE`.
"Transparency"	Integer number (value range from `255` to `0`). Specifies the transparency of the associated color. `255`: The color is opaque. `0`: The color is fully transparent. Please note: If the color is a style color and already contains a transparency value, then this property is write-protected.

See also

- ↳ *Chapter 1.4.5.3.3 "Assigning a color" on page 1258*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- ↳ *" Element property 'Appearance variables'" on page 1443*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1439*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

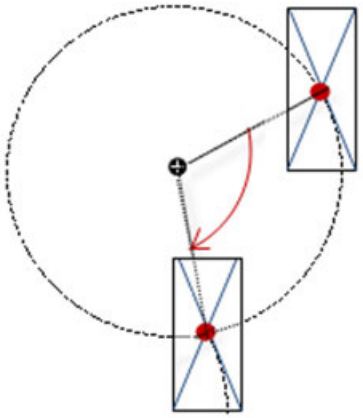
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.

You can link the variables to a unit conversion.

The properties "X", "Y", "Rotation", and "Interior rotation" are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*"Position"* property). The shape of the element can change.

"Movement top-left"	
"X"	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
"Y"	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
"Movement bottom-right"	

"X"	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
"Y"	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *"Element property 'Absolute movement'" on page 1396*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1436*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

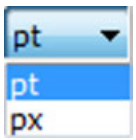
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1437*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	Color variable for the Frame ● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Requirement: *"Show Frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable for the Frame in alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Element property 'Appearance variables'

The properties contain variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels). Note: The values 0 and 1 both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible: The line is not drawn.

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1436*

Element property 'Switch frame variable'

The variable controls the switching of the referenced visualizations. This variable indexes one of the referenced frame visualizations and this is displayed in the frame. When the value of the variable changes, it switches to the recently indexed visualization.

“Variable”	● Variable (integer data type) that contains the index of the active visualization Example: `PLC_PRG.uiIndexVisu` Hint: The *“Frame Configuration”* dialog includes a list of referenced visualizations. The visualizations are automatically numerically indexed via the order in the list. Note: This variant of switching usually affects all connected display variants. ● Array element (integer data type) for index access via `CURRENTCLIENTID` Example: `PLC_PRG.aIndexVisu[CURRENTCLIENTID]` Note: This variant of switching applies to the current client only, and therefore only on one display variant. That is the display variant where the value change was triggered (for example, by means of user input).

See also

- *Chapter 1.4.5.19.2.9 “Command 'Frame Selection'” on page 1727*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.
"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.15 "Creating a structured user interface" on page 1321*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Visualization Element 'Label'

Symbol:

Category: *"Common Controls"*

The element is used to label visualizations.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Header_Parameter`
"Type of element"	*"Label"*

Element property 'Texts'

The property requires a character string.

This text is entered automatically into the `GlobalTextList` text list and can be localized there.

"Text"	Character string (without single straight quotation marks) Example: `Main page`

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

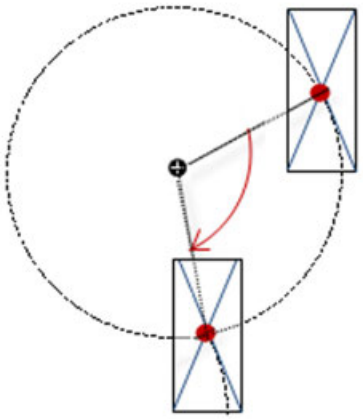
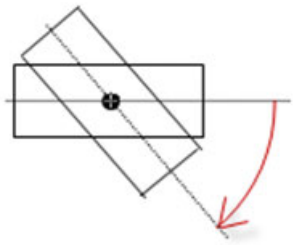
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.

"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Combo Box, Integer'

Symbol:

Category: *"Common Controls"*

The element shows values as a list box. When the user clicks an entry, the ID of the entry is written to an integer variable. The entries in the list box can be from a list and contain images from an image pool.

Element properties

"Element name"	Example: `List of product numbers` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Combo Box, Integer"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	At runtime, the text list ID of the list entry that the user clicks is saved at runtime. If only one image pool is displayed, then the image ID is saved. Property value ● Variable (integer data type) Example: `PLC_PRG.iIDComboboxEntry` ● Enumeration variable with text list support Example: `PLC_PRG.eMyCombobox<COMBO>`
"Text List"	Displayed as a combo box. Every text list entry becomes a combo box entry. Note: A maximum of 32766 entries can be displayed. Transfer value ● Text list identifier as string Example: `'TextList_A'` Note: The IDs of the text list have to be within the range of values of `DWORD` or `DINT`. ● Blank – When an enumeration variable with text list support is specified in the *"Variable"* property – When only one image pool is displayed
"Image Pool"	Displayed as a combo box. Every image in the image pool becomes a combo box entry. Example: `'ImagePool_A'`

See also

- *Enumerations*
- *Chapter 1.4.5.6 "Setting Up Multiple Languages" on page 1286*

Element property 'Settings of the list'

Displayed list that expands when a visualization user clicks into the element.

"Number of rows setting"	● *"From style"*: ● *"Explicit"*: Then the *"Number of visible rows"* property appears below it.
"Number of visible rows"	Number of visible lines of the combo box drop-down list defined here ● Integer literal Example: `5` ● Variable (integer data type) Example: `PLC_PRG.iNumberOfVisibleRows` Note: The property is available when the *"Number of rows setting"* property is set to *"Explicit"*.
"Row height"	● *"From style"*: ● Literal Example: `20`
"Height of image"	Image height (in pixels) of the image displayed in the drop-down list entry ● *"From style"*: ● Integer literal Example: `30` Note: Images are displayed only when a value is specified in the *"Image pool"* property.

"Width of image"	Image width (in pixels) of the image displayed in the drop-down list entry ● *"From style"*: ● Literal Example: `30` Note: Images are displayed only when a value is specified in the *"Image pool"* property.
"Offset of image"	Makes the images in the selection list appear offset (in pixels) from the left margin. An offset of `0` means that the images are displayed directly on the margin. ● *"From style"*: ● Literal Example: `4` Note: Images are displayed only when a value is specified in the *"Image pool"* property.
"Scrollbar size"	Size of the scrollbar (in pixels). The scrollbar is displayed when more entries are specified in the drop-down list than in *"Number of visible rows"*. Default: `20`

Element property 'Texts'

"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element in runtime mode Example: `Products of customer A` Hint: The text is accepted automatically into the *"GlobalTextList"* text list and can be localized there.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Value range'

"Limit valuerange"	Limits the text list to one subrange. This subrange is displayed by the combo box. Requirement: A value is specified in the *"Text list"* property. ☑: Only the subrange that is defined by the *"Minimum value"* *"Maximum value"* properties is displayed as a drop-down list.
"Minimum value"	ID of the text list entry from which a combo box entry is displayed ● Literal (`ANY_NUM`) Example: `5` ● Variable (integer data type) Example: `PLC_PRG.iFirstEntry`

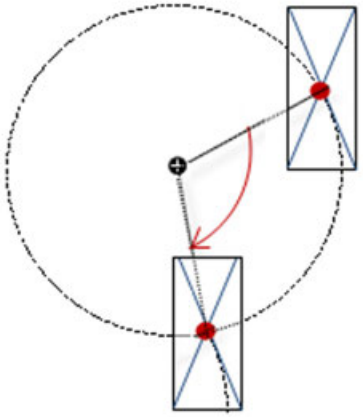
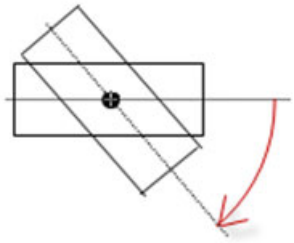
"Maximum value"	ID of the text list entry up to which combo box entries are displayed ● Literal (`ANY_NUM`) Example: `10` ● Variable (integer data type) Example: `PLC_PRG.iLastEntry`
"Filter missing textentries"	: Text list is refreshed and any unused texts (IDs) are removed. Requirement: A value is specified in the *"Text list"* property.

Element property 'Text properties' The properties contain fixed values for the text properties.

"Usage of"	● *"Default style values"*: The values of the visualization style are used. ● *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Font"	Example: *"Default"* : The *"Font"* dialog box opens. : Drop-down list with style fonts.
"Font color"	Example: *"Black"* : The *"Color"* dialog box opens. : Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement' The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

● *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Combo Box, Array'

Symbol:

Category: *“Common Controls”*

The element shows values of an array as a list box. When the visualization user clicks an entry, the array index of the entry is written to an integer variable.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `List_Product_Number`
“Type of element”	*“Combo Box, Array”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10.`
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10.`
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	The array index of the list entry that the user clicks is saved at runtime. Property value ● Variable (integer data type) Example: `PLC_PRG.iIndexComboboxEntry` ● Enumeration variable with text list support Example: `PLC_PRG.eMyCombobox<COMBO>` Note: Value range of the enumeration value that lies within the `DWORD` or `DINT` value range
"Data array"	Displayed as a combo box. Every array component becomes a combo box entry. Property value ● Array variable (`ARRAY[...] OF`) Example: `PLC_PRG.astrCombobox` Declaration: `astrCombobox : ARRAY[0..4] OF STRING := ['First', 'Second', 'Third', 'Fourth'];`

See also

- *Enumerations*
- *Chapter 1.4.5.6 "Setting Up Multiple Languages" on page 1286*

Element property 'Columns'

The "Combo box – Array" element visualizes an array variable or structure variable in a tabular view. The index of array elements or structure members is shown in a column or row. Two-dimensional arrays or structure arrays are shown in several columns. You specify the visualized variable in the *"Data array"* property. If a variable is assigned there, then you can specify the display of the table columns where the array elements are shown. You can customize each column that is assigned to an index [<n>].

"Columns" ● [<n>]	Due to the structure of the variable that is defined in *"Data array"*, CODESYS determines the number of columns and defines them with the index <n>. Example: `StringTable : ARRAY [0..2, 0..4] OF STRING := ['BMW','Audi','Mercedes','VW','Fiat', '150','150','150','150','100','blue','gray','silver','blue','red'];`: three columns are formed `[0]`, `[1]` and `[2]`.
"Max. array index"	Optional. Variable (integer data type) or value. Defines up to which array index the data is displayed.
"Row height"	Height of the rows (in pixels).
"Number visible rows"	Optional. If the array is larger than the number of visible rows, then a scrollbar is included.
"Scrollbar size"	Width of the vertical scrollbar (in pixels).

Table 270: “Element property 'Columns: Column [<n>]'”

“Width”	Column width (in pixels).
“Image column”	☑: Images can be displayed in the column. Images are used from the global image pool or user-defined image pools. The image IDs are shown in the cells of the table as defined in the image pool.
“Image configuration”	
“Fill mode”	• *“Fill cell”* The image resizes to the dimensions of the cell without fixing the height/ width ratio. • *“Centered”* The image is centered in the cell and retains its proportions (height-width ratio).
“Transparency”	☑: The color that is specified in *“Transparent color”* is displayed as transparent.
“Transparent color”	When the *“Transparent”* property is enabled, the color specified here is not displayed. Pixels with this color are transparent.
“Text alignment in column”	• *“Left”* • *“Centered”* • *“Right”*

Element property 'Texts'

“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element in runtime mode Example: `Products of customer A` Hint: The text is accepted automatically into the *“GlobalTextList”* text list and can be localized there.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element property 'Text properties'

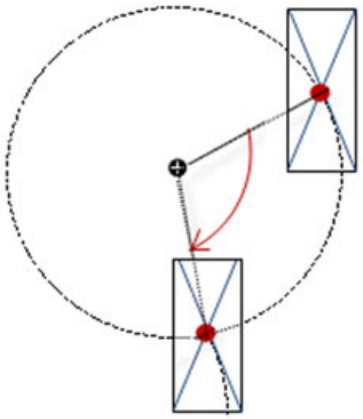
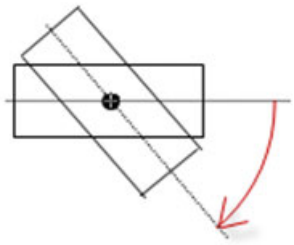
The properties contain fixed values for the text properties.

“Usage of”	• *“Default style values”*: The values of the visualization style are used. • *“Individual settings”*: The "Individual text properties" property group is shown The values can be customized here.
“Individual text properties” Requirement: The *“Individual settings”* text property is defined.	
“Font”	Example: *“Default”* ...: The *“Font”* dialog opens. ▼: List box with style fonts

"Font color"	Example: *"Black"* ...: The *"Color"* dialog opens. ▼: List box with style colors
"Transparency"	Integer (value range from `0` to `255`). This determines the transparency of the respective color. `255`: The color is opaque. `0`: The color is completely transparent. Note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Tabs'

Symbol:

Category: *"Common Controls"*

The element displays selected visualizations in tabs. The tabs can be used by means of the tab header without having to configure an input configuration. A visualization user switches between visualizations by clicking the tab header.

Element properties

"Element name"	Example: `Assembly A` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Tabs"*

"Tab width"	Width of the tab (in pixels). If there is not space for all tab headers, then a scroll bar is added. Example: `30`
"Tab height"	Height of the tab (in pixels) ● Integer literal Example: `15` ● *"From style"*

"Scaling type"	The method with which the height and width of the **referenced** visualization are scaled. ● *"Isotropic"*: The visualization is scaled to the size of the element. The visualization retains its proportions with a fixed height/width ratio. ● *"Anisotropic"*: The visualization is scaled to the size of the element. The height and width are adapted to the element independently of each other. ● *"Fixed"*: the visualization is displayed in its original size without taking into account the size of the element. ● *"Fixed and scrollable"*: The visualization is displayed fixed in the element. If it is larger than the element, the element will be provided with scrollbars. Please note: assign variables to the properties *"Scroll position variable horizontal"* or *"Scroll position variable vertical"*. You can then edit the data of the scrollbar position in the application.

"Deactivate background drawing"	☐: The non-animated elements of the referenced visualization are displayed as background images in order to optimize the performance of the visualization. Result: At runtime, the elements can be displayed in any order, for example when an element moves behind the frame at runtime. ☑: Deactivates the background display in order to prevent the behavior described above The property is not available for the following settings: ● The *"Scaling type"* property is set to *"Fixed and scrollable"* ● The client animation functionality is enabled.

Element property 'Scroll bar settings'

The properties include variables for the position of the scroll boxes in the scroll bars. You can process the data for the scroll box position in the application.

Requirement: The *"Scaling type"* property is *"Fixed and scrollable"*.	

"Scroll position variable horizontal"
"Scroll position variable, vertical"
	Variable (integer data type, also array). Includes the position of the horizontal or vertical scroll box. The array contains the position for each display variant. If the visualization is running on multiple display variants, then the position changes are disconnected from each other.
	Example:
	`PLC_PRG.iScrollHor[CURRENTCLIENTID]`
	`PLC_PRG.iScrollVer[CURRENTCLIENTID]`
	In this example, the variable is declared as an array:
	`iScrollHor: ARRAY[0..20] OF INT;`
	`iScrollVer: ARRAY[0..20] OF INT;`
	`CURRENTCLIENTID` indicates the current display variant.

See also

- *Unit conversion*

Element property 'References'

"References"	Clicking *"Configure"* opens the *"Frame Configuration"* dialog. You can select an existing visualization there. Selected visualization references are shown in the properties. Selected visualization references are listed here as subordinate properties.
Name pf the visualization reference (example: `PLC_PRG.S1`)	
"Heading"	Tab caption (example: `Panel`)
"Image ID"	Image ID in the theme `<image pool name>.<ID>` Example: `Imagepool_A.1` for the image with ID `1` in `Imagepool_A`
Interface parameter of the visualization reference Example: `iX`	If the visualization has an interface, then their parameters are displayed here as subordinate properties. Variable (data type conforms to data type of the interface parameter). Includes the initialization value for the instantiation of the visualization.

See also

- *Chapter 1.4.5.15 "Creating a structured user interface" on page 1321*
- *Chapter 1.4.5.19.2.1 "Command 'Interface Editor'" on page 1719*
- *Chapter 1.4.5.19.2.9 "Command 'Frame Selection'" on page 1727*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Switch frame variable'

“Variable”	Variable (integer data type). Specifies the index of the active visualization. Example: `PLC_PRG.uiActiveVisuID`. Tip: The *“Frame Configuration”* dialog box includes a list of selected visualizations. The visualizations are ordered automatically in numeric order in the list.

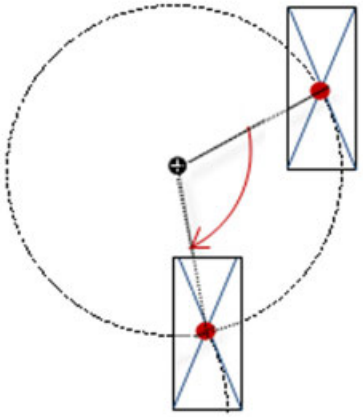
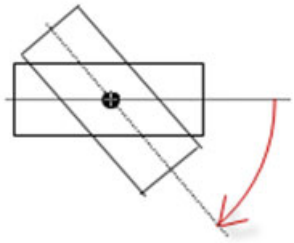
See also

- *Chapter 1.4.5.19.2.9 “Command 'Frame Selection'” on page 1727*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Button'

Symbol:

Category: *“Common Controls”*

The element triggers an action, such as setting a variable.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Voltage_on`
“Type of element”	*“Button”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the element in its normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the element in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.
"Use gradient color"	☑: The element is displayed with a color gradient.
"Gradient setting"	The *"Color gradient editor"* dialog box opens.

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Image'

"Static ID"	Reference to an image in an image pool of the format <name of image pool>.<image ID> (example: `image_pool.GreenButton`). If the image is from the *"GlobalImagePool"*, then you can omit the name of the image pool because CODESYS always searches this pool first. …: The *"Input Assistant"* dialog box opens and lists all available image pools and images in the entire project.
"Scale type"	Behavior of the image when resizing the button. ● *"Isotropic"*: The image retains its proportions. The ratio of height to width is retained, even if you change the height or width of the button separately. ● *"Anisotropic"*: The image resizes to the dimensions of the button. ● *"Fixed"*: The image retains its original size, even if you change the size of the button.
"Transparency"	The visualization displays the image with the transparency color that is selected in *"Transparency color"*.
"Transparency color"	Color that is transparent in the image (example: *"White"*). if the image background that is reference by *"Static ID"* is white, then this background is displayed transparent. Clicking … opens a color selection dialog. Requirement: The *"Transparency"* option is activated.
"Horizontal alignment"	Horizontal alignment of the image ● *"Left"* ● *"Centered"* ● *"Right"*
"Vertical alignment"	Vertical alignment of the image ● *"Top"* ● *"Centered"* ● *"Bottom"*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1473*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

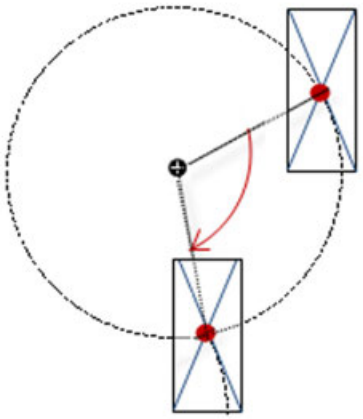
Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`

"Y"	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
"Movement bottom-right"	
"X"	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
"Y"	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *"Element property 'Absolute movement'" on page 1396*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1470*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

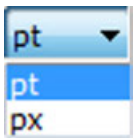
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1471*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	Color variable for the Frame ● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Requirement: *"Show Frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable for the Frame in alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Button state variable'

“Digital variable”	At runtime, the property controls whether the Button is displayed as pressed or not. Values: ● `FALSE`: The Button is displayed as not pressed. ● `TRUE`: The Button is displayed as pressed. Argument passed to the property: ● Placeholder for the user input variable to couple the representation of the Button with the input variable. – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the Button. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `prgA.xButtonState` Note: Implement a value assignment in the code for the variable specified here.

Element property 'Image ID variable'

“Image ID”	Variable (`STRING`). Contains the image ID. The contents of the string corresponds to the description of the *“Static ID”* property. Example: `PLC_PRG.stImageID := 'ImagePool_A.Image3';`

See also

- *Chapter 1.4.5.18.1.5 “Visualization Element 'Image'” on page 1418*
- *Chapter 1.4.1.20.2.13 “Object 'Image Pool'” on page 873*

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *“Configure”* button opens the *“Input Configuration”* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *“Execute ST Code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog.
“OnMouseClick”	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the mouse button.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Group Box'

Symbol:

Category: *“Common Controls”*

The element provides a visual grouping of visualization elements. The group box can have multiple levels of nesting.

> *You can also use drag&drop to add elements to a “Group Box”. To do this, drag the element to the window area of the “Group Box”. The appearance of the cursor changes (a small plus sign is displayed). When you click the [Shift] key at the same time, the element is not added.*
>
> *You can remove elements from the “Group Box” by dragging them out of the window area.*

Element properties

“Element name”	Example: `Parameter axis 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Group Box”*
“Clipping”	☑: Elements that protrude beyond the size of the group box are clipped.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

> *You can also change the values by dragging the box symbols (⊡) to other positions in the editor.*

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

The properties contains character strings for labeling the element.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Example: `Axis 1`.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Parameters of Axis 1`.

See also

- ↳ *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

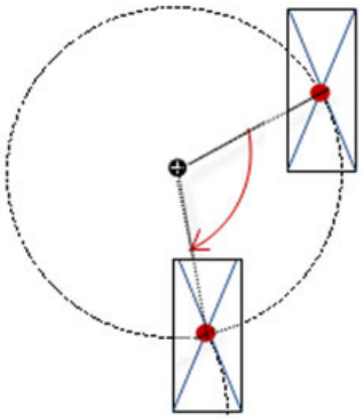
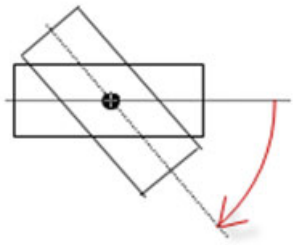
Element property 'Text properties'

The properties contain fixed values for the text properties.

“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed. 

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Table'

Symbol:

Category: *“Common Controls”*

The element displays data that can be represented as an array in a table. Therefore, the data type of the visualizing variable can be 1) a one-dimensional array, 2) a maximum two-dimensional array, 3) an array of an array, 4) an array of structures, or 5) an array of a function block.

Element properties

“Element name”	Example: `Data set component 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	Table
“Data array”	Array whose data is visualized as a table Variable (`ARRAY`) whose data type determines the number of columns and rows in the table Array types ● One-dimensional array: The table has one column. ● Two-dimensional array: The second dimension determines the number of columns. ● Array of an array: The number of array elements of the back array determines the number of columns. ● Array of a structure: The number of structure members determines the number of columns. ● Array of a function block: The number of local variables determines the number of columns. Example: `PLC_PRG.aiTable` Declaration: `aiTable : ARRAY[0..3, 0..4] OF INT := [4(1, 2, 3, 4, 5)];` Hint: If the declaration of the array changes, then the table can be refreshed by placing the cursor in the data array value field and pressing the *[Enter]* key.
“Max. array index”	Top index limit for the displayed table. Limits the number of displayed rows. The index begins at 0. ● Variable (integer data type) Example: `PLC_PRG.iUpperIndexBoundToDisplay` ● Integer literal Example: `4` is displayed as 5 in the row of the table.

See also

- *Data Type 'ARRAY'*

Element property 'Columns'

The *“Table”* element shows the values of a variable in a tabular view. The array elements of structure members are shown in a column or in a row. Two-dimensional arrays or arrays of a structure are shown in multiple columns. The visualized variable is defined in the *“Data array”* property. When a variable is assigned there, you can specify the display of the Table columns where the array elements are shown. An individual configuration is possible for each column that is assigned to an index [<n>].

“Show row header”	☑: The row header is visible. Example: For an array, the index of the array element is displayed in the header.
“Show column header”	☑: The column label is visible.
“Row height”	Height of the rows (in pixels)
“Row header width”	Width of the row label
“Scroll bar size ”	Size of the scroll bar (in pixels)

Table 271: “Element property 'Columns: Column [<n>]'”

“Column header”	By default, the name of the array or structure is applied as the heading with the index or structure member for the column. If an array of a function block has been selected for *“Data array”*, then the name of the array is applied to the column header with the local variables of the function block that belong to the column. The column label can be changed here by specifying a new title.
“Width”	Column width (in pixels)
“Image column”	☑: Images can be displayed in the column. Images are used from the global image pool or custom image pools. The image IDs are shown in the cells of the Table as they are defined in the image pool.
“Image configuration”	
“Fill mode”	• `Fill cell`: The image resizes to the dimensions of the cell without fixing the height/width ratio. • `Centered`: The image is centered in the cell and retains its proportions (height/width ratio).
“Transparency”	☑: The color which is specified in *“Transparent color”* is displayed as transparent.
“Transparent color”	This color is displayed as transparent. Requirement: The *“Transparency”* property is activated.
“Text alignment of header”	Alignment of the column header: • `Left` • `Centered` • `Right`
“Use template”	☑: Another visualization element (type *“Rectangle”*, *“Rounded Rectangle”*, or *“Ellipse”*) is inserted into each line of this Table column. The properties list is extended automatically with the properties of this element in *“Template”*.
“Text alignment of the headline from the template”	Requirement: The *“Use template”* property is activated. ☑: When activated, the settings for font (size) and alignment in the inserted template are also applied to the column header.
“Template”	Requirement: The *“Use template”* property is activated. The properties of all elements assigned to the column are listed in *“Template”*. They can be modified there as described in *“Rectangle”*, *“Rounded Rectangle”*, and *“Ellipse”*.

See also

- *Chapter 1.4.5.18.1.1 “Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'” on page 1368*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.

"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables enable dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog box.
"Size"	Variable (integer data type). Contains the font size (in pixels). Example: `PLC_PRG.iFontHeight := 16;.` The selection of font sizes corresponds to the default *"Font"* dialog box.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Charset”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog box.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

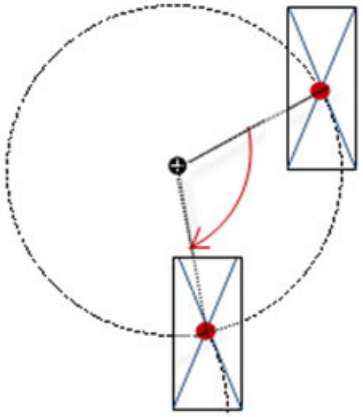
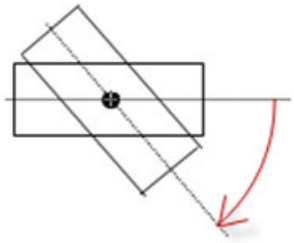
See also

- *“Element property 'Text properties'” on page 1495*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X.` Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y.` Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Selection'

“Background color on selection”	Fill color of the selected row.
“Selection font color”	Font color of the selected row.
“Selection type”	Selection when clicking the table row. ● `No selection`: No selection ● `Cell selection`: The clicked cell only. ● `Row selection`: Row of the clicked cell. ● `Column selection`: Column of the clicked cell. ● `Row and column selection`: Row and column of the clicked cell.
“Frame around selected cells”	☑: A frame is drawn around the selected cells.
“Variable for selected column”	Variable (INT). Contains the array index of the *“Column”* of the selected cell. If the data array points to a structure, then the structure components are indexed, starting at 0. Warning: This index represents the correct position in the array only if no columns have been removed from the table in the display.
“Variable for selected row”	Variable (INT). Contains the array index of the *“Row”* of the selected cell.
“Variable for valid column selection”	Variable (BOOL). TRUE: The *“Variable for selected column”* variable contains a valid value.
“Variable for valid row selection”	Variable (BOOL). TRUE: The *“Variable for selected row”* variable contains a valid value.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.9.1 "Displaying Array Variables in Tables" on page 1298*
- *Data Type 'ARRAY'*

Visualization Element 'Text Field'

Symbol:

Category: *"Common Controls"*

The element is used for the following purposes:

- Static output of text. The contents of a variable can be part of the text.
- Showing a tooltip. The text is managed as static text and can also be defined so that the contents of a variable are also displayed.
- Dynamic output of text. Texts of a text list are displayed dynamically.
- Input of text. For example, a user can input a number or a text literal.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element properties

“Element name”	Optional Example: `FileName_A` Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Text Field”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Colors'

“Normal state”	The normal state is in effect if the variable in *“Color variables ➔ Toggle color”* is not defined or it has the value `FALSE`.
“Frame color”	Frame and fill color for the corresponding state of the variable.

“Fill color”	
“Transparency”	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
“Alarm state”	The alarm state is in effect if the variable in *“Color variables ➔ Toggle color”* has the value `TRUE`.

See also

- *Chapter 1.4.5.19.3.5 “Dialog 'Gradient Editor'” on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

“Line width”	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Fill attributes”	The way in which the element is filled. ● *“Filled”*:The element is filled with the color from property *“Colors ➔ Fill color”*. ● *“Invisible”*: The fill color is invisible.
“Line style”	Type of line representation ● *“Solid”* ● *“Dashes”* ● *“Dots”* ● *“Dash Dot”* ● *“Dash Dot Dot”* ● *“not visible”*

You can assign variables in the “Appearance variables” property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- *“ Element property 'Appearance variables'” on page 1443*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Text”*.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Tooltip”*.

See also

- *“Element property 'Text variables'” on page 1495*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

Element property 'Text properties'

The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1494*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

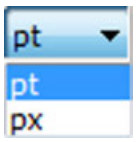
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1495*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

“Toggle color”	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *“Color”* property. ● `TRUE`: The element is displayed with the color specified in the *“Alarm color”* property. Assignment options: ● Placeholder for the user input variable – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
“Normal state” *“Alarm state”*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *“Color variables”*, *“Toggle color”* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *“Colorvariables”*, *“Toggle color”* has the value `TRUE`.
“Frame color”	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
“Filling color”	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the “Activate semi-transparent drawing” option of the visualization manager is selected.

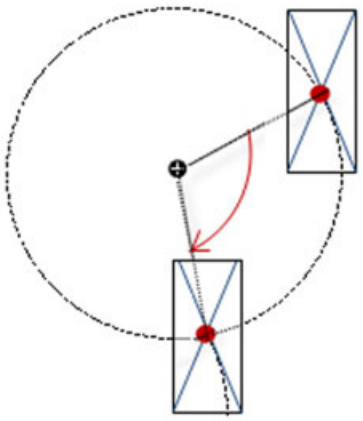
Select the “Advanced” option in the toolbar of the properties view. Then all element properties are visible.

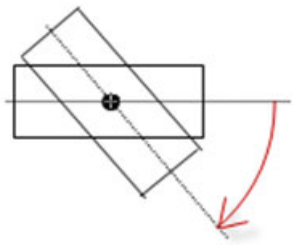
See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Selection and caret configuration'

The variables allow for controlling the caret position and the selection of the text.

"Caret position"	Variable (integer data type). Contains the position of the cursor.
"Selection start"	Variable (integer data type). Contains the position of the first selected character. Example: `PLC_PRG.iSelStart`
"Selection end"	Variable (integer data type). Contains the position of the last selected character. Example: `PLC_PRG.iSelEnd`
"All selected"	Variable (BOOL). Toggles the selection of the entered text. `TRUE`: The text in the text field is selected. `FALSE`: The selection starts with the value in *"Selection start"* and ends with *"Selection end"*.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.

"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.
"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Scroll Bar'

Symbol:

Category: *"Common Controls"*

The element sets the value of a variable, depending on the position of the scroll bar.

Element properties

"Element name"	Example: `Speed Conveyor Belt 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Scroll Bar"*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Value"	Variable as type integer that includes the position of the scroll bar.
"Minimum value"	Smallest value of the scroll bar (fixed value or variable).
"Maximum value"	Largest value of the scroll bar (fixed value or variable).

"Page size"	Page size ● As a fixed value, for example `10` ● As a variable of data type integer Requirement: Visible when the *"Move to click"* property is **not** selected.
"Move to click"	Behavior of the scroll bar at visualization runtime when it is clicked: ☑: The scrollbar moves to the clicked position. ☐: The scrollbar moves to one *"Page size"* in the direction of the click.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (↔) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Bar'

The property defines the representation of scaling and direction of travel.

"Orientation"	Alignment of the slider; defined by the ratio of width to height. ● *"Horizontal"* ● *"Vertical"* You can modify the alignment in the visualization editor by using the pointing device to adjust the width and height of the Scroll Bar.
"Running direction"	The drop-down list varies depending on the alignment of the slider. Horizontal ● *"Left to right"*: Scale starts at the left. ● *"Right to left"*: Scale starts at the right. Vertical ● *"Bottom to top"*: Scale starts at the bottom. ● *"Top to bottom"*: Scale starts at the top.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the element in its normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the element in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *“Element property 'Text variables'” on page 1507*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

Element property 'Text properties'

The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1506*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

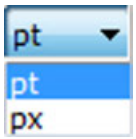
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1495*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assignment options: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Normal state" *"Alarm state"*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *"Color variables"*, *"Toggle color"* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *"Colorvariables"*, *"Toggle color"* has the value `TRUE`.
"Frame color"	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
"Filling color"	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

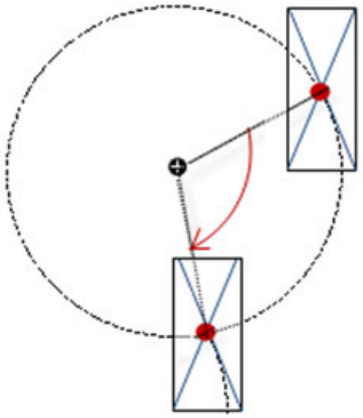
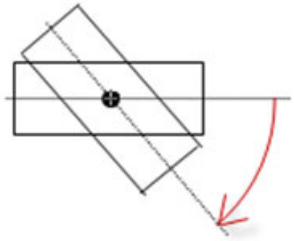
Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

● *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Slider'

Symbol:

Category: *"Common Controls"*

The element changes the value of a variable, depending on the position of the slider within the slider bar. You define the value range of the slider bar by means of the scale start and scale end.

Element properties

"Element name"	Example: `Speed controller conveyor belt 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Slider"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

“Variable”	Variable (numeric data type) Example: `PLC_PRG.rSlider` When executed, the variable assigns a value that corresponds to the position of the slider in the bar.
“Page size”	Page size ● As a fixed value, for example `10` ● As an IEC variable of data type integer Requirement: The *“Move to click”* element property is **not** selected.
“Move to click”	Behavior of the slider at visualization runtime when it is clicked: ☑: The slider moves to the clicked position. ☐: The slider moves to the value (defined in the *“Page size”* element property) in the direction of the click.

Element property 'Scale'

“Show scale”	☑: The element has a visible scale. Note: This option is available for the *“Slider”* only.
“Scale start”	Least value of the scale and the lower limit of the value range for the element. Example: `0` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale start.
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element. Example: `100` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale end.
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two tick marks on the rough scale. Example: `10` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the distance.
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Subscale”	Distance between two dashes on the fine scale. You can hide the fine scale by setting the value to `0`. Example: `2` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the distance.
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 5;`
	`END_VAR`

“Scale format (C Syntax)”	Formatting of the scale label (example: `%d %s`) Note: This property is available for the Slider only.
“Scale proportion”	Size of the scale (in %) of the total size

Element property 'Bar'

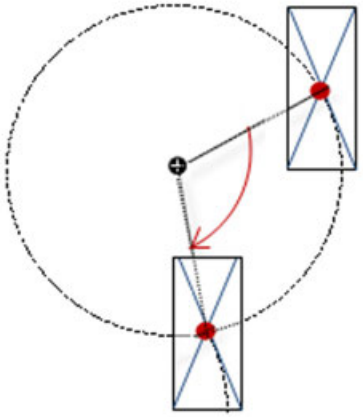
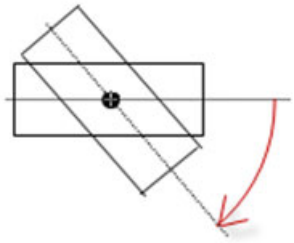
The property defines the representation of scaling and direction of travel.

"Diagram type"	The drop-down list varies depending on the alignment of the diagram. Horizontal ● *"Top"*: Scale is above the slider. ● *"Bottom"*: Scale is below the slider. ● *"Top and bottom"*: Two scales frame the slider above and below. Vertical ● `Left`: Scale is left of the slider. ● `Right`: Scale is right of the slider. ● `Left and right`: Two scales frame the slider on the left and the right.
"Orientation"	Alignment of the slider; defined by the ratio of width to height. ● *"Horizontal"* ● *"Vertical"* You can modify the alignment in the visualization editor by using the pointing device to adjust the width and height of the scrollbar.
"Running direction"	The drop-down list varies depending on the alignment of the slider. Horizontal ● *"Left to right"*: Scale starts at the left. ● *"Right to left"*: Scale starts at the right. Vertical ● *"Bottom to top"*: Scale starts at the bottom. ● *"Top to bottom"*: Scale starts at the top.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Spin Box'

Symbol:

Category: *“Common Controls”*

The element increments or decrements the value of a variable in defined intervals.

Element properties

“Element name”	Example: `Speed controller conveyor belt` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Spin Box”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	Variable (numeric data type) Example: `PLC_PRG.iTemp`
"Number format"	Format of the value in printf syntax Example: %d, %5.2f
"Interval"	Interval used for modification of the value

Element property 'Value range'

"Minimum value"	Lower limit of the output value ● fixed value ● Variable (INT)
"Maximum value"	Upper limit of the output value ● fixed value ● Variable (INT)

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Usage of"	● *"Default style values"*: The values of the visualization style are used. ● *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

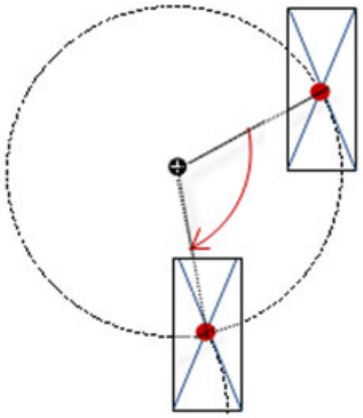
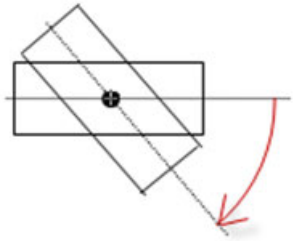
See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The "X", "Y", "Rotation", and "Interior rotation" properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. User input is a user event from the perspective of the element.

The input configuration refers to the text area of the element only, not the two buttons.

The *“Configure”* button opens the *“Input configuration”* dialog box for creating or modifying a user input configuration. A configuration contains one or more input actions for the respective input event. Existing input actions are displayed below it. Example: *“Execute ST code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog box.
“OnMouseClick”	Input event: The user clicks the element completely. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the element only.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.
“OnMouseMove”	Input event: The user moves the mouse pointer over the element area.
“OnMouseUp”	Input event: The user releases the mouse button over the element area.

See also

- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

“Tap”	When a mouse click event occurs, the variable defined in *“Variable”* is described in the application. The coding depends on the options *“Tap FALSE”* and *“Tap on enter if captured”*.
“Variable”	Variable (`BOOL`). Contains the information whether a mouse click event exists. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts while the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *“Tap FALSE”* option is not activated.
“Tap FALSE”	☑: The mouse click event leads to a complementary value in *“Variable”*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
“Tap on enter if captured”	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

“Shift”	When a mouse click event occurs, the variable here is described in the application. When the mouse click event ends, its value is toggled with the *“Toggle on up if captured”* option.
“Variable”	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Tip: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
“Toggle on up if captured”	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

“Hotkeys”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Event(s)”* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
“Key”	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
“Event(s)”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed in the “Keyboard configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Invisible Input'

Symbol:

Category: *"Common Controls"*

This element is displayed in the editor with a dashed line which is not visible in online mode. You define the behavior of the el in the input configuration.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Unsichtbare_Eingabe_1`
"Type of element"	*"Invisible Input"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

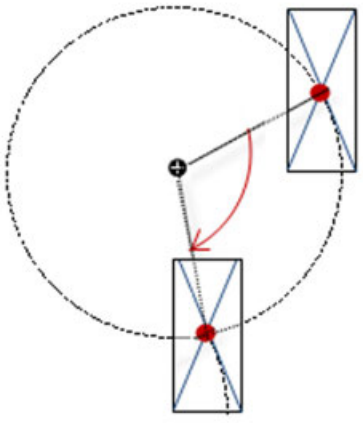
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

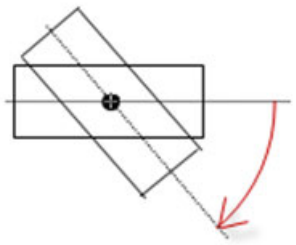
“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *“Configure”* button opens the *“Input Configuration”* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *“Execute ST Code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog.
“OnMouseClick”	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the mouse button.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

“Hotkey”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Events”* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
“Key”	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
“Events”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the “Keyboard Configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Progress Bar'

Symbol:

Category: *“Common Controls”*

The element displays the value of a variable as a progress bar.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Progress_Data_Transfer`
“Type of element”	*“Progress Bar”*
“Text ID”	ID of the global text list Requirement: Text is configured in the property *“Texts ➔ Text”*.
“Variable”	Variable (numeric data type). Represents the length of the progress bar.
“Minimum value”	Value range of the variable
“Maximum value”	
“Style”	● *“Blocks”* ● *“Bar”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊡) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

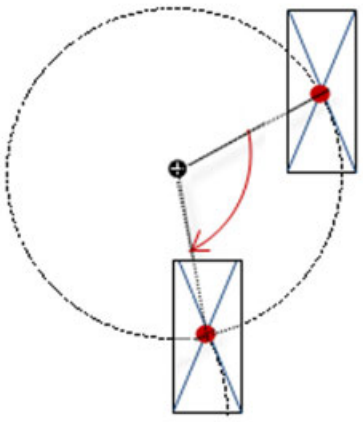
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

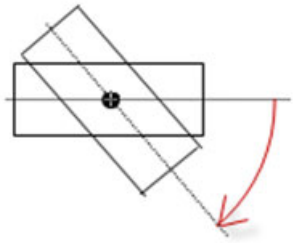
Element property 'Texts'

"Text"	String label for the element. Example: `Zoom`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

● *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Check Box'

Symbol:

Category: *"Common Controls"*

The element is used for setting and resetting a Boolean variable. The set state is identified by a check mark.

Element properties

"Element name"	Example: `signal_tone_for_parts_deficit` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Check Box"*
"Text ID"	ID for the text in the *"GlobalTextList"* Example: `22` The text ID cannot be changed. As soon as you specify and save a text in *"Texts"* - *"Text"*, CODESYS automatically creates an entry in the *"GlobalTextList"* and displays the ID here.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	Variable of type `BOOL` Example: *"PLC_PRG.xIsTrue"*
"Frame size"	Distance of the element to the edge Example: *"From style"*

Element property 'Texts'

The properties contains character strings for labeling the element.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Example: `Axis 1`.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Parameters of Axis 1`.

See also

- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Text properties'

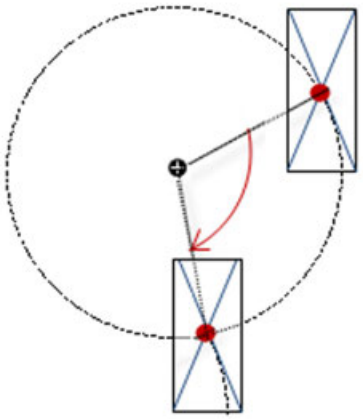
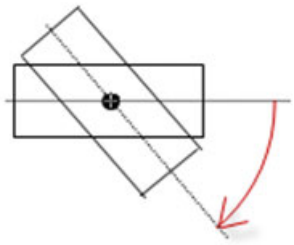
The properties contain fixed values for the text properties.

"Usage of"	• *"Default style values"*: The values of the visualization style are used. • *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long • *"Default"*: The long text is truncated. • *"Line break"*: The text is split into parts. • *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.

"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Radio Buttons'

Symbol:

Category: *"Common Controls"*

The element provides a series of radios buttons with an unlimited number of options.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Morning Shift`
“Type of element”	*“Radio Buttons”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Variable”	Variable (integer data type) that gives the index of the radio button that the visualization user has activated Example: `PLC_PRG.iNrOfActivatedRadioButton`
“Number of columns”	Definition of the number of list boxes displayed in a row Example: `2`
“Radio button order”	*“Left to right”*: The radio buttons are aligned by rows until the number of columns is reached. *“Top to bottom”*: The radio buttons are aligned row by columns until the number of columns is reached.
“Frame size”	Defines the distance from the list boxes to the edge (in pixels).
“Row height”	Height of the row (in pixels) Modifying the height of the row also changes the size of the list box.

Element property 'Text properties'

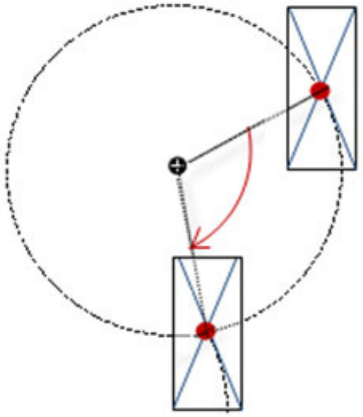
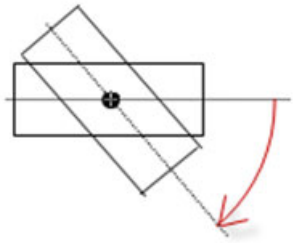
The properties contain fixed values for the text properties.

"Usage of"	● *"Default style values"*: The values of the visualization style are used. ● *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Radio button settings'

“Radio button” ● *“Areas”* – *“[<n>]”*	*“Create new”*: Clicking this button creates a new selection button in the editor and lists an additional area in the properties editor. For each radio button, an area is visible that records the settings. ● [<n>] – *“[<n>]”*: This number indicates the area. Clicking *“Delete”* will delete the associated radio button with its settings *“Text”*, *“Tooltip”*, and *“Line spacing (in pixels)”*.
Areas: [<n>]	
“Text”	The button name is specified here. Default value: *“Radio_button”*
“Tooltip”	Text is specified here that is displayed in a tooltip.
“Line spacing (in pixels)”	The distance (in pixels) to the upper button can be specified here.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Alarm Table'

Symbol:

Category: *“Alarm Manager”*

The element displays alarms in a list. In the element properties, you specify which information is shown. You define the appearance of the element and the variables that control the element behavior.

In online mode, you can sort an alarm table by a specific column – even in the classic view. Click into the column header. A small triangle indicates the current sort order (ascending, descending). Clicking the symbol reverses the order.

Sorting inside the column depends on the type of the contained information. The "Priority" column is sorted numerically, and the "Message" and "Class" columns alphabetically. The "Value" and "Latch" columns may contain different value types. In this case, sorting is first by type (blank, Boolean, numeric value, character string) and then either numerically or alphabetically depending on the type.

If an alarm history has been created, then you can programmatically delete it at runtime. The recording starts again from the time of deletion. See the help page for "Visualizing Alarm Management".

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Alarm Table"*
"Data source"	Selection of the device and the application where the data to be visualized and the alarms are generated ● Remote data source which accesses a remote device, accesses a remote application, and then transfers the data to the alarm configuration Example: `DataSource_A` Below the (now visible) *"Application"* property, the remote application is displayed as configured in the data source. Example: `App_A` Note: If the data source is accessed symbolically by means of a symbol file (CODESYS symbolic), then the required symbol file and the corresponding project have to be saved in the same folder. ● Local application below which the alarm configuration is located Example: *"<local application>"*

See also

- *Object 'Data Source'*

Element property 'Alarm configuration'

"Alarm groups"	Opens the *"Select Alarm Group"* dialog where you define the alarm groups that you want to display.
"Priority from"	Least priority for alarm display. (0 to 255).
"Priority to"	Greatest priority for alarm display. (0 to 255).
"Alarm classes"	Opens the *"Select Class Group"* dialog where you define the alarm classes that you want to display.
"Filter criterion"	For the *"Alarm Banner"* element only ● *"Most important"*: The alarm with the highest priority (lowest value) is displayed. ● *"Newest"*: The most recent alarm is displayed.

"Filter by latch 1"	The generated alarms (previous and current) can be filtered by the contents of *"Latch variable 1"*, which is specified in the configuration of the alarm group. In *"Filter type"*, you define whether or not the filtering is performed by a string value or a numerical value. ● *"Filter variable"*: Indicates what the alarms are filtered by. Possible entries: Application variable of data type `STRING` or `WSTRING`, or a literal value directly. Examples:`PLC_PRG.strFilterVariable`, `'STRING'`. ● *"Filter type"*: Integer value that determines by which criteria the latch variable value is used for filtering. Possible entries: Numerical variable from the application (example: `PLC_PRG.diFilterType`, or a value directly (example: `2`). Possible values: – `0`: No filtering – `1`: Filter by alarms whose latch variable 1 contains the string specified in *"Filter variable"*. Example: The filter variable contains `'Error 1'` which is the latch variable 1 of different alarms of type `STRING` and has the value `'Error 1'` ->. Only these alarms are displayed. – `2`: Filter by alarms whose latch variable 1 contains the typed literal specified in *"Filter variable"* according to IEC 61131-3. Examples: `T#1h2s`, `DINT#15`, `REAL#1.5`, `FALSE` – `3`: Filter by alarms whose latch variable 1 contains the LINT literal value specified in *"Filter variable"*. Therefore, the value of the latch variables has to be in the range of 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. – All other values: The behavior is not defined and can change in the future.
"Filter by time range"	The generated alarms (remote, historical, local) can be displayed for a specified time range. You use the *"Filter type"* to define whether filtering by time range is enabled or disabled. ● *"Filter variable, from"*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeFrom`) for the start time that the alarms are displayed. ● *"Filter variable, to"*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeTo`) for the end time that the alarms are displayed. ● *"Filter type"*: Variable of integer data type that determines whether *"Filter by time range"* is enabled or disabled. Possible values: – `1`: Filtering is enabled – `0`: Filtering is disabled

See also

- *Chapter 1.4.5.19.3.17 "Dialog 'Selected Alarm Group'" on page 1769*
- *Chapter 1.4.5.19.3.16 "Dialog 'Selected Alarm Class'" on page 1768*

Element property 'General table configuration'

"Show row header"	☑: Display of the row number at the beginning of the row.
"Show column header"	☑: Display of the column heading as defined in *"Column heading"*.
"Row height"	Height of the table rows (in pixels).
"Row header width"	Width of the line header (in pixels).

"Scrollbar size"	Width of the scrollbar when it runs vertically. Width of the scrollbar when it runs horizontally. Specified in pixels
"Automatic line break for alarm message"	☑: The message text is truncated at the end of the line. ☐: The message text is truncated at the end of the column, if the text is too long.

Element property 'Columns: Column [<n>]'

By default, columns [0] and [1] are configured: *"Time stamp"* and *"Message text"*. You can create more columns by clicking the *"Create new"*, and remove columns by clicking *"Delete"*.

Animations (dynamic text, font variables), text, and tooltip are not supported.

"Column header"	The standard header is set and changed here by specifying a new text.
"Use text alignment in title"	☑: The text in the column header is aligned according to the current definition in *"Text alignment"*. ☐: The text in the column header is centered.
"Width"	Width of the column (in pixels).
"Data type"	Notice about time stamps: For use in a TargetVisu or WebVisu, you can control the date and time format with the help of the global string variables from the library `Alarmmanager.library`: `AlarmGlobals.g_sDateFormat` (example: `AlarmGlobals.g_sDateFormat := 'MM.yyyy'`) and `AlarmGlobals.g_sTimeFormat` (example: `AlarmGlobals.g_sTimeFormat := 'HH:mm'`). Define the information to be displayed in the column. ● *"Symbol"* ● *"Time stamp"*: Date and time of the last status change of the alarm. ● *"Time stamp active"*: Date and time of the last activation of the alarm. ● *"Time stamp inactive"*: Date and time of the last deactivation of the alarm. ● *"Time stamp acknowledge"*: Date and time of the last acknowledgment. ● *"Value"*: Current value of the printout ● *"Message text"*: Output of the message text ● *"Priority"*: Alarm priority ● *"Class"*: Alarm class ● *"State"*: Alarm state ● *"Latch Variable <n>"*: Value of the selected latch variables
"Text alignment"	Alignment of the text in this column ● *"Left"* ● *"Centered"* ● *"Right"*
"Color settings"	● *"Activate color settings"*: Boolean variable for activating and deactivating the color settings defined here. Example: `PLC_PRG.bColorSettings` ● *"Cell fill color"*: – *"Color variable"*: Variable for the cell fill color, example: `dwCellColor` (hexadecimal color definition: 16#TTRRGGBB) – *"Use color also for column header"*: ☑: The color defined via *"Color variable"* is used in the column header as well. ● *"Text color"*: – *"Color variable"*: Variable for the definition of the text color in the column, example: `dwTextColor` (hexadecimal color definition: 16#TTRRGGBB) – *"Use color also for column header"*: ☐: The color defined via *"Color variable"* is used in the column header as well.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Selection'

"Selection color"	Fill color of the selected row
"Selection font color"	Font color of the selected row
"Frame around selected cells"	☑: A frame is drawn around the selected cells at runtime.
"Variable for selected alarm group"	Name of the affected alarm group; type: `STRING`, `WSTRING`
"Variable for selected alarm ID"	Alarm ID of the affected alarm group; type: `STRING`, `WSTRING`
"Variable for selected line"	Index of the selected alarm line (0-based). The index can be read and written; integer data type
"Variable for valid selection"	TRUE: An alarm line is selected. FALSE: The selection is invalid. For example, for an empty alarm table or when an alarm is not selected yet.
"Variable for selected alarm information"	Information about the selected alarm. Type `AlarmSelectionInfo` For easy usage, the function block `AlarmSelectionInfoDefault` is provided. This FB fills the structure with the memory for 10 messages and 10 latch variables. Example: `myAlarmSelectionInfoDefault.AlarmSelectionInfo` The following information is available: • `sAlarmgroup` • `uialarmID` • `timeStampActive` • `timeStampInactive` • `timeStampAcknowledge` • `timeStampLast` • `paLatchVariables` • `iLatchVariablesCount` • `papwsAlarmMessages` • `dwAlarmMessageTextBufferSize` • `iAlarmMessagesCount` • `iSelectionChangedCounter`

Element property 'Control variables'

Boolean variables are defined here for executing specific actions in the table can be executed at runtime.

“Acknowledge selected”	Variable (`BOOL`) Example: `PLC_PRG.bAckSelectedAlarms` If the assigned variable is `TRUE`, then the selected alarm is acknowledged.
“Acknowledge all visible”	Variable (`BOOL`) Example: `PLC_PRG.bAckVisibleAlarms` If the assigned variable is `TRUE`, then all alarms are acknowledged that are visible in the alarm table.
“History”	Variable (`BOOL`) Example: `PLC_PRG.bShowHistory` If the assigned variable is `TRUE`, then the history alarms are displayed in addition to the active alarms. In the classic view, the same sort options apply as in normal mode. Note: Acknowledgment is not possible in this view.
“Freeze scroll position”	Variable (`BOOL`) Example: `PLC_PRG.bFreezeScrollPosition` If the assigned variable is `TRUE`, then the scroll position set in the *“History”* view is retained, even if a new alarm is active. If not, then the scroll position jumps to the first table row (the newest alarm).
“Count alarms”	Variable (integer data type) Example: `PLC_PRG.iNumberOfAlarms.` Number of alarms that are currently displayed in the alarm table. Defined by the alarm table.
“Count visible rows”	Variable (integer data type) Example: `PLC_PRG.iNumberVisibleLines` Number of alarms that can be displayed on one page of the alarm table. Defined by the alarm table.
“Current scroll index”	Variable (integer data type) Example: `PLC_PRG.iScrollIndex` The index of the first visible row if the alarm table (0-based). The variable can be read and written.
“Current column sorting”	Variable (integer data type) Example: `PLC_PRG.iColSort` The variable contains a value of the enumeration "VisuElemsAlarm.VisuEnumAlarmDataType". This value determines the column that sorts the alarm table.
“Variable for sorting direction”	Variable (BOOL) Example: `PLC_PRG.xSortAscending` The variable determines the sort order for the entries in the alarm table (`TRUE`: ascending; `FALSE`: descending).

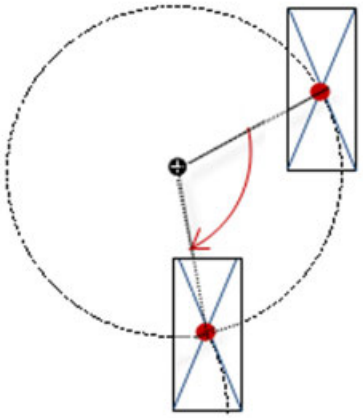
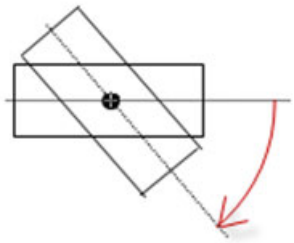
You can also use the “Insert Elements for Acknowledging Alarms” command to define buttons with predefined control variables.

See also

- *Chapter 1.4.5.19.2.23 “Command 'Add Elements for Alarm Acknowledgement'” on page 1744*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Alarm Banner'

Symbol:

Category: *"Alarm Manager"*

The element is a simplified version of the alarm table. It visualizes a single alarm only. In the element properties, you specify which information is shown. You define the appearance of the element and the variables that control the element behavior.

The alarm banner displays active alarms only. If the alarm is acknowledged, then it disappears from the alarm banner.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Alarm Banner"*
"Data source"	If you intend to use a remote alarm configuration, then you have to specify the name of the remote application here. If you do not specify anything, the alarm configuration will be located locally.

Element property 'Alarm configuration'

"Alarm groups"	Opens the *"Select Alarm Group"* dialog where you define the alarm groups that you want to display.
"Priority from"	Least priority for alarm display. (0 to 255).
"Priority to"	Greatest priority for alarm display. (0 to 255).
"Alarm classes"	Opens the *"Select Class Group"* dialog where you define the alarm classes that you want to display.
"Filter criterion"	For the *"Alarm Banner"* element only ● *"Most important"*: The alarm with the highest priority (lowest value) is displayed. ● *"Newest"*: The most recent alarm is displayed.

"Filter by latch 1"	The generated alarms (previous and current) can be filtered by the contents of *"Latch variable 1"*, which is specified in the configuration of the alarm group. In *"Filter type"*, you define whether or not the filtering is performed by a string value or a numerical value. ● *"Filter variable"*: Indicates what the alarms are filtered by. Possible entries: Application variable of data type `STRING` or `WSTRING`, or a literal value directly. Examples:`PLC_PRG.strFilterVariable`, `'STRING'`. ● *"Filter type"*: Integer value that determines by which criteria the latch variable value is used for filtering. Possible entries: Numerical variable from the application (example: `PLC_PRG.diFilterType`, or a value directly (example: `2`). Possible values: – `0`: No filtering – `1`: Filter by alarms whose latch variable 1 contains the string specified in *"Filter variable"*. Example: The filter variable contains `'Error 1'` which is the latch variable 1 of different alarms of type `STRING` and has the value `'Error 1'` ->. Only these alarms are displayed. – `2`: Filter by alarms whose latch variable 1 contains the typed literal specified in *"Filter variable"* according to IEC 61131-3. Examples: `T#1h2s`, `DINT#15`, `REAL#1.5`, `FALSE` – `3`: Filter by alarms whose latch variable 1 contains the LINT literal value specified in *"Filter variable"*. Therefore, the value of the latch variables has to be in the range of 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. – All other values: The behavior is not defined and can change in the future.
"Filter by time range"	The generated alarms (remote, historical, local) can be displayed for a specified time range. You use the *"Filter type"* to define whether filtering by time range is enabled or disabled. ● *"Filter variable, from"*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeFrom`) for the start time that the alarms are displayed. ● *"Filter variable, to"*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeTo`) for the end time that the alarms are displayed. ● *"Filter type"*: Variable of integer data type that determines whether *"Filter by time range"* is enabled or disabled. Possible values: – `1`: Filtering is enabled – `0`: Filtering is disabled

See also

- *Chapter 1.4.5.19.3.17 "Dialog 'Selected Alarm Group'" on page 1769*
- *Chapter 1.4.5.19.3.16 "Dialog 'Selected Alarm Class'" on page 1768*

Element property 'Columns: Column [<n>]'

By default, columns [0] and [1] are preconfigured: *"Time stamp"* and *"Message text"*. You create more columns by clicking *"Create new"*. You remove columns by clicking *"Delete"*.

Animations (dynamic text, font variables), texts, and tooltips are not supported.

"Width"	Width of the column (in pixels)
"Type of data"	About time stamps: When used in a TargetVisu or WebVisu, you can control the date and time format by means of the global string variables from the library `Alarmmanager.library`: `AlarmGlobals.g_sDateFormat` (example: `AlarmGlobals.g_sDateFormat := 'MM.yyyy'`) and `AlarmGlobals.g_sTimeFormat` (example: `AlarmGlobals.g_sTimeFormat := 'HH:mm'`). Here you define the information to be displayed in the column. ● *"Bitmap"* ● *"Time stamp"*: Date and time of the last status change of the alarm ● *"Time stamp active"*: Date and time of the last activation of the alarm ● *"Time stamp inactive"*: Date and time of the last deactivation of the alarm ● *"Time stamp acknowledge"*: Date and time of the last acknowledgement ● *"Value"*: Actual value of the expression ● *"Message"*: Output of the message text ● *"Priority"*: Alarm priority ● *"Class"*: Alarm class ● *"State"*: Alarm state ● *"Latch Variable <n>"*: Value of the selected latch variables
"Text alignment"	Alignment of the contents in the column ● *"Left"* ● *"Centered"* ● *"Right"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

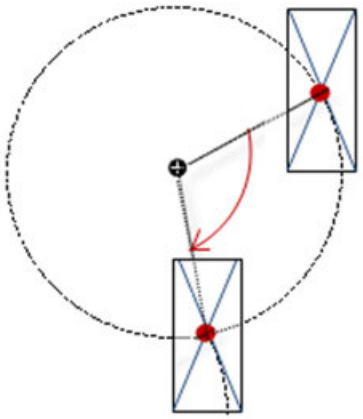
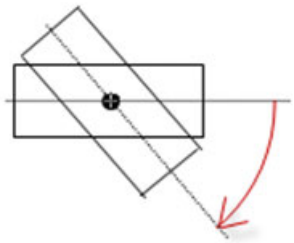
"Acknowledge variable"	A rising edge of this variable acknowledges the alarm.

Handling of multiple active alarms

"Automatic switch"	☑: The display in the alarm banner is switched automatically according to the time to the next alarm as configured in *"Every N second"*.
"Every N second"	Time period until the next switching. Available only if *"Automatic switch"* is selected.
"Next alarm"	Variable for switching to the next alarm. Available only if *"Automatic switch"* is not selected.
"Previous alarm"	Variable for switching to the previous alarm. Available only if *"Automatic switch"* is not selected.
"Multiple alarms active"	Variable that has the value `TRUE` if multiple alarms are active.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Bar Display'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element properties

“Element name”	Example: `GenElemInst_2`
“Type of element”	*“Bar Display”*
“Value”	Variable (numeric data type) The value of the variable is displayed as a bar length.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊡) to other positions in the editor.

See also

- ↳ *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Background color"	Drop-down list with background colors Note: This property depends on the style. For example, there are no heterochromatic background images for *"FlatStyle"* and *"Whitestyle"*.
"Own image"	● *"image"*: Image ID of the background image. You select the background image from an image pool by clicking the ... symbol. Info: If you specify the *"<default>"* value or select the image from the *"Default"* category in the input assistant, then the original element background image is used. ● *"Transparent color"*: Color of pixels that are displayed as transparent. Selection from drop-down list or input assistant.
"Optimized drawing"	☑: The background image is drawn one time. If there is a change in the foreground, then only the affected part of the image is redrawn. ☐: The background image is redrawn in cycles. Note: Deactivating this option is sensible only in certain exceptional cases.

Element property 'Bar'

"Diagram type"	Position of the scale ● *"Scale besides bar"* ● *"Scale in bar"* ● *"Bar in scale"* ● *"No scale"*
"Orientation"	Orientation depending on the ratio of width to height of the Bar Display: ● *"Horizontal"* ● *"Vertical"*
"Running direction"	Direction the values are increased. Drop-down list for *"Orientation Horizontal"*: ● *"Left to right"* ● *"Right to left"* Drop-down list for *"Orientation Vertical"*: ● *"Bottom to top"* ● *"Top to bottom"*
"Optimum size for bar"	☑: The bar width requires the majority of the element surface. Note: This property depends on the style. It is not provided for *"FlatStyle"* or *"WhiteStyle"*.

Element property 'Scale'

“Scale start”	Least value of the scale and the lower limit of the value range for the element. Example: `0` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale start.
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element. Example: `100` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale end.
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between 2 values on the rough scale. Example: `10` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the distance.
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Subscale”	Distance between 2 values on the fine scale. You can hide the fine scale by setting the value to `0`. Example: `2` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the spacing.
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

“Scale line width”	Specified in pixels. Example: `3`
“Scale color”	Color of scale lines •  : The *“Color”* dialog box opens. • ▼: A drop-down list with color names opens.
“Scale in 3D”	☑: Tick marks are displayed with slight 3D shadowing. Note: This property depends on the style. Not available for *“FlatStyle”*.
“Element frame”	☑: A frame is drawn around the element.

Element property 'Label'

“Unit”	Text that is displayed in the element. Example: Units displayed in m/s.
“Font”	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *“”* button. 
“ Scale format (C Syntax)”	Values scaled in "printf" syntax Examples: `%d, %5.2f`
“Max. text width of labels”	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Text height of labels”	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Font color”	Selection from the drop-down list or by clicking the  button.

Element property 'Positioning'

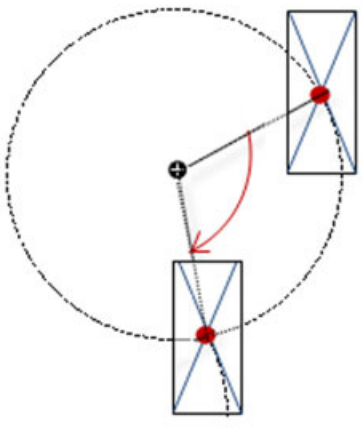
“Horizontal offset”	Distance from the scale (bar) to the horizontal element frame Specified in pixels. Used for achieving the exact position relative to the background image.
“Vertical offset”	Distance from the scale (bar) to the vertical element frame Specified in pixels. Used for achieving the exact position relative to the background image.
“ Horizontal scaling”	Horizontal division of the scale Specified in pixels. Used for achieving the exact positioning relative to the background image.
“Vertical scaling”	Vertical division of the scale Specified in pixels. Used for achieving the exact positioning relative to the background image.

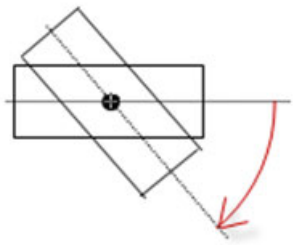
Element property 'Colors'

"Graph color"	Color of the bar
"Bar background"	☑: The background of the bar is black. ☐: The background of the bar is white.
"Frame color"	Color that the frames are drawn.
"Switch whole color"	☑: The total color of the bar is switched to the color of the color area of the current value.
"Use gradient color for bar"	☑: Bar is displayed with a gradient.
"Color range markers"	The color areas can be separated from each other inside the bar with a vertical mark. ● *"No markers"*: No display. ● *"Marker forwards"*: The color of the vertical mark corresponds to the color of the previous color area. ● *"Marker backwards"*: The color of the vertical mark corresponds to the color of the next color area.
"Color areas"	
"Create new"	A new color area is added.
" Delete"	The color area is removed from the list.
"Begin of area"	Start value of the color area
"End of area"	End value of the color area
"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Meter 90°'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable. The needle is positioned according to the value of the assigned variable. A meter is used to represent a tachometer, for example.

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Meter 90°”*
“Value”	Variable (numeric data type) The variable value determines the pointer direction of the element.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • : Drop-down list with color names
"Angle range"	Drop-down list for the alignment of the element
"Additional arrow"	: An additional arrow is shown inside the scale.

Element property 'Scale'

“Sub scale position”	● *“Outside”*: The subscale is displayed on the outer scale ring. (*“Frame outside”*) ● *“Inside”*: The subscale is displayed on the inner scale ring. (*“Frame inside”*)
“Scale type”	Type of scale ● *“Lines”* ● *“Dots”* ● *“Squares”*
“Scale start”	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *“Variable”* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

"Variable"	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

"Scale line width"	Specified in pixels Example: `3`
"Scale color"	Color of scale lines ● ...: The *"Color"* dialog opens. ● ▼: A list box with style colors opens.
"Scale in 3D"	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *"FlatStyle"*.
"Show scale"	☑: The scale is displayed.
"Frame inside"	☑: A frame is drawn at the inner end of the scale.
"Frame outside"	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

"Label"	Selection list ● *"Outside"*: Scale values are placed outside of the scale. ● *"Inside"*: Scale values are placed inside of the scale.
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *""* button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

"Usage of"	• *"Preset style values"*: Values from the current style • *"User-defined settings"*: The subnode *"Positioning"* appears.
"Positioning" Requirement: *"User-defined settings"* is selected as *"Usage of"*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
"Needle movement"	Length of the needle (in pixels)
"Scale movement"	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *"Background"*.
"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` ◆: The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

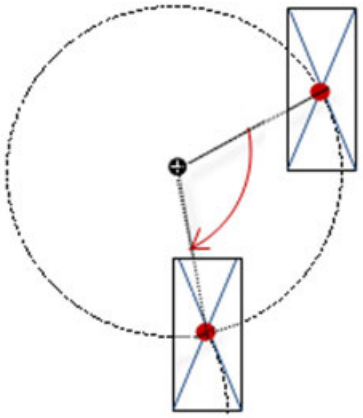
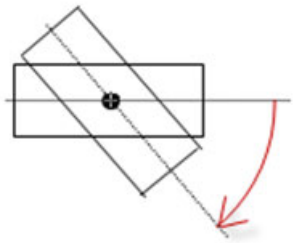
"End of area"	End value of the color area Example: `120` ◆: The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254“”*

Visualization Element 'Meter 180°'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable. The needle is positioned according to the value of the assigned variable on a scale. A meter is used to represent a tachometer, for example.

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Meter 180°”*
“Value”	Variable (numeric data type) The variable value determines the pointer direction of the element.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.

"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • : Drop-down list with color names
"Angle range"	Drop-down list for the alignment of the element
"Additional arrow"	: An additional arrow is shown inside the scale.

Element property 'Scale'

"Sub scale position"	• *"Outside"*: The subscale is displayed on the outer scale ring. (*"Frame outside"*) • *"Inside"*: The subscale is displayed on the inner scale ring. (*"Frame inside"*)
"Scale type"	Type of scale • *"Lines"* • *"Dots"* • *"Squares"*
"Scale start"	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *"Variable"* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

“Scale line width”	Specified in pixels Example: `3`

“Scale color”	Color of scale lines ● ...: The *“Color”* dialog opens. ● ▼: A list box with style colors opens.
“Scale in 3D”	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *“FlatStyle”*.
“Show scale”	☑: The scale is displayed.
“Frame inside”	☑: A frame is drawn at the inner end of the scale.
“Frame outside”	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

“Label”	Selection list ● *“Outside”*: Scale values are placed outside of the scale. ● *“Inside”*: Scale values are placed inside of the scale.
“Unit”	Text that is displayed in the element. Example: Units displayed in m/s.
“Font”	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *“”* button. ...
“ Scale format (C Syntax)”	Values scaled in "printf" syntax Examples: `%d, %5.2f`
“Max. text width of labels”	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Text height of labels”	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Font color”	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

“Usage of”	● *“Preset style values”*: Values from the current style ● *“User-defined settings”*: The subnode *“Positioning”* appears.
“Positioning” Requirement: *“User-defined settings”* is selected as *“Usage of”*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
“Needle movement”	Length of the needle (in pixels)
“Scale movement”	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *“Background”*.

"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

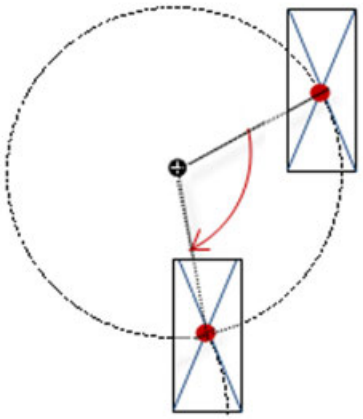
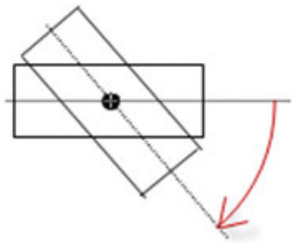
"End of area"	End value of the color area Example: `120` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Meter'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable. The needle is positioned according to the value of the assigned variable. A meter is used to represent a tachometer, for example.

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Meter”*
“Value”	Variable (numeric data type). The variable value determines the pointer direction of the element.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.

"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • ▼: Drop-down list with color names
"Arrow start"	Angle (in degrees) between the scale start and the horizontal axis
"Arrow end"	Angle (in degrees) between the right edge of the pointer instrument and the horizontal axis
"Additional arrow"	☑: An additional arrow is shown inside the scale.

Element property 'Scale'

"Sub scale position"	• *"Outside"*: The subscale is displayed on the outer scale ring. (*"Frame outside"*) • *"Inside"*: The subscale is displayed on the inner scale ring. (*"Frame inside"*)
"Scale type"	Type of scale • *"Lines"* • *"Dots"* • *"Squares"*
"Scale start"	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *"Variable"* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

“Scale line width”	Specified in pixels Example: `3`

"Scale color"	Color of scale lines ● ...: The *"Color"* dialog opens. ● ▼: A list box with style colors opens.
"Scale in 3D"	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *"FlatStyle"*.
"Show scale"	☑: The scale is displayed.
"Frame inside"	☑: A frame is drawn at the inner end of the scale.
"Frame outside"	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

"Label"	Selection list ● *"Outside"*: Scale values are placed outside of the scale. ● *"Inside"*: Scale values are placed inside of the scale.
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the "" button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

"Usage of"	● *"Preset style values"*: Values from the current style ● *"User-defined settings"*: The subnode *"Positioning"* appears.
"Positioning" Requirement: *"User-defined settings"* is selected as *"Usage of"*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
"Needle movement"	Length of the needle (in pixels)
"Scale movement"	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *"Background"*.

"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

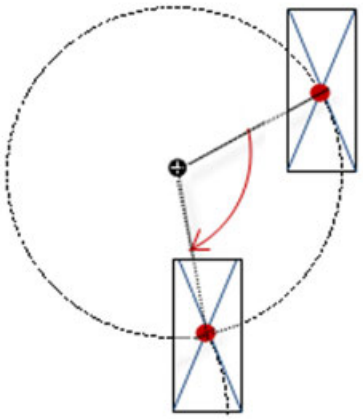
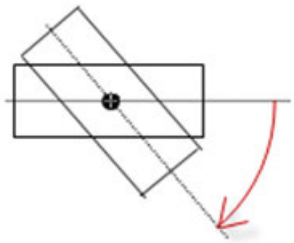
"End of area"	End value of the color area Example: `120` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Potentiometer'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable as a setting on the potentiometer. A visualization user can modify the value by dragging the pointer to another position.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Potentiometer”*
“Variable”	Variable (numeric data type). Contains the position of the pointer for the potentiometer. A visualization user can modify the value by dragging the pointer to another position.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • ▼: Drop-down list with color names
"Arrow start"	Angle (in degrees) between the left edge of the element and the horizontal axis
"Arrow end"	Angle (in degrees) between the right edge of the element and the horizontal axis

Element property 'Scale'

“Sub scale position”	● *“Outside”*: The subscale is displayed on the outer scale ring. (*“Frame outside”*) ● *“Inside”*: The subscale is displayed on the inner scale ring. (*“Frame inside”*)
“Scale type”	Type of scale ● *“Lines”* ● *“Dots”* ● *“Squares”*
“Scale start”	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *“Variable”* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

"Variable"	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

"Scale line width"	Specified in pixels Example: `3`
"Scale color"	Color of scale lines ● ...: The *"Color"* dialog opens. ● ▼: A list box with style colors opens.
"Scale in 3D"	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *"FlatStyle"*.
"Show scale"	☑: The scale is displayed.
"Frame inside"	☑: A frame is drawn at the inner end of the scale.
"Frame outside"	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

"Label"	Selection list ● *"Outside"*: Scale values are placed outside of the scale. ● *"Inside"*: Scale values are placed inside of the scale.
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *""* button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

"Usage of"	• *"Preset style values"*: Values from the current style • *"User-defined settings"*: The subnode *"Positioning"* appears.
"Positioning" Requirement: *"User-defined settings"* is selected as *"Usage of"*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
"Needle movement"	Length of the needle (in pixels)
"Scale movement"	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *"Background"*.
"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` ◆: The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

"End of area"	End value of the color area Example: `120` ◆: The property *"Variable"* is shown below.

“Variable”	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

“Color”	Color that is used for displaying the area.

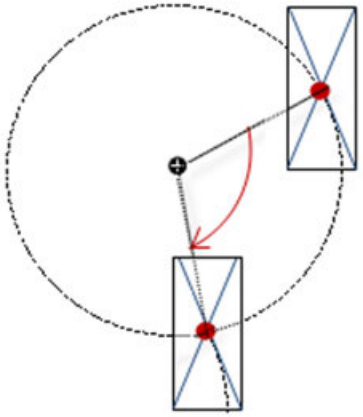
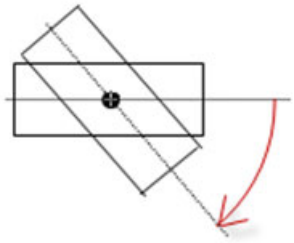
Element property 'State variables' The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Absolute movement' The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Histogram'

Symbol:

Category: *“Measurement Controls”*

The element displays the data of a one-dimensional array as a histogram. You can define specific colors for certain value ranges.

See also

- *Chapter 1.4.5.21.4 “Displaying Array Data in a Histogram” on page 2138*

Element properties

“Element name”	Example: `GenElemInst_35`
“Type of element”	*“Histogram”*
“Data array”	One-dimensional array with data displayed in this histogram. Example: `PLC_PRG.arr1`

Element property 'Subrange of array'

“Use subrange”	☑: Only part of the array is displayed in the histogram.
“Start index”	First array index with a displayed value. Requirement: *“Use subrange”* is activated.
“End index”	Last array index with a displayed value. Requirement: *“Use subrange”* is activated.

“Display type”	● *“Bars”*: Data is displayed as bars. ● *“Lines”*: Data is displayed as lines. ● *“Curve”*: Interpolation of data into a curve.
“Line width”	Specified in pixels Requirement: *“Curve”* is selected as the *“Display type”*.
“Show horizontal lines”	☑: Horizontal lines are drawn on the main scale. Note: Not all visualization styles have this property. This element property is not available for visualization styles that have striped backgrounds (example: *“Flat style”*).
“Relative bar width”	Integer value between `1` and `100` ● `1`: The bars are drawn as lines. ● `100`: The entire width of the histogram is filled with the bars.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Scale'

"Scale start"	Least value of the scale and the lower limit of the value range for the element. Example: `0` : The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the scale start. Example: `PLC_PRG.iScaleStart`
"Scale end"	Greatest value of the scale and the upper limit of the value range for the element. Example: `100` : The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the scale end. Example: `PLC_PRG.iScaleEnd`

"Main scale"	Distance between 2 values on the rough scale. Example: `10` : The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the distance. Example: `PLC_PRG.iMainScale`
"Subscale"	Distance between 2 values on the fine scale. You can hide the fine scale by setting the value to `0`. Example: `2` : The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the spacing. Example: `PLC_PRG.iSubScale`
"Scale color"	Color of scale lines • ...: The *"Color"* dialog box opens. • : A drop-down list with color names opens.
"Base line"	Value of the main scale where the horizontal base line of the Histogram is located. The drawing of the bar starts at the base line.

Example A valid declaration is required for the variables used as an example in the table above.

```
PROGRAM PLC_PRG
VAR
    iScaleStart : INT := 0;
    iScaleEnd : INT := 120;
    iMainScale : INT := 20;
    iSubScale : INT := 5;
END_VAR
```

Element property 'Label'

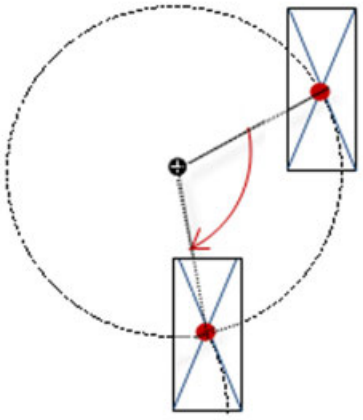
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *""* button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	Optional value that defines the maximum width of the scale label. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	Optional value that defines the maximum height of the scale label. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

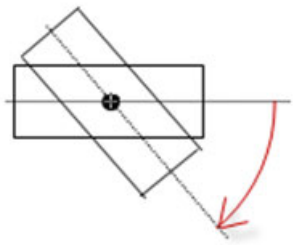
Element property 'Colors'

"Graph color"	Color of the bar in normal state. Note: The normal state is in effect when the current value of the array component does not fulfill the alarm condition.
"Alarm value"	Threshold for the alarm
"Alarm condition"	If the current value of the array component fulfills the alarm condition, then the alarm condition is set. ● *"Less"*: The current value is less than the *"Alarm value"* ● *"More"*: The current value is greater than the *"Alarm value"*
"Alarm color"	Color of the bar in alarm state.
"Use color areas"	☑: The color areas defined in this element are used.
"Color areas"	
"Create new"	A new color area is added.
" Delete"	The color area is removed from the list.
"Begin of area"	The start value on the *"Scale"* of the Histogram where the color area begins.
"End of area"	The end value on the *"Scale"* of the Histogram where the color area ends.
"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Image Switcher'

Symbol:

Category: *"Lamps/Switches/Bitmaps"*

The element displays one of three referenced images. Mouse actions change the displayed image. The images are defined in the *"Image settings"* element properties. The effects of mouse clicks are defined in the *"Element behavior"* property.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `ImageSwitcher_1`
“Type of element”	*“Image Switcher”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Variable”	Variable (`BOOL`). The value of the variable changes according to user input and it is independent of the *“Element behavior”* element property.

Image settings

“Image "on"”	Image ID from an image pool. The image can be selected using the input assistant. The image is used if the variable of the *“Variable”* property has the value `TRUE`.
“Image "off"”	Image ID from an image pool. The image can be selected using the input assistant. The image is used if the variable of the *“Variable”* property has the value `FALSE`.
“Image "clicked"”	Image ID from an image pool. The image is selected using the input assistant. In runtime mode, the visualization displays the referenced image when the element is clicked (and the mouse button is held down). Requirement: The *“Element behavior”* is *“Image toggler”*.

"Transparency"	☑: The *"Transparent color"* is selected.
"Transparent color"	The image pixels that have the transparent color are displayed as transparent. Requirement: *"Transparency"* is activated. ● ...The *"Color"* dialog box opens. ● ▼: A drop-down list with color names opens.
"Scaling type"	Defines how an image fits in the element frame. ● *"Fixed"*: The original size of the image is retained, regardless of the dimensions of the element. ● *"Isotropic"*: The entire image is shown in the element frame, either larger or smaller. As a result, the proportion of height and width are retained. ● *"Anisotropic"*: The image resizes automatically to the dimensions of the element frame, filling the entire element frame. As a result, the proportions are not retained.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

"Element behavior"	● *"Image toggler"*: Every mouse click switches the image. ● *"Image tapper"*: While a visualization user holds down the mouse button, the image of the *"Image on"* property is displayed. At the same time, the value `TRUE` is assigned to the *"Variable"* property.
"Tap FALSE"	☑: While the mouse button is pressed, the image of the *"Image"* property is displayed and the *"Variable"* property gets the value `FALSE` instead of the value `TRUE`, and back. Requirement: *"Image tapper"* is selected in the *"Element behavior"* property.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

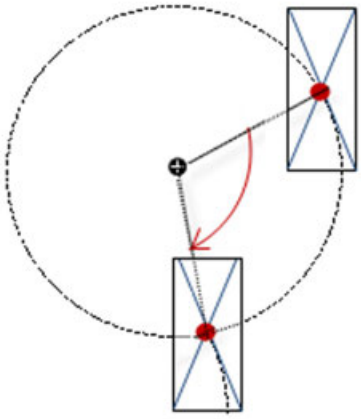
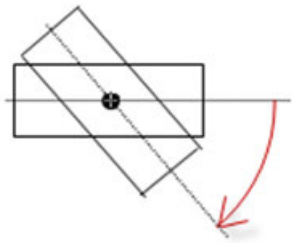
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access`.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Lamp'

Symbol:

Category: *“Lamps/Switches/Bitmaps”*

The element shows the value of a variable, and the element is displayed as illuminated or not.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Lamp_green`
"Type of element"	*"Lamp"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Variable"	Variable (`BOOL`). The variable value is displayed as a lamp that goes on (`TRUE`) or off (`FALSE`).

Image settings

"Transparency"	: The *"Transparent color"* property is selected.
"Transparent color"	Pixels in this color are displayed as transparent. Requirement: *"Transparency"* is activated. • The *"Color"* dialog box opens. • : A drop-down list with style colors opens.

"Scaling type"	Reaction of the element when the dimension of the *"Frame"* element is changed: ● *"Isotropic"*: The height and width of the image are resized proportionally to the *"Frame"*. Please note: To retain the alignment of elements also within a scaled *"Frame"* element, define the *"Horizontal alignment"* or *"Vertical alignment"* explicitly with *"Centered"*. ● *"Anisotropic"*: The image fills the entire *"Frame"* regardless of its proportions.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

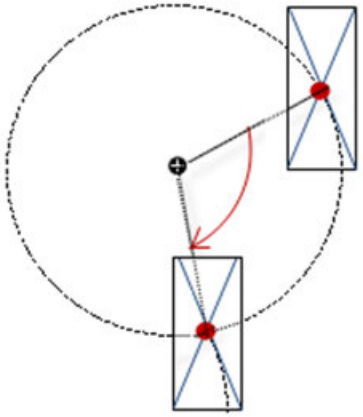
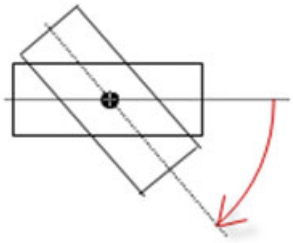
Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access`.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Background'

“Image”	Drop-down list with background colors Depends on the visualization style

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

● *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Dip Switch', 'Power Switch', 'Push Switch', 'Push Switch LED', 'Rocker Switch'

Symbols:

Category: *"Lamps/Switches/Bitmaps"*

The element assigns a value to a Boolean variable. The switch position "on" the value `TRUE` to the variable, and the switch position "off" assigns the value `FALSE`. Use the mouse to change the switch position.

Element properties

"Element name"	Example: `Operating_Switch` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	Depending on the element: *"Dip Switch"*, *"Power Switch"*, *"Push Switch"*, *"Push Switch LED"*, or *"Rocker Switch"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Variable"	Variable (`BOOL`) The value of the variables `TRUE` and `FALSE` indicates the switch position on/off.

Image settings

"Transparency"	☑: The *"Transparent color"* property is selected.
"Transparent color"	Pixels in this color are displayed as transparent. Requirement: *"Transparency"* is activated. ● ...The *"Color"* dialog box opens. ● ▼: A drop-down list with style colors opens.
"Scaling type"	Reaction of the element when the dimension of the *"Frame"* element is changed: ● *"Isotropic"*: The height and width of the image are resized proportionally to the *"Frame"*. Please note: To retain the alignment of elements also within a scaled *"Frame"* element, define the *"Horizontal alignment"* or *"Vertical alignment"* explicitly with *"Centered"*. ● *"Anisotropic"*: The image fills the entire *"Frame"* regardless of its proportions.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

"Element behavior"	● *"Image toggler"*: Every mouse click changes the switch and the *" Variable"* value. ● *"Image tapper"*: The switch is "on" and the *"Variable"* value is `TRUE` while the mouse button is pressed.
"Tap FALSE"	☑: The value `TRUE` is assigned to the *"Variable"* property instead of the value `FALSE`, and back. Requirement: *"Image tapper"* is selected in the *"Element behavior"* property.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

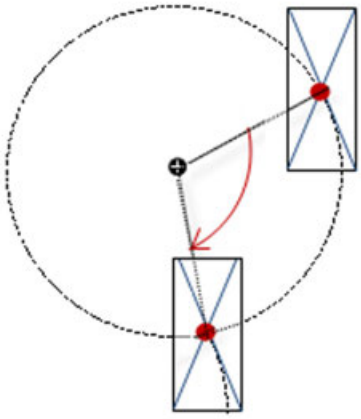
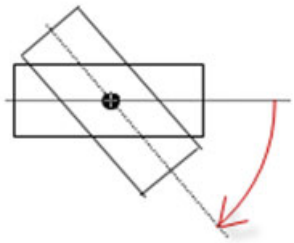
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access.`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Background'

"Image"	Drop-down list with background colors Depends on the visualization style

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Rotary Switch'

Symbol:

Category: *“Lamps/Switches/Bitmaps”*

The element assigns a value to a Boolean variable. The switch position "on" the value `TRUE` to the variable, and the switch position "off" assigns the value `FALSE`. Use the mouse to change the switch position.

Element properties

“Element name”	Example: `Operating_Switch` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Rotary Switch”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Variable”	Variable (`BOOL`). The value of the variables `TRUE` and `FALSE` indicates the switch position on/off.

Image settings

"Transparency"	☑: The *"Transparent color"* property is selected.
"Transparent color"	Pixels in this color are displayed as transparent. Requirement: *"Transparency"* is activated. ● ...The *"Color"* dialog box opens. ● ▼: A drop-down list with style colors opens.
"Scaling type"	Reaction of the element when the dimension of the *"Frame"* element is changed: ● *"Isotropic"*: The height and width of the image are resized proportionally to the *"Frame"*. Please note: To retain the alignment of elements also within a scaled *"Frame"* element, define the *"Horizontal alignment"* or *"Vertical alignment"* explicitly with *"Centered"*. ● *"Anisotropic"*: The image fills the entire *"Frame"* regardless of its proportions.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

"Element behavior"	● *"Image toggler"*: Every mouse click changes the switch and the " *Variable"* value. ● *"Image tapper"*: The switch is "on" and the *"Variable"* value is `TRUE` while the mouse button is pressed.
"Orientation"	● *"At top"*: The rotary switch turns from the top right to the top left. ● *"At side"*: The rotary switch turns from the top right to the bottom right.
"Color change"	☑: The element changes in color when " *Variable"* is `TRUE`.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

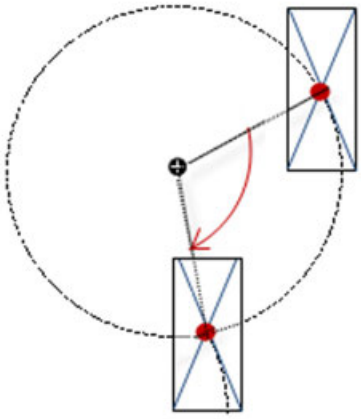
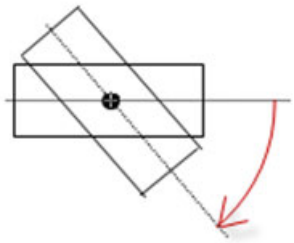
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access`.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Background'

“Image”	Drop-down list with background colors Depends on the visualization style

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Trace'

Symbol:

Category: *"Special Controls"*

The element displays the graphical curve of variable values. In addition, variables can be configured to control the view.

See also

- *Chapter 1.4.5.10 "Displaying data curve with trace" on page 1306*
- *"Dialog box 'Trace Configuration'" on page 1734*

Element properties

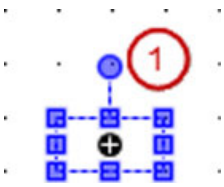
"Element name"	Example: `Velocity`
"Data Source"	Location where the trace data is buffered. : ● *"<local application>"*: The trace record is listed below the local application. The visualization that contains the trace is located below this application. When the application is downloaded, the trace configuration is downloaded to the local device. During execution, the data is stored locally in the trace buffer. ● *" <data source name>"*: Data source that identifies the remote device where the trace record is created. When the local application is downloaded with the visualization, the trace configuration is downloaded to the **remote** device. During execution, the trace buffer is filled, and the trace data is transferred and then displayed in the local visualization as HMI. Example: `DataSoure_PLC_A` Note: The trace buffer is filled only if the remote application is being executed. The data recording is started when the local visualization is started.
"Application"	Application where data was recorded. : Lists all applications that are present below the data source. Requirement: A remote data source (not *"<local application>"*) is referenced in the *"Data source"* property.
"Type of element"	*"Trace"*
"Trace"	*" <name of trace configuration>"*: Opens the *"Trace Configuration"* dialog where you can modify the trace configuration.

See also

- *"Dialog box 'Trace Configuration'" on page 1734*
- *Data Source Manager*

Element property 'Position'

The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Y"	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Width"	Specified in pixels Example: `150`
"Height"	Specified in pixels Example: `30`
	Tip: You can change the values in *"X"*, *"Y"*, *"Width"*, and *"Height"* by dragging the corresponding symbols to another position in the editor.
"Angle"	Static angle of rotation (in degrees) Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Show cursor"	☑: A cursor (vertical line) is displayed at the mouse position. The trigger and variable values where the cursor points are displayed as a tooltip.
"Overwrite existing trace on PLC"	☑: If a trace with the same name is on the PLC, then it is overwritten at download with the configuration that is defined here.
"Number format"	Number format of values in the tooltip in printf syntax (example: %d, %5.2f).

Element property 'Control variables'

The control variables are assigned automatically when you click *"Insert elements for controlling Trace"*.

“Reset Trigger”	Variable (BOOL). Standard control variable: bResetTrigger TRUE: Resets the triggering. After the action is executed, the variable is set automatically to FALSE.
“Start Trace”	Variable (BOOL). Standard control variable: bStart TRUE: Starts the Trace. After the action is executed, the variable is set automatically to FALSE.
“Stop Trace”	Variable (BOOL). Standard control variable: bStop TRUE: Stops the Trace. After the action is executed, the variable is set automatically to FALSE.
“Save Trace to a file”	
“Save Trace”	Variable (BOOL). Standard control variable: bStore TRUE: Saves the current trace configuration and the data that is stored in the development system to a file. When the action is ended, the variable is set automatically to FALSE.
“File name”	Variable (STRING) that contains the file name of the file to be saved. Standard control variable: sStoreFilename
“Load trace from file”	
“Load Trace”	Variable (BOOL). Standard control variable: bRestore TRUE: Reads the file specified below and loads its contents into the trace editor. The file contains a trace configuration and possibly also trace data. To do this, the stored trace configuration must match the application where the trace configuration is located. When the action is ended, the variable is set automatically to FALSE. Note: A trace configuration can be loaded from a file only under special circumstances. The file must have been created with exactly the same (running) application with which it will then be loaded. The consequence of changing the running application (for example by downloading again) is that a file which was previously created from the application cannot no longer be read into the application. Even external manual changes to the file can cause this. You should edit only those configuration settings that have an effect on displaying the variables. If you change variable definitions directly in the file (for example by replacing variable x with v y), then the file cannot be loaded.
“File name”	Variable (STRING) that contains the file name of the file to be read. Standard variable: sRestoreFilename

See also

- *Chapter 1.4.5.19.2.15 “Command 'Insert Elements for Controlling Trace'” on page 1737*

Element property 'Center'

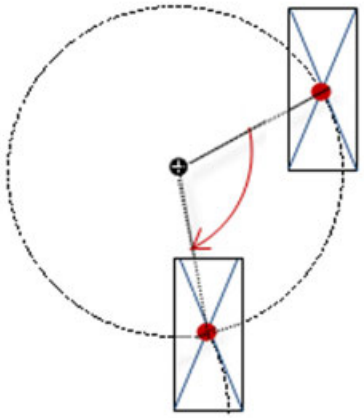
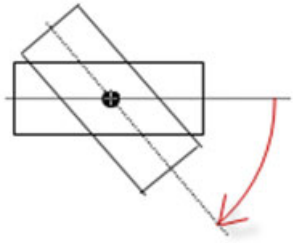
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Trend'

Symbol:

Category: *"Special Controls"*

The element displays the curve of variable values as a trend diagram. The trend diagram is suitable for representing a long-term data curve because the data is read from a trend recording and hence from a database. Moreover, you can run the *"Trend"* element together with the *"Date Range Picker"*, *"Legend"*, and *"Time Range Picker"* operating elements so that the user can navigate conveniently in the diagram.

> *You can programmatically delete the recorded trend curve at runtime. The recording starts again from the time of deletion. See the help page for "Programming a Trend Visualization".*

Element properties

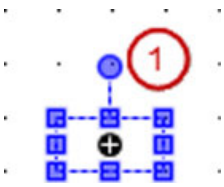
"Element name"	Example: `Velocity`
"Data source"	Data source for the connection via the device and the application to the *"Trend Recording"* object where the trend data that you want to show was saved. If the *"Trend Recording"* object is on the local device, then it is sufficient when you specify the respective application. If the trend recording is on a remote device, then you need to specify the data source connection to this device. ● *" <local application>"* The *"Trend Recording"* object is located on the local device in the local application. ● `<device name> . <application name>` Example: `Device_A.App_A` The *"Trend Recording"* object is located on the local device `Device_A` below the application `App_A`. ● `<data source name>` Example: `DataSource_B` The *"Trend Recording"* object is located on a remote device that is connected via the data source `DataSource_B`. Below the (now visible) *"Application"* property, the remote application is displayed as configured in the data source. Example: `App_B` Note: If the data source is accessed symbolically by means of a symbol file (CODESYS symbolic), then the required symbol file and the corresponding project have to be saved in the same folder.
"Type of element"	*"Trend"*
"Trend recording"	: Trend recording whose data is displayed as a diagram The trend recording is located on the device specified in the *"Data source"* property.
"Display Mode"	: Opens the *"Display Settings"* dialog.

See also

- *Chapter 1.4.5.11 "Displaying data curve with trend" on page 1309*
- *Chapter 1.4.5.19.2.12 "Command 'Configure Display Settings of Trend'" on page 1732*
- *Object 'Data Source'*

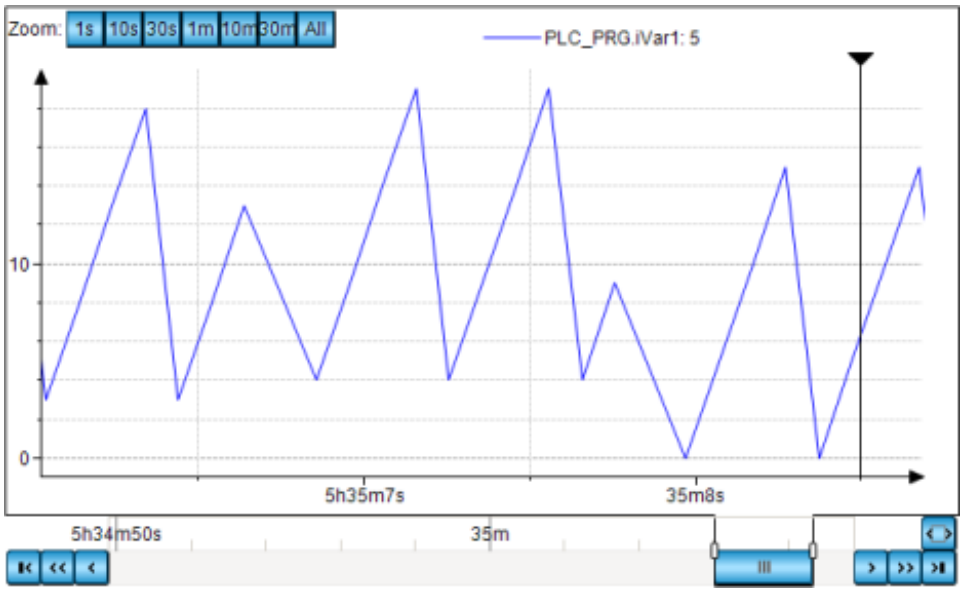
Element property 'Position'

The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Y”	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Width”	Specified in pixels Example: `150`
“Height”	Specified in pixels Example: `30`
	Tip: You can change the values in *“X”*, *“Y”*, *“Width”*, and *“Height”* by dragging the corresponding symbols to another position in the editor.
“Angle”	Static angle of rotation (in degrees) Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *“Absolute movement ➔ Internal rotation”*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

"Show cursor"	☑: A cursor (black triangle with vertical line) is shown in the trend diagram. Behavior at runtime: As soon as the graph is drawn, the user can move the cursor along the time axis in order to mark a specific time. Then the variable value belonging to the cursor position is displayed in the legend above the graph. 
"Show tooltip"	Requirement: *"Show cursor"* is activated. ☑: A tooltip opens at the cursor. Behavior at runtime: The variable value belonging to the cursor position is displayed as a tooltip.
"Show frame"	☑: The trend diagram is drawn with a frame.
"Number format"	Format specification in printf syntax, which determines how the values are displayed in the tooltip and in the legend Example: `%d` (integer variable) or `%5.2f` (floating-point number)

Element property 'Tick mark labels'

The time stored in the trend recording are in the UTC time zone. If the time is displayed in the trend of the visualization element, then the time stamps are converted to the local time zone of the operating system of the PLC.

Change the time zone in the operating system if the times in the trend diagram are not in the zone that you need.

“Time stamps”	X-value of the trend diagram ● *“Absolute time stamps”* The absolute time with date and time is displayed at each tick mark on the time axis. Example: `03/18/2016 12h30m50s` ● *“Relative time stamps”* The time period from the start of the recording (=0) is displayed at each tick mark. Example: `5m30s`
“Draw labels on two lines”	☑: The time stamps are displayed on two lines (for example, the date is displayed on the first line and the time on the second line). ☐: The time stamp is displayed on one line. Example: `2019-11-01-12:30:50.`
“ Omit irrelevant information in timestamps”	☑: The time stamps are displayed in a truncated form (without insignificant information). For example, the date is displayed at the first tick mark, and only the time is displayed at the following tick marks. The *“Internationalization (format strings)”* property is not visible and is ignored. ☐: The time stamps are displayed with all information. This takes into consideration the *“Internationalization (format strings)”* property which contains the format specification for the date and time display.
“Internationalization (format strings)”	Format specification for the date and time display of the time stamp (when it is displayed in full) Note: The property is visible only if the *“Omit irrelevant information in timestamps”* option is **not** selected.

"Date"	Format string that returns the date display according to the defined format. The operating system locale is used as the default setting.
	Defined format strings for the date:
	● Year: `yyyy`, `yy`, `y`
	● Month: `MM`, `M`
	● Day: `dd`, `d`
	● Recommended separator: `- . /`
	Example:
	`yyyy-MM-d` displays `2019-10-25`
	`yyyy-MM-dd` displays `2019-10-25`
	`dd.MM.yyyy` displays `25.10.2019`
	`dd/MM/yyyy` displays `25/10/2019`

"Time"	Format string that returns the time (or time of day) display according to the defined format. The operating system locale is used as the default setting.
	Defined format strings for the time:
	● 24-hour time definition: `HH`, `H`
	● 12-hour time definition: `hh`, `h`
	● AM/PM for 12-hour time definition: `tt`
	● Minutes: `mm`, `m`
	● Seconds: `ss`, `s`
	● Milliseconds: `ms`
	● Microseconds: `us`
	● Recommended separator: `:` or space character
	Example:
	`HH:mm:ss:ms` displays `15:30:59:123`
	`h:mm:ss tt` displays `3:30:59 PM`

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Assigned control elements'

These elements are created automatically when the control elements are added with the command *"Insert elements for controlling Trend"*.

"Date Range Picker"	Control element for changing the date and time of the displayed data sets. With ..., all elements are provided that have implemented the interface `IDateRangeSelector`. By default, instances of the *"Date Range Picker"* visualization element are available.
"Time Range Picker"	Control element for changing the time of the displayed data sets. With ..., all elements are provided that have implemented the interface `ITimeSelector`. By default, instances of the *"Time Range Picker"* visualization element are available.
"Legend"	Control element for displaying a legend for the graphs. With ..., all elements are provided that have implemented the interface `ILegendDisplayer`.

See also

- *Chapter 1.4.5.19.2.18 "Command 'Insert Elements for Controlling the Trend'" on page 1739*

Element property 'Center'

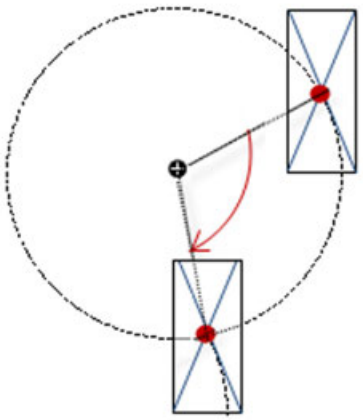
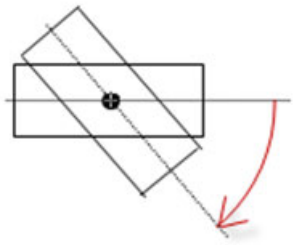
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.11 "Displaying data curve with trend" on page 1309*
- *Chapter 1.4.5.11.1 "Getting Started with Trend Visualization" on page 1309*
- *Chapter 1.4.5.11.2 "Programming a Trend Visualization" on page 1312*
- *Object 'Trend Recording'*

Visualization Element 'Legend'

Symbol:

Category: *"Special Controls"*

The element is used as a legend for another element (for example, a trend). The legend is assigned in the properties of the other element.

See also

- *Chapter 1.4.5.11 “Displaying data curve with trend” on page 1309*

Element properties

“Element name”	Example: `LegendOfTrendA` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Legend”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Orientation"	Orientation of the element. The value is configured in the assigned element. ● *"Horizontal"* ● *"Vertical"*
"Attached element instance"	Example: `Element_A`
"Show frame"	☑: The element is displayed with frames.
"Number format"	The format of the value in printf syntax (example: %d, %5.2f)

Element Property 'Layout'

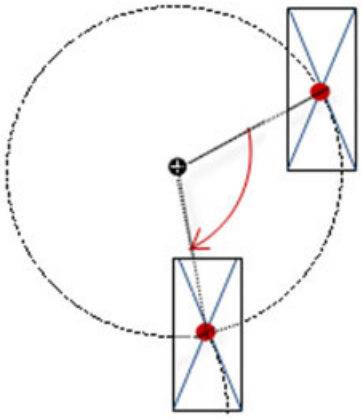
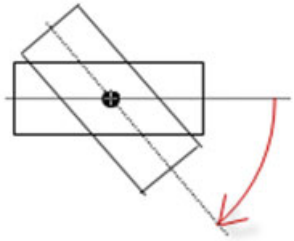
Defines how many variables can be displayed at a maximum and is calculated from the row and column number.	
"Max. number of rows"	Example: `3`
"Max. number of columns"	Example: `2`

Element Property 'Text properties' The property affects the text configured in the associated element.

"Text format"	*"Default"*: The text will be cut and displayed in only the part that fits into the visualization element. *"Linebreak"*: The text will be wrapped in rows. *"Ellipsis"*: The text is cut and ellipsis `...` are added to indicate that something is missing.
"Font"	Font of the text. The entries of the selection list are defined in the visualization style.
"Font color"	Text color, for example `Grey`. The entries of the selection list are defined in the visualization style.
"Transparency"	Transparency value (255 to 0), which defines the transparency of the corresponding color. Example: 255: The color is opaque. 0: The color is fully transparent.

Element property 'Absolute movement' The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The "X", "Y", "Rotation", and "Interior rotation" properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'ActiveX'

Symbol:

Category: *"Special Controls"*

The element is used to link an existing ActiveX control in the visualization. You can configure the method calls and their parameters in the element properties of the *"ActiveX"* element.

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“ActiveX”*
“Element”	Installed ActiveX component that is linked to the visualization. Hint: To avoid typing errors, select the required ActiveX component by means of the Input Assistant.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

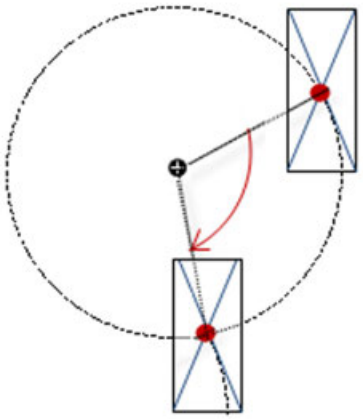
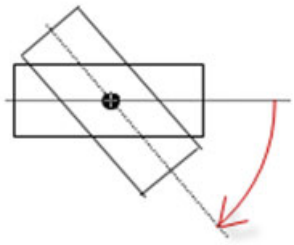
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed. 

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Initial calls'

These method calls are executed during initialization. They are executed in the first cycle only.

“Method calls ”	Button '*“Create new”*' Creates a subnode below *“Methods”* with parameters for the method call.
“Methods”	*“[<number>]”* ● *“Method”*: Name of the method ● *“Parameter”*: Parameter passed at the method call ● *“Result parameter”*: Optional variable for the return value of the method

Element property 'Cyclic calls'

These method calls are executed in every cycle. They are executed in the refresh rate of the visualization.

“Method calls ”	Button '*“Create new”*' Creates a subnode below *“Methods”* for a method call and its parameters.
“Methods”	*“[<number>]”* ● *“Method”*: Name of the method ● *“Parameter”*: Parameter passed at the method call ● *“Result parameter”*: Optional variable for the return value of the method

Element property 'Conditional calls'

These method calls are executed in the refresh rate of the visualization. You define the call condition in the property *“Methods ➔ [<number>] ➔ Call condition”*.

“Method calls ”	Button '*“Create new”*' Creates a subnode below *“Methods”* with a call condition and parameters for the method call.
“Methods”	*“[<number>]”* ● *“Method”*: Name of the method ● *“Call condition”*: Variable (`BOOL`). A rising edge of this variable triggers the call of this method. ● *“Parameter”*: Parameter passed at the method call ● *“Result parameter”*: Optional variable for the return value of the method

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Web Browser'

Symbol:

Category: *“Special Controls”*

The element shows a website, PDF file, or video that has a URL.

NOTICE!
The display options of the *“Web Browser”* element depend on the operating system and the display variant of the visualization.

Requirement: The software components of the web browser are available in the runtime and configured accordingly (example: videos to be shown on Linux).

See also

- *Chapter 1.4.5.21.6 “Displaying Web Contents” on page 2141*

Element properties

“Element name”	Example: `GenElemInst_59`
“Type of element”	*“Web Browser”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

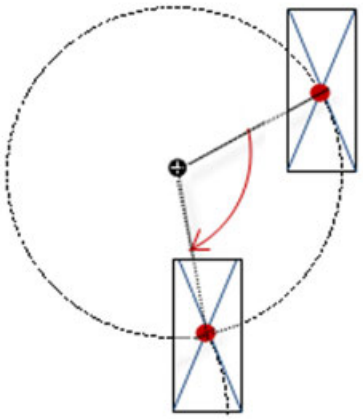
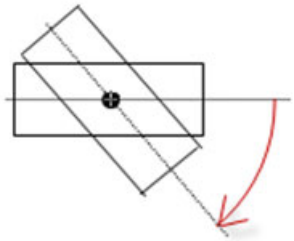
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Control variables'

“URL”	URL of the web page that is displayed in the visualization. ● Variable (`STRING` or `WSTRING`) Example: `PLC_PRG.stURL` ● Literal in single straight quotation marks Example: `'http://de.wikipedia.org'`
“Show”	Variable (`BOOL`). Example: `PLC_PRG.bSetURL` Controls the display of the *“Web browser”* element. If the variable contains a rising edge, then the visualization calls the web page given in *“URL”* and displays its contents in the 'Web browser' visualization element.
“Back”	Variable (`BOOL`). Example: `PLC_PRG.bGoBack` Controls the back navigation in the *“Web browser”*. If the variable has a rising edge, then the visualization displays the contents of the previously displayed page.
“Forward”	Variable (`BOOL`). Example: `PLC_PRG.bGoForward` Controls the forward navigation in the *“Web browser”*. If the variable has a rising edge, then the visualization displays the contents of the previously displayed page.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Busy Symbol, Cube'

Symbol:

Category: *“Special Controls”*

At runtime, this element indicates automatically that the runtime is busy or waiting for data.

Element properties

"Element name"	Example: `Data_Transfer` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Busy Symbol, Cube"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

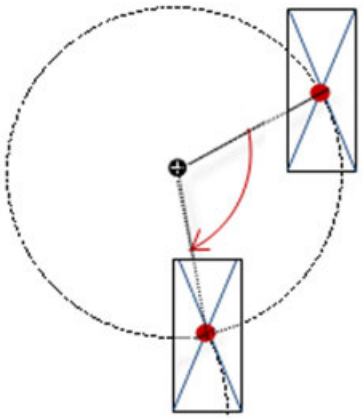
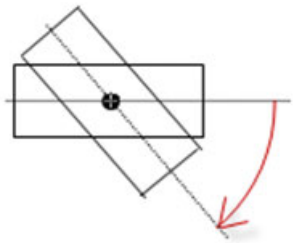
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Busy Symbol, Flower'

Symbol:

Category: *“Special Controls”*

The element indicates that the system is busy or waiting for data.

Element properties

“Element name”	Example: `Data_Transfer` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Busy Symbol, Flower”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Frame color"	
"Fill color"	
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Fill attributes"	The way in which the element is filled. ● *"Filled"*:The element is filled with the color from property *"Colors ➔ Fill color"*. ● *"Invisible"*: The fill color is invisible.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values here are overwritten.

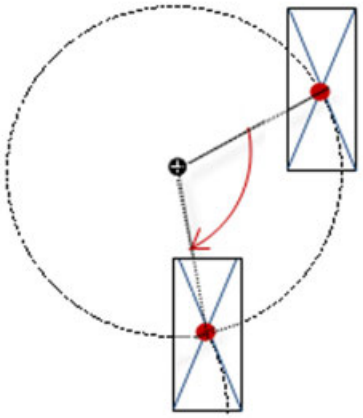
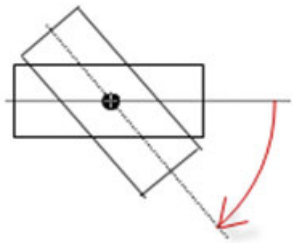
See also

- *" Element property 'Appearance variables'" on page 1671*

“Symbol color”	Selection of a color for the flower symbol.
“Line”	Stroke width of the lines (in pixels).

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Text Editor'

Symbol:

Category: *"Special Controls"*

The element shows the contents of text files that are saved on the controller. Files can be encoded in ASCII or Unicode formats.

A visualization user can also edit the text.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Text Editor"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Font'

“Font name”	Non-proportional font used by the visualization to display the contents of the file Example: *“Courier New”*
“Size”	Font size Example: `12`

Element property 'Control variables'

Table 272: Element property “Control variables --> File”

“Variable”	Variable (`STRING`). Contains the file names and optionally the location of the file. It is located in the file system of the controller. Example: `PLC_PRG.strFile: STRING := '/Documentation/Info.txt';`
“Open”	Variable (`BOOL`). Controls opening the file which is defined in the *“Variable”* property Example: `bOpen: BOOL;` `TRUE`: The file is opened.
“Close”	Variable (`BOOL`). Controls closing the file which is defined in the *“Variable”* property Example: `bClose: BOOL;` `TRUE`: The file is closed.
“Save”	Variable (`BOOL`). Controls saving the file which is defined in the *“Variable”* property Example: `bStore: BOOL;` `TRUE`: The file is saved.
“New”	Variable (`BOOL`). Controls creating a new file. The name is defined in the *“Variable”* property. Example: `bCreate: BOOL;` `TRUE`: A file is created and opened.

Table 273: Element property “Control variables --> Edit ”

“Variable”	Variable (`STRING`). Contains the string to search for in the file Example: `strFind: STRING := 'abc';`
“Find”	Variable (`BOOL`). Controls executing the search for the string in the *“Variable”* property Example: `bFind: BOOL;` `TRUE`: The search is performed. The variable is automatically reset to `FALSE`.
“Find next”	Variable (`BOOL`). Controls the location to begin the search in the file Example: `bFindNext: BOOL;` `TRUE`: The search begins at the last search result location. `FALSE`: The search begins at the beginning of the file.

Table 274: Element property “Control variables --> Cursor position”

“Line”	Variable (integer data type). Contains the line of the cursor Example: `iRowCursor: INT;`
“Column”	Variable (integer data type). Contains the column of the cursor Example: `iColumnCursor: INT;`
“Position”	Output variable (integer data type). Shows the **absolute** cursor position in the text. Example: `iPosCursor: INT;`
“Set cursor”	Variable (`BOOL`). Controls the setting of the cursor at a specific location Example: `iSetCursor: INT;` `TRUE`: The cursor is moved. The new position is defined in the *“Line”* and *“Column”* properties. `FALSE`: The *“Line”*, *“Column”*, and *“Position”* properties contain the actual values. Note: The variable is used as the control variable for an input event triggered by a visualization user.

Table 275: Element property “Control variables --> Selection”

“Start position”	Output variable (integer data type). Shows the **absolute** position for starting the text selection Example: `iPosSelection: INT;`
“End position”	Output variable (integer data type). Shows the **absolute** position for ending the text selection. Example: `iPosEndSelection: INT;`
“Start line number”	Output variable (integer data type). Shows the line where the text selection begins Example: `iRowSelection: INT;`
“Start column index”	Output variable (integer data type). Shows the column where the text selection begins Example: `iColumnSelection: INT;`
“End line number”	Output variable (integer data type). Shows the line where the text selection ends Example: `iRowEndSelection: INT;`

"End column index"	Output variable (integer data type). Shows the column where the text selection ends Example: `iColumnEndSelection: INT;`
"Line to select"	Variable (integer data type). Contains the line number that is selected Note: The selection is controlled by the variables in the *"Trigger selection"* property.
"Set selection"	Variable (`BOOL`). Controls the selection of a line. Example: `bSetSelection: BOOL;` `TRUE`: The line from the *"Line to select"* property is selected and highlighted in the Text Editor. if the line is not in the current text segment of the Text Editor, then the text segment is moved to this line. Note: The variable is used as the control variable for an input event triggered by a visualization user. The control variable is not reset automatically. You are responsible for this to occur in the visualization.

Table 276: Element property "Control variables --> Error handling"

"Variable for error code"	Variable (integer data type). Contains the error code when an error occurs Example: `iError: INT;` The error codes are declared in `GVL_ErrorCodes` in the `VisuElemTextEditor` library. To display the error text, the `VisuFctTextEditorGetErrorText()` function of the library must be called.

"Variable for content changed"	Variable (`BOOL`). Shows whether the contents have changed Example: `bIsContentEdited: BOOL;` `TRUE`: The contents of the Text Editor have changed.
"Variable for access mode"	Variable (`BOOL`). Controls the access privileges to the file Example: `bIsReadOnly: BOOL;` `TRUE`: A visualization user has read-only permission. At runtime, the file contents are highlighted in gray in the Text Editor. `FALSE`: A visualization user has read/write permission. Note: The variable overwrites the setting in the *"Editor mode"* property.

"Maximum line length"	Maximum number of characters per line
"Editor mode"	● *"Read-only"*: A visualization user has read-only permissions to the file. At runtime, the file contents are highlighted in gray in the text editor. ● *"Read/write"*: A visualization user has read-write permissions.

Element property 'New files'

"Encoding"	Character encoding of the new file: ● *"ASCII"* ● *"Unicode (Little endian)"* ● *"Unicode (Big endian)"*
"New line character sequence"	End of line character of the new file: ● *"CR/LF"*: Normal for Windows systems ● *"LF"*: Normal for UNIX systems Please note: When a visualization user opens an existing file, the end-of-line character of the file is detected and used automatically.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

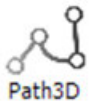
- ↳ *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Path3D'

Symbol:



Category: *“Special Controls”*

The *“Path3D”* visualization element graphically displays the curves of two independent records as a 3D path. It is specially designed for use with Motion Solution CNC in order to display the trajectory of a machine tool or a robot. The programmed path (path) and the path actually traveled (track) is displayed.

Although the visualization element is designed for use with Motion Solution CNC, it can also be used to display any other record. In this case the application has to provide the path data. The sample application `3D Path Generator`, which is available in CODESYS Forge, shows how these data can be generated.

If the element is used together with SoftMotion CNC, then function blocks from the library `SM3_CNC_Visu` help to generate the data from the path and track. These function blocks are used by the sample project `CNC_File_3DPath`, which is stored in the installation directory of CODESYS.

- `SMC_PathCopier`
- `SMC_PathCopierCompleteQueue`
- `SMC_PathCopierFile`
- `SMC_PositionTracker`

A description of the function blocks can be found in the Library Manager in the library `SM3_CNC_Visu`.

> *The element does not work with the CODESYS HMI display variant.*

See also

- *CNC Example 6: Using Path3D with SoftMotion CNC*
- *Sample project in CODESYS Forge*

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Path3D”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Path description

"Path data (VisuStruct3DTrack)"	Variable of the type `VisuStruct3DTrack`, which is declared in the IEC code. Example: `PLC_PRG.pc.vs3dt`. A description of the structure can be found in the library manager in the library `VisuElem3DPath.library`. The data structure describes a path or track through a certain number of points. The points are determined and buffered by the application. The track typically displays the last n positions, so that only a certain part of them is ever displayed at any one time. `VisuStruct3DTrack.pProjection` is a variable that is set by the visualization element and contains information about the path/track projection. It can be read (only) by the application. In addition, the methods `Projection.Apply` or `.ApplyV` can be used in order to see whether the transformed position lies inside or outside the visualization display area, which is defined by `Projection.ElementRect`.
"Path color"	Color of the path drawn
"Path line width"	Path line width in pixels, e.g.: "2"
"Style of boundary points"	Display of the points between two successive objects in the path ● End points are not displayed ● End points are marked with a circle ● End points are marked with a cross ● End points are marked with a plus

Track description The track data are structured in exactly the same way as the path data: VisuStruct3DTrack

"Track data (VisuStruct3DTrack)"	Variable of the type `VisuStruct3DTrack`, which is declared in the IEC code. Example: `PLC_PRG.pc.vs3dt`. A description of the structure can be found in the library manager in the library `VisuElem3DPath.library`.
"Track color"	Color of the track drawn
"Track line width"	Track line width in pixels, e.g.: "2"

Camera control The camera position for the 3D mode is controlled with a reference to the external data structure. This structure allows the following operations:

- Shifting to the left/to the right/upwards/downwards
- Rotation around the X/Y/Z axis
- Resetting of the view at X/Y, Y/Z or Z/X level, so that the path and the track are completely visible.

"Control data (VisuStruct3DControl)"	Variable of the type `VisuStruct3DControl`, which is declared in the IEC code. Example: `PLC_PRG.pc.vs3dc`. A description of the structure can be found in the library manager in the library `VisuElem3DPath`. The values can be set via the application itself or via the visualization element *"ControlPanel"*. The library `VisuElem3DPath` contains ready-to-use visualization frames that provide a possible user interface for these data.

Additional aspects

"Coordinate system"	☑: The coordinate system is displayed
"Grid"	☑: Grid lines are displayed
"Grid color"	Color of the grid lines

Highlighting Individual parts of the path can be visually highlighted. This is typically used to mark the already processed part of a track with a different color. Each point in the path is given a unique ID, which in the case of a CNC editor is linked with the object ID on which the point lies. This ID ("highlight ID") can be specified via the application so that dynamic elements/parts of the track can be highlighted.

Highlight mode	Select one of the following highlight modes: ● Only the element whose ID corresponds to the value of the variable is highlighted. ● All elements whose ID (linked with the object ID in the case of a CNC editor) is smaller than or equal to the value in Variable are highlighted.
Variable	Project variable that specifies the ID of an element. Example: `PLC_PRG.iVarElementID`. This "highlight ID" is taken into account for the setting of the highlight mode. The variable must be set in the IEC application.
Highlight color	

Element look

"Frame line width"	Width of the frame around the element, in pixels, for example: "1"
"Frame line style"	Select one of these style types for the frame line: ● `Solid` ● `Dashes` ● `Dots` ● `Dash Dot` ● `Dash Dot Dot` ● `Hollow`

"Transparent background"	☑: The background of the element is displayed transparently. ☐: The background of the element is displayed in the defined background color.
"Background color"	

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Control Panel'

Symbol:

Category: *“Special Controls”*

This visualization element is used in connection with the *“Path3D”* visualization element. It is used for changing the position and orientation to the CNC path shown with *“Path3D”*.

See also

- *Chapter 1.4.5.18.1.42 “Visualization Element 'Path3D'” on page 1658*

Element properties

“Element name”	Optional. Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Camera_Path_1`
“Type of element”	*“Frame”*
“Clipping”	☑: If you have set the *“Scaling type”* to *“Fixed”*, then only that part of the visualization is displayed that fits in the frame.
“Show frame”	Displays the frame ● *“No frame”*: The displayed area of the frame does not have borders. ● *“Frame”*: The displayed area of the frame has borders. ● *“No frame with offset”*: The displayed area of the frame does not have a border and the displayed area of the referenced visualization is reduced inwards by one pixel as compared to the frame area. The resulting gap prevents the referenced visualization from touching any adjacent elements.
“Scaling type”	Describes how the frame reacts when the visualization is resized: ● *“Isotropic”*: The frame retains its proportions. This allows the ratio of height to width to be preserved, even if the height and width of the visualization have been changed separately. ● *“Anisotropic”*: The frame depends on the size of the visualization, so that height and width of the referenced visualization can be changed separately. ● *“Fixed”*: The original size of the frame is retained, regardless of the visualization size. If you have also selected the *“Clipping”* option, then only the fitting part is displayed. ● *“Fixed and scrollable”*: The referenced visualization is displayed without scaling. If the value is greater than the window area of the frame, then scrollbars are added to the frame. To set the position of the scroll bar with a variable, use the *“Scroll position variable horizontal”* or *“Scroll position variable vertical”* property.
“Deactivation of the background drawing”	☐: To optimize the performance of the visualization, the non-animated elements of the frame element are drawn as a background bitmap. This could result in the elements not being displayed in the expected order. ☑: Deactivation of the background drawing. This can prevent the behavior described above.

Element property 'References'

Contains the currently configured visualization references as a subnode

"References"	Clicking *"Configure"* opens the *"Frame Configuration"* dialog. This is used to manage the referenced visualizations. Caution: Visualizations can be nested at any depth by means of Frame elements. In order to use the *"Switch to any visualization"* Frame selection type without any problems, a Frame must not contain more than 21 referenced visualizations. For more information, see also the description for the *"Input configuration"* of an element: Action *"Switch Frame visualization"*.
List of the currently referenced visualizations	Visualizations that have a button also have this displayed as a subnode. Each interface variable is listed with the currently assigned transfer parameters. Example: `vis_FormA` ● `iDataToDisplay_1 : PLC_PRG.iVar1` ● `iDataToDisplay_2 : PLC_PRG.iVar2` Hint: You can change the assignment of the variables to an interface variable here and edit the value field. Or click the *"Configure"* button instead.

See also

- *Chapter 1.4.5.19.2.1 "Command 'Interface Editor'" on page 1719*
- *Chapter 1.4.5.15 "Creating a structured user interface" on page 1321*
- *"Input action 'Switch Frame Visualization'" on page 1756*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the element in its normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the element in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.
"Use gradient color"	☑: The element is displayed with a color gradient.
"Gradient setting"	The *"Color gradient editor"* dialog box opens.
"Frame color"	Example: *"Black"*
"Fill color"	Example: *"Light gray"*

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*
-

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- “ *Element property 'Appearance variables'” on page 1671*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Text”*.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Tooltip”*.

See also

- *“Element property 'Text variables'” on page 1667*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

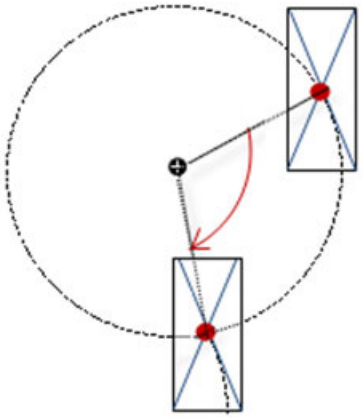
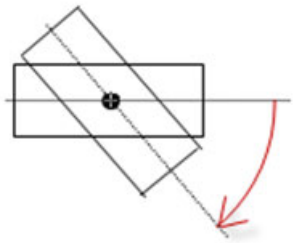
Element property 'Text properties'

The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 1698*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1665*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

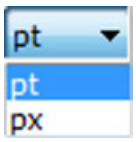
“Text list”	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
“Text index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
“Tooltip index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1665*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

“Toggle color”	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *“Color”* property. ● `TRUE`: The element is displayed with the color specified in the *“Alarm color”* property. Assignment options: ● Placeholder for the user input variable – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
“Normal state” *“Alarm state”*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *“Color variables”*, *“Toggle color”* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *“Colorvariables”*, *“Toggle color”* has the value `TRUE`.
“Frame color”	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
“Filling color”	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the “Activate semi-transparent drawing” option of the visualization manager is selected.

Select the “Advanced” option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Element property 'Appearance variables'

The properties contain variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels). Note: The values 0 and 1 both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible: The line is not drawn.

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1664*

Element property 'Switch frame variable'

The variable controls the switching of the referenced visualizations. This variable indexes one of the referenced frame visualizations and this is displayed in the frame. When the value of the variable changes, it switches to the recently indexed visualization.

"Variable"	● Variable (integer data type) that contains the index of the active visualization Example: `PLC_PRG.uiIndexVisu` Hint: The *"Frame Configuration"* dialog includes a list of referenced visualizations. The visualizations are automatically numerically indexed via the order in the list. Note: This variant of switching usually affects all connected display variants. ● Array element (integer data type) for index access via `CURRENTCLIENTID` Example: `PLC_PRG.aIndexVisu[CURRENTCLIENTID]` Note: This variant of switching applies to the current client only, and therefore only on one display variant. That is the display variant where the value change was triggered (for example, by means of user input).

See also

- *Chapter 1.4.5.19.2.9 "Command 'Frame Selection'" on page 1727*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *“Configure”* button opens the *“Input Configuration”* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *“Execute ST Code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog.
“OnMouseClick”	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the mouse button.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Cartesian XY Chart'

Symbol:

Category: *"Special Controls"*

The element displays the curve of array values graphically as a line or bar chart in the Cartesian coordinate system. The chart can display multiple curves at one time.

NOTICE!
Constraint
The element can be used with controller with V3.5 SP11 and higher.

in CODESYS Forge, you will find a sample project for using “Cartesian XY Chart” elements in visualizations.

See also

- *Sample project in CODESYS Forge*

Element properties

“Element name”	Example: `Velocity chart`
“Type of element”	*“Cartesian XY Chart”*
“Cartesian XY Chart”	`XYChart`: Opens the *“XY Chart Configuration”* dialog. This is where the chart is configured.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Element look'

The properties contain fixed values for defining the look of the element.

“Border line width”	Value (in pixels) Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *“Border line style”* property must be set to the option *“Invisible”*.
“Border line style”	● *“Solid”* ● *“Dash”* ● *“Dots”* ● *“Dash Dot”* ● *“Dash Dot Dot”* ● *“Invisible”*
“Frame line color”	● Style color from the list box. Example: `Black` ● Fixed value that is selected in the color dialog. Example: `0; 0; 0`

Element property 'Axis font'

“Font”	Example: *“Default”* ...: Opens the *“Font”* dialog. ▼: List box with style fonts

Element property 'Control variables'

Table 277: “Zoom”

	Zooming the displayed curve is done by means of the mouse, or the pinch gesture on a multitouch device. It also applies to all axes. At runtime when *“Enable”* is `TRUE`, you can draw a box with the mouse by holding down the left mouse button. When you release the mouse button, the display zooms in on the box and the curve is magnified. To zoom in and out on a multitouch device, move two fingers together or away from each other, respectively. Zooming and panning can work together.
“Enable”	Variable (`BOOL`) that enables or disables zooming. `TRUE`: Enables zooming Example: `PLC_PRG.xZoomEnable`
“Home”	Variable (`BOOL`) Rising edge: Reset the displayed curve to the initial state after the display has changed due to zooming. Example: `PLC_PRG.xZoomHome`

“Undo”	Variable (`BOOL`) Rising edge: Reset the displayed curve to the previous position after the display has changed due to zooming. Example: `PLC_PRG.xZoomUndo`
“Is zoomed”	Variable (`BOOL`) that indicates whether or not the displayed curve was modified due to zooming. `TRUE`: Curve setting was zoomed. Example: `PLC_PRG.xIsZoomed`

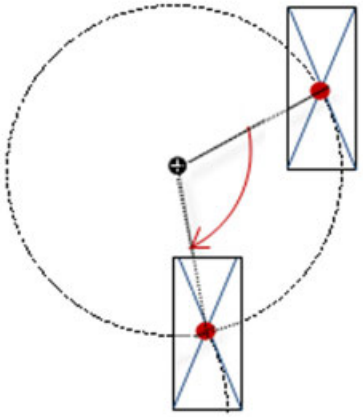
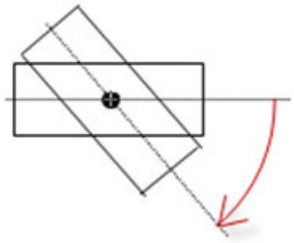
Table 278: “Pan”

	Panning the displayed curve is done by means of the mouse or the pinch gesture on a multitouch device. It also applies to all axes. At runtime if *“Enable”* is `TRUE`, then you can drag the displayed curve to another position by holding down the left mouse button. To pan the displayed curve on a multitouch device, drag it with one finger to another position.
“Enable”	Variable (`BOOL`) to enable or disable panning. `TRUE`: Enables panning Example: `PLC_PRG.xPanEnable`
“Home”	Variable (`BOOL`) Rising edge: Reset the displayed curve to the initial position after the display has changed due to panning. Example: `PLC_PRG.xPanHome`
“Is panned”	Variable (`BOOL`) whose state indicates whether or not the displayed curve was modified due to zooming. `TRUE`: Curve setting was panned. Example: `PLC_PRG.xIsPanned`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Date Range Picker'

Symbol:

Category: *“Date/Time Controls”*

The element provides the capability of selecting the date and time range of a saved data set. The element is used with the *“Trend”* visualization element.

Element properties

“Element name”	Example: `DateTrend1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Date Range Picker”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

"Show frame"	☑: The visualization element is drawn with a frame.
"Resolution"	Resolution saved for the time stamp: *"Millisecond"* or *"Microsecond"*
"Attached element instance"	The element can be assigned to a *"Trend"* visualization element. As a result, the time range of the trend element can be changed. The available visual elements are selected with the help of the Input Assistant (...).

Element property 'Tick mark labels'

"Two-line labelling"	☑: The time stamps are displayed in two lines. The date is displayed in the first line and the time is displayed in the second line. ☐: Time stamp is displayed in one line. The date and time can also be displayed in one line depending on the formatting.
" Omit irrelevant information in time stamp"	☑: The time stamp has a shorter form. For example, the date is displayed only for the first tick mark, and only the time for the following tick marks. The settings in *"Internationalization (format strings)"* are ignored for this setting. ☐: All information is displayed for all time stamps.
"Internationalization (format strings)"	Only active when the parameter *"Omit irrelevant information in timestamps"* is deactivated.
"Date"	Definition of the date format. The default setting is taken from the Windows control panel.
"Time"	Definition of the time format. The default setting is taken from the Windows control panel.

Element property 'Text properties'

The properties contain fixed values for the text properties.

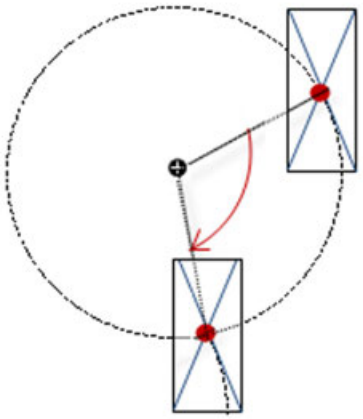
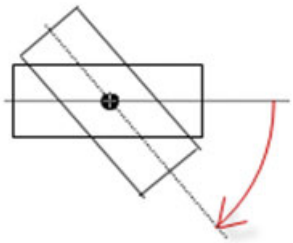
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Additional buttons'

"Jump to the largest possible time stamp"	☑: An additional button ( ) is displayed for jumping to the last time stamp.
"Jump to the smallest possible time stamp"	☑: An additional button ( ) is displayed for jumping to the first time stamp.
"Zoom out"	☑: An additional button ( ) is displayed for setting the current min./max. range to the maximum range. The selected range is left.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Time Range Picker'

Symbol:

Category: *"Date/Time Controls"*

The element provides configurable buttons for setting the time range of a trend display to a defined time. In the process the end time of the previous display is left unchanged and the start time is adapted.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `TimeRangeTemperature`
"Type of element"	*"Time range picker"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Orientation"	Specifies whether the time picker element is aligned horizontally or vertically in the editor. Hint: Change the width to height ratio of the element in the editor.
"Show frame"	☑: The visualization element is drawn with a frame.
"Resolution"	Resolution saved for the time stamp: *"Millisecond"* or *"Microsecond"*
"Attached element instance"	Assignment to the element that processes the time picker The element can be assigned for example to a *"Trend"* visualization element. Then the time range of the trend element can be changed. The available visual elements are selected with the help of the input assistance (...). Example: `GenElemInst_1`

Element property 'Texts'

"Text"	String label for the element. Example: `Zoom`

Element property 'Text properties' The properties contain fixed values for the text properties.

“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Property 'Times' In *“Times”*, the buttons that the element provides at runtime are defined and configured in an array.

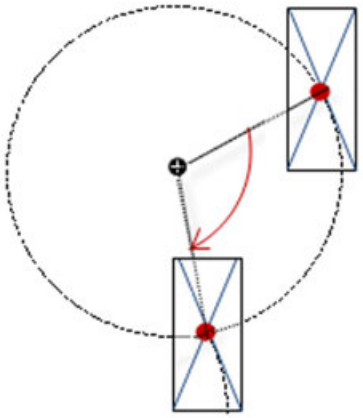
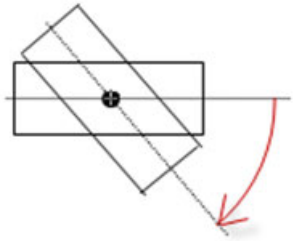
“Provide "All" selection”	☑:Time Range Picker bar extended by "All" button. The diagram represents a time interval that covers all time stamps.
“Times”	: Adds another button to the Time Range Picker bar and increases the array by one entry. An additional index is present in the property *“Times ➔ Times ➔ Times ➔ [<new>]”*. *“Time”* is located under this index. The configuration of the button is to be entered there.
“Times” ● *“ [Index]”* with index ∈ {0, 1, 2,...}	Array of all buttons in the time selection bar. Index corresponds to the number of buttons. ✕: The associated button is removed from the Time Range Picker bar. The configuration entry is deleted from the *“Times”* property list.
“ [Index]” ● *“Time”*	: Time interval in standardized notation. Example: `3M` for 3 months; `30m` for 30 minutes. If a time interval is indicated in the field, then the button is labelled with it. If a user clicks on the button at runtime, the command is executed to switch the diagram to this time interval. The default is empty.

Element property 'Control variables'

“Time”	Displays which time is currently selected. Variable (`STRING`) Example: `PLC_PRG.strSelcetedTime`
“"All" selected”	Displays the state of the "All" button Variable (`BOOL`) Example: `PLC_PRG.AllTimesAreSelected`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Date Picker'

Symbol:

Category: *“Date/Time Controls”*

The element is a calendar that displays the current date. A user can click a day to select a date, which is saved to a variable. In addition, it can customize the time interval that the calendar displays. Clicking the calendar header changes the year. Clicking the arrows in the calendar header changes the month.

Language-dependent texts of the element

The element contains language-dependent texts that are managed in the `System` text list. This deals with the names of the month and the days of the week written out completely or abbreviated. When the date picker is added to a visualization, CODESYS generates the text list automatically below the POU view. The IDs correspond to the standard text and therefore English terms. The text list makes it possible to translate these texts.

Example

`System` text list

ID	Default
`Apr`	`Apr`
`April`	`April`

See also

- *Chapter 1.4.5.6 “Setting Up Multiple Languages” on page 1286*

Element properties

“Element name”	Example: `DueDateCalendar`
“Type of element”	*“Date Picker”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.

“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

“Variable”	Input variable (`DATE`). Contains the date that a user selects in the calendar. Example: `PLC_PRG.dtDueDate`
“Design”	• *“From style”*: All settings are preconfigured according to the style. • *“Explicit”*: The *“Design settings”* property is available. You can customize the calendar here.

Design settings

Requirement: This property is visible only if the *“Design”* property is set to *“Explicit”*.

The values of the property can be predefined in the style. Then they are available in the drop-down list.

Table 279: “Header of Date Picker”

Design of the header	
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Arrows”	
“Arrow color”	Style color or user-defined color
“Color of printed arrow”	
“Background”	

“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Fill color”	Style color or user-defined color

Table 280: Design of the main display area

Design of the main display area	
“Today”	Design of today
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The style defines whether and which background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	Style color or user-defined color. Used if *“Yes”* is selected in *“Draw background”*.
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Selected day”	Design of the selected day
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	Style color or user-defined color
“Show frame”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Current month”	Design of the current month
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color

“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Other months”	Design of the previous and subsequent months
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Display other month”	Design of the previous and subsequent months
“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Day of week heading”	Design of the heading with the days of the week
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The background is filled with the style color *“From style”*. The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

"Display separator line"	*"From style"*: The style defines whether and how a separator line is drawn. *"Yes"*: Display with the following properties. *"No"*: A separator line is not displayed.
"Color of the separator line"	Used if *"Yes"* is selected in *"Display separator line"*.
"Width of separator line"	

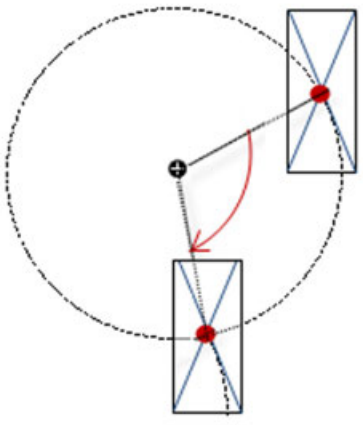
"Background"	Design of the calendar days
"Draw background"	*"From style"*: The style defines whether and how a background is drawn. *"Yes"*: The background is filled with the color in the *"Fill color"* property and framed in the *"Frame color"*. *"No"*: The background is not filled with a color.
"Fill color"	Style color or user-defined color
"Frame color"	

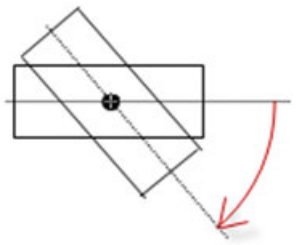
Element property 'Display type'

"Rows"	Number of month calendars per row (preset: `1`)
"Columns"	Number of month calendars per column (preset: `1`)

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

● *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Analog Clock'

Symbol:

Category: *"Date/Time Controls"*

The element is a clock that displays the current time of day. The clock can also display a random time.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Analog Clock"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Time Display'

"Use system time"	☑: The system time of the PLC is displayed.
"Variable"	Variable (time data type `TOD`, `TIME_OF_DAY`). This receives the time of day that is not the system time. Example: `PLC_PRG.todTimeTokio` Requirement: The *"Use system time"* property is not activated.

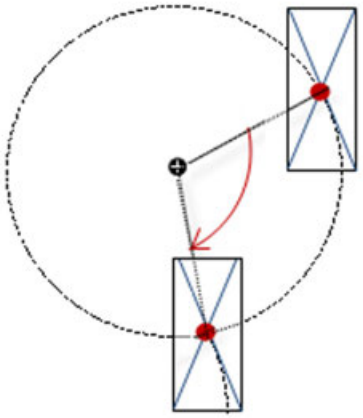
See also

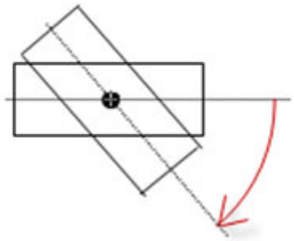
- *Chapter 1.4.1.19.5.5 "Data Type 'TIME'" on page 649*

"Design"	• *"From style"*: All settings are preconfigured according to the style. • *"Explicit"*: The *"Settings"* property is available. Here you can customize the analog clock.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
“Use REAL values”	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Settings'

Requirement: The *“Property”* is *“Explicit”*. Only then is the *“Clock Settings”* category visible.

Table 281: “Background”

“Background color”	Color variants of the default background image ● *“Yellow”* ● *“Red”* ● *“Blue”* ● *“Green”* ● *“Black”*
“Own background”	Background display with the specific *“Image”*. Replaces the default background image.

"Image"	Image from an image pool or library Example: `myImagepool.myImage`
"Transparency color"	The transparent color in the image representation. Example: *"White"*. The white parts of the image are transparent.
"Use background color"	☑: The image background is displayed using the color defined in the *"Background color"* property. Requirement: No image reference is given in the *"Image"* property.
"Background color"	Style color or color Requirement: *"Use background color"* is activated.

Table 282: "Hands"

"Hand style"	Example: *"Thin arrow"*
"Color hour hand"	Style color or color for the hands
"Color minute hand"	
"Color second hand"	

Table 283: "Lines"

"Lines style"	Clock face graduation ● *"None"* ● *"Line"*: Graduation lines by hour ● *"Hours and minutes"*: Graduation lines by hours and minutes ● *"Dots"*: Graduation dots by hour
"Color"	Color of the clock face graduation
"Line width"	Line weight of the clock face graduation
"Scale in 3D"	☑: Representation of the clock face with 3D effect

Table 284: "Numerics"

"Style of numerics"	Digits on the clock face ● *"None"* ● *"Quarter"* ● *"All"*
"Font"	Font for displaying the digits
"Font color"	Font for displaying the digits

Table 285: "Center point"

"Color"	Color of the center of the clock

Table 286: "Positioning"

"Usage of"	● *"Default style values"*: Presetting of the style values ● *"Individual settings"*: User-defined settings in the subordinate *"Positioning"* property.
"Positioning"	Requirement: Visible when the *"Usage or"* property is set to *"Individual settings"*.

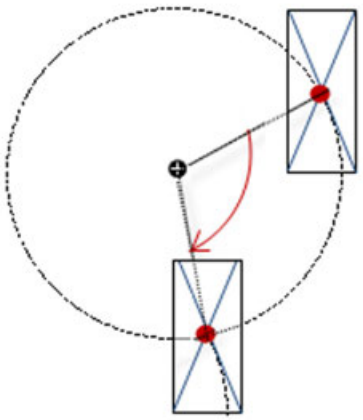
"Numerics movement"	Value (in pixels) for shifting the digits. Example: `80`
"Line movement"	Value (in pixels) for shifting the hour lines. Example: `100`
"Hands scaling"	Factor for scaling the length of the hour hand. You can customize the exact position of the hour hand relative to the background image. Example: `100`

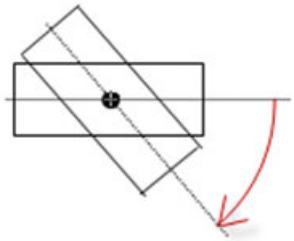
"Scaling type"	Defines the scaling of the height and width of the element. ● *"Anisotropic"*: The background image is scaled to the size of the element The height and width are scaled independently of each other. ● *"Isotropic"*: The background image is scaled to the size of the element, retaining its proportion. The proportion of height and width is fixed.

"Optimized drawing"	☑: The background image is drawn one time. When the hour hand moves, only the affected part of the image is redrawn. ☐: The background image is redrawn in cycles. Hint: Disable this option only for extreme exceptions.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Date/Time Picker'

Symbol:

Category: *"Date/Time Controls"*

The element provides the capability of selecting the date and time. The value can be changed by means of the arrow keys on the keyboard. The date can be selected from a calendar.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `StartDateAndTime`
"Type of element"	*"Date/Time Picker"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	Variable (`DATE, DT, TIME, LTIME, TOD`) The value of the value of the variable is displayed and modified by means of the element. The data type automatically determines the displayed value units: ● `TIME`: Day, hour, minute, and second (by default, milliseconds are not displayed) ● `DATE`: Year, month, and day ● `DT`: Year, month, day, hour, minute, and second ● `TOD`: Hour, minute, and second (by default, milliseconds are not displayed) ● `LTIME`: Day, hour, minute, and second (by default, milliseconds, microseconds, and nanoseconds are not displayed)
"Format string"	The format can restrict the output to individual values. Example for `LTIME`: Format: `HH:mm:ss.ms.us.ns` --> displayed: 08:15:12.780.150.360 `LTIME` restricted: format: `HH:mm` --> displayed: 08:15 Example for `DATE`: Format: `yyyy/MM/dd` --> displayed: 2015/12/17 . Basically, all usual formats available for %t are also supported.
"Design date time picker"	● *"From style"*: All settings are preconfigured according to the style. ● *"Explicit"*: The *"Design settings"* property is available. You can customize the calendar here.
"Design date picker"	● *"From style"*: All settings are preconfigured according to the style. ● *"Explicit"*: The *"Design settings"* property is available. You can customize the calendar here.
"Positioning date picker"	● *"Dynamic"*: The calendar is adapted and positioned automatically. ● *"Manual"*: The *"Position settings"* property is available. You can customize the calendar here.

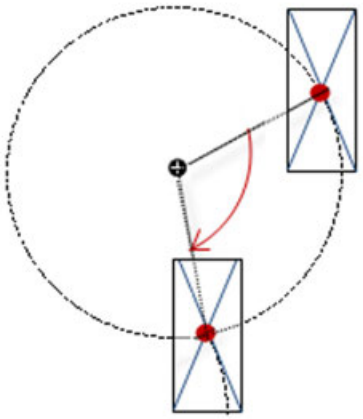
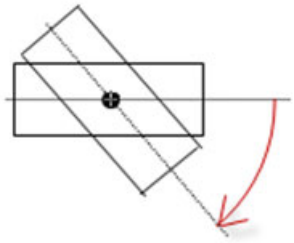
See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The properties "X", "Y", "Rotation", and "Interior rotation" are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

1.4.5.18.2 Placeholders with Format Definition in the Output Text

A character string which is output in the visualization can include the placeholder `%` for a variable. At runtime, the placeholder is replaced by the actual value of the variable in the defined format. The data type in the format definition and of the variable have to be identical. A character string can contain a maximum of one placeholder.

Character strings for output are listed in the *“Text”* property. The assigned variable is listed in the *“Text variable”* property.

See also

- *Integer Data Types*
- *REAL/LREAL Data Type*
- *Time Data Types*

For the output of integers

`%d` `%i`	Output of variable (integer data type) as decimal number	Code: `iCounter : INT := 12;` Property *“Text”*: `Value: %i` Property *“Text variable”*: `PLC_PRG.iCounter` Output: `Value: 12`
`%b`	Output of variable (integer data type) as binary number	Code: `byCode : BYTE := 255;` Property *“Text”*: `Coding: %b` Property *“Text variable”*: `PLC_PRG.byCode` Output: `Coding: 11111111`
`%o`	Output of variable (integer data type) as unsigned octal number without a preceding zero	Code: `byCode : BYTE := 8#377;` Property *“Text”*: `Coding: %o` Property *“Text variable”*: `PLC_PRG.byCode` Output: `Coding: 377`
`%x`	Output of variable (integer data type with max. 32 bits) as unsigned hexadecimal number without a preceding "0x"	Code: `dwCode : INT := 16#FFFFFFFF;` Property *“Text”*: `Coding: %x` Property *“Text variable”*: `PLC_PRG.dwCode` Output: `Coding: ffffffff`
`%llX` `%012llX`	Output of 64-bit variable (`LWORD`, `LINT`, `ULINT`) as hexadecimal number. Note: `llx` means "long long hexadecimal"	Code: `lwCode : LWORD := 16#4FFF_3FFF_2FFF_1FFF;` Property *“Text”*: `Coding: %llx` Property *“Text variable”*: `PLC_PRG.lwCode` Output: `Coding: 4fff3fff2fff1fff`
`%u`	Output of variable (integer data type) as unsigned decimal number	Code: `uiNumber : UINT := 1234;` Property *“Text”*: `Number: %u` Property *“Text variable”*: `PLC_PRG.uiNumber` Output: `Number: 1234`

For the output of floating-point numbers

Floating-point numbers have the data type `REAL` or `LREAL`.

`%f`	In decimal form with decimal point in format 1.6	Code: `rWeight : REAL := 1.123456789;` Property *"Text"*: `Weight: %f` Property *"Text variable"*: `PLC_PRG.rWeight` Output: `Weight: 1.123456`
`%<alignment><minimum width>.<accuracy>f`	As decimal number in user-defined format ● `<alignment>`: `-` or `+`, optional `-`: Left-aligned `+`: Right-aligned ● `<minimum width>`: Number of places to the left of the decimal point ● `<accuracy>`: Number of places to the right of the decimal point	Code: `rWeight : REAL := 12.1` Property *"Text"*: `Weight: %2.3f` Property *"Text variable"*: `PLC_PRG.rWeight` Output: `Weight: 12.100`
`%e`	Output of floating-point number (`REAL` or `LREAL`) in exponential notation of base `10`	Code: `rValue : REAL := 1.234567%e-003;` Property *"Text"*: `Value: %E` Property *"Text variable"*: `PLC_PRG.rValue` Output: `Value: 1.23E-6`
`%E`		Code: `rValue : REAL := 1.234567%e-003;` Property *"Text"*: `Value: %e` Property *"Text variable"*: `PLC_PRG.rValue` Output: `Value: 1.23e-6`

For the output of text

`%c`	Output of single character in ASCII character set	Code: `bChar := 16#41;` Property *"Text"*: `Key: %c` Property *"Text variable"*: `PLC_PRG.bChar` Output: `Key: A`
`%s`	Output of character string	Code: `strName := 'Paul Smith';` Property *"Text"*: `Name: %s` Property *"Text variable"*: `PLC_PRG.strName` Output: `Name: Paul Smith`

For the output of the percent sign

`%%`	Output of percent sign in character string	Property *"Text"*: `Valid until 90%%` Output: `Valid until 90%`
		Code: `iPercentage : INT := 80;` Property *"Text"*: `Valid until %d%%.` Property *"Text variable"*: `PLC_PRG.iPercentage := 80;` Output: `Valid until 80%`

For the output of the date and time

If the output text in the element *"Text"* property contains the placeholder "%t", then a date and/or time is output. If a variable is **not** specified in the *"Text variable"* property, then the **system time** is output; otherwise it is the value of the variable.

By default, the names of the days and months are displayed in English. If localized texts are used, then the text list `System` has to be supplemented. This text list is created automatically in the *"POUs"* view when the placeholder %t is used. The English terms have to be used as the ID here. The localization can be done for both the abbreviated names and full names.

Time data types include `LTIME`, `TIME`, `TIME_OF_DAY`, `TOD`, `DATE`, `DATE_AND_TIME`, and `DT`.

Compatibility Notice

In order to get the usual display, in V3.5 SP17 and higher, as a rule three digits are used for the output of fractions of a second (ms/µs/ns). Example: In `%t[dd-HH:mm:ss:ms], ms` is specified with three digits for the milliseconds. For this purpose, the two-digit ms number is prepended with a zero. If a two-digit output is desired (like before V3.5 SP17), then a special compiler define has to be set in the compiler properties of the application: `VISU_MILLISEC_NOLEADING_ZERO`.

Date and time formats		
`%t[yyyy]`	Year with century	Code: `dateBy : DATE := DATE#2020-1-1;` Property *"Text"*: `By the year %t[yyyy]` Property *"Text variable"*: `PLC_PRG.dateBy` Output: `By the year 2020`
`%t[yy]`	Year without century (00–99)	Code: `dateSince : DATE := DATE#2000-1-1;` Property *"Text"*: `Since: %t[yy]` Property *"Text variable"*: `PLC_PRG.dateSince` Output: `Since: 00`
`%t[y]`	Year without century (0–99)	Code: `dateSince : DATE := DATE#2000-1-1;` Property *"Text"*: `Since: %t[y]` Property *"Text variable"*: `PLC_PRG.dateSince` Output: `Since: 0`

`%t[MMMM]`	Month as full name	Code: `dateMonth : DATE := DATE#2016-1-1;` Property *"Text"*: `Month: %t[MMMM]` Property *"Text variable"*: `PLC_PRG.dateMonth` Output: `Month: January`
`%t[MMM]`	Month as abbreviated name	Code: `dateMonth : DATE := DATE#2016-1-1;` Property *"Text"*:: `Month: %t[MMM]` Property *"Text variable"*: `PLC_PRG.dateMonth` Output: `Month: Jan`
`%t[MM]`	Month as number (01–12)	Code: `dateMonth : DATE := DATE#2016-1-1;` Property *"Text"*:: `Month: %t[MM]` Property *"Text variable"*: `PLC_PRG.dateMonth` Output: `Month: 01`
`%t[M]`	Month as number (1–12)	Code: `dateMonth : DATE := DATE#2016-1-1;` Property *"Text"*:: `Month: %t[M]` Property *"Text variable"*: `PLC_PRG.dateMonth` Output: `Month: 1`
`%t[ddddd]`	Day of week as number (1=Monday – 7=Sunday)	Code: `iDay : INT := 7;` Property *"Text"*:: `Day: %t[ddddd]` Property *"Text variable"*: `PLC_PRG.iDay` Output: `Day: 7`
`%t[dddd]`	Day of week as full name	Code: `iDay : INT := 7;` Property *"Text"*:: `Day: %t[dddd]` Property *"Text variable"*: `PLC_PRG.iDay` Output: `Day: Sunday`
`%t[ddd]`	Day of week as abbreviated name	Code: `iDay : INT := 7;` Property *"Text"*:: `Day: %t[ddd]` Property *"Text variable"*: `PLC_PRG.iDay` Output: `Day: Sun`
`%t[dd]`	Day of month as number (01–31)	Code: `iDay : INT := 1;` Property *"Text"*:: `Day: %t[dd]` Property *"Text variable"*: `PLC_PRG.iDay` Output: `Day: 01`

`%t[d]`	Day of month as number (1–31)	Code: `iDay : INT := 1;` Property *"Text"*:: `Day: %t[d]` Property *"Text variable"*: `PLC_PRG.iDay` Output: `Day: 1`
`%t[jjj]`	Day of year as number (001–366)	Code: `dateOfNoReturn : DATE := DATE#2016-09-01;` Property *"Text"*:: `Day of no return: %t[jjj]` Property *"Text variable"*: `PLC_PRG.dateOfNoReturn` Output: `Day of no return: 245`
`%t[HH]`	Hour in 24-hour format (00–23)	Code: `todEnd : TOD := TIME_OF_DAY#17:0:0;` Property *"Text"*: `Ends at: %t[HH]:00` Property *"Text variable"*: `PLC_PRG.todEnd` Output: `Ends at 17:00`
`%t[hh]`	Hour in 12-hour format (01–12)	Code: `todEnd : TOD := TIME_OF_DAY#17:0:0;` Property *"Text"*: `Ends at: %t[hh] o'clock` Property *"Text variable"*: `PLC_PRG.todEnd` Output: `Ends at: 05 o'clock`
`%t[mm]`	Minutes with leading zero (00–59)	Code: `tPeriod : TIME := T#5M;` Property *"Text"*: `Period: %t[mm]m` Property *"Text variable"*: `PLC_PRG.tPeriod` Output: `Period: 05m`
`%t[m]`	Minutes without leading zero (0–59)	Code: `tPeriod : TIME := T#5m;` Property *"Text"*: `Period: %t[m 'm']` Property *"Text variable"*: `PLC_PRG.tPeriod` Output: `Period: 5 m`
`%t[ss]`	Seconds with leading zero (00–59)	Code: `tPeriod : TIME := T#5m3s;` Property *"Text"*: `Period: %t[mm'm'ss's']` Property *"Text variable"*: `PLC_PRG.tPeriod` Output: `Period: 05m03s`
`%t[s]`	Seconds without leading zero (0–59)	Code: `tPeriod : TIME := T#5m3s;` Property *"Text"*: `Period: %t[m'm' s's']` Property *"Text variable"*: `PLC_PRG.tPeriod` Output: `Period: 5m 3s`

`%t[ms]`	Milliseconds without leading zero (0–999)	Code: `tPeriod : TIME := T#500ms;` Property *"Text"*: `Period: %t[ms 'ms']` Property *"Text variable"*: `PLC_PRG.tPeriod` Output: `Period: 500 ms`
`%t[us]`	Only for `LTIME` variables: microsecond definition (0–999)	Code: `ltPeriod :LTIME := LTIME#1000D23H44M12S34MS2US44NS;` Property *"Text"*: `'Period': %t[dd.HH.m.s.ms.us.ns]` Property *"Text variable"*: `PLC_PRG.ltPeriod` Output: `Period: 1000.23.44.12.34.2.44` Hint: Overflow is permitted in the greatest time unit of a definition.
`%t[ns]`	Only for `LTIME` variables: nanosecond definition (0–999)	
`%t[t]`	If the value is a time < 12h, then A is output; otherwise P is output.	Code: `tClosed : TOD := TOD#17:17:17.17;` Property *"Text"*: `Closed at %t[hh:mm t]` Property *"Text variable"*: `PLC_PRG.tClosed` Output: `Closed at 05:17 P`
`%t[tt]`	If the value is a time < 12h, then AM is output; otherwise PM is output.	Code: `tClosed : TOD := TOD#17:17:17.17;` Property *"Text"*: `Closed at %t[hh:mm tt]` Property *"Text variable"*: `PLC_PRG.tClosed` Output: `Closed at 05:17 PM`
`%t[' ']`	If character strings should be output which correspond to a format definition, then these have to be represented in single straight quotation marks.	
`TIME` and `LTIME` values can be specified with integer values or with decimal places:		

`%t[<f><n>]`	A number (<n>) which defines the number of decimal places of the time value follows the letters which define the time unit (<f>). As a result, the hours, minutes, and seconds (for TIME values) and also the microseconds and nanoseconds (for LTIME values) can be specified or displayed as values with decimal places. Note: Even if a decimal number is not desired for the input or display, at least the number "0" has to be specified to allow for fractional input.	Examples of the formating `%t[hh4]` or `%t[HH4]`: The time can be specified/displayed with a hour definition of four decimal places. `%t[mm2]` or `%t[m2]`: The time can be specified/displayed with a minute definition of four decimal places. Then for a value of `t#1h20m15s`, this leads to the following output: `80.25`. `%t[ss0]`: The time can be specified/displayed with a second definition without decimal places.
The format definitions can be represented in a series.		
`%t[HH:mm:ss:ms]`	Output of the time	Code: `dwTime : DWORD := 4294967295;` Property *"Text"*: `Time: %t[HH:mm:ss:ms]` Property *"Text variable"*: `PLC_PRG.dwTime` Output: `Time: 23:59:59:999`
%t[yyyy-MM-dd dddd]	Output of the date and day of the week	Code: `dateSet : DATE := DATE#2016-02-12;` Property *"Text"*: `Date: %t[yyyy-MM-dd dddd]` Property *"Text variable"*: `PLC_PRG.dateSet` Output: `Date: 2016-02-12 Friday`

See also

- *Time Data Types*

1.4.5.18.3 Methods of the Dialog Manager

Visualizations that are a *"Dialog"* visualization type and are used to prompt an input are instantiated automatically and managed by the internal dialog manager.

In the application, the dialog manager can be accessed via the also internal Visualization Manager by calling the method `GetDialogManager`.

The dialog manager is provided with the following methods for handling a dialog.

NOTICE!

You can program the method calls in function blocks or functions which are themselves called from the visualization by the action `Execute ST Code`.

Moreover, you can program the method calls in the application code. Make sure that the call runs in `VISU_TASK`. If this is not the case, then the behavior is undefined.

Method 'GetDialog' Returns the instance (`IVisualisationDialog`) of the dialog whose name is passed.

Table 287: Inputs (VAR_INPUT)

Name	Data Type	Description
`stName`	`STRING`	Name of the dialog

Table 288: Outputs (VAR_OUTPUT)

Name	Data Type	Description
`GetDialog`	`VisuElems.IVisualisationDialog`	Instance (`IVisualisationDialog`) of the dialog

Method 'GetClientInterface' Returns a pointer to the dialog structure.

Respective dialog data held for each display variant.

Table 289: Inputs (VAR_INPUT)

Name	Data Type	Description
`dialog`	`VisuElems.IVisualisationDialog`	Name of the visualization
pClient	`POINTER TO VisuElems.IVisualisationDialogVisuStructClientData`	Pointer to the display variant

Table 290: Outputs (VAR_OUTPUT)

Name	Data Type	Description
`GetClientInterface`	Example: `POINTER TO Login_VISU_STRUCT`	Pointer to the dialog structure

Method 'OpenDialog' Opens the dialog of the client.

Next to it, there is the extended method 'OpenDialog(number)'.

Table 291: Inputs (VAR_INPUT)

Name	Data Type	Description
`dialog`	`VisuElems.IVisualisationDialog`	Name of the visualization
`pClient`	`POINTER TO VisuElems.VisuStructClientData`	

Name	Data Type	Description
bModal	BOOL	
pRect	POINTER TO	

Table 292: Outputs (VAR_OUTPUT)

Name	Data Type	Description
OpenDialog		

Method 'Close-Dialog' Closes the dialog of the client.

Table 293: Inputs (VAR_INPUT)

Name	Data Type	Description
dialog	VisuElems.IVisualisationDialog	Dialog object as received by GetDialog
pClient	POINTER TO VisuElems.VisuStructClientData	

Table 294: Outputs (VAR_OUTPUT)

Name	Data Type	Description
CloseDialog		

Method 'Close-Dialog2' Closes the dialog of the client. Extension of the method CloseDialog.

Table 295: Inputs (VAR_INPUT)

Name	Data Type	Description
dialog	VisuElems.IVisualisationDialog	Dialog object as received by GetDialog
pClient	POINTER TO VisuElems.VisuStructClientData	
DialogFlags	DWORD	Specification of possible options for closing the dialogs. Only the values 0 (behavior as for CloseDialog) and 16#40 are relevant in the case that a dialog should be closed on all connected clients.

Table 296: Outputs (VAR_OUTPUT)

Name	Data Type	Description
CloseDialog2		

1.4.5.18.4 Attribute 'VAR_IN_OUT_AS_POINTER'

Function: The pragma {attribute 'VAR_IN_OUT_AS_POINTER'} allows for the passing of a reference to a data object to the interface variable of a visualization.

Requirement: The referenced visualization must be used as a dialog.

Syntax:

```
{attribute 'VAR_IN_OUT_AS_POINTER'}
```

> **NOTICE!**
> Uppercase and lowercase characters must be maintained.

Example: Declaration of an interface

```
VAR_IN_OUT
    {attribute 'VAR_IN_OUT_AS_POINTER'}
    itfController : ControlFB;
END_VAR
```

See also

- *Chapter 1.4.5.15.4 “Calling a Dialog with an Interface” on page 1343*
- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*

1.4.5.18.5 Attribute 'parameterstringof'

The pragma `{attribute 'parameterstringof'}` allows that the instance name of the specified parameter is made accessible for the referenced visualization. An interface variable (`STRING`) will contain the instance name of the specified parameter. The interface variable is visible within the referenced visualization and can for example be used in a text output.

Syntax:

{attribute 'parameterstringof' := '<variable>'}

Example: declaration of a interface

```
VAR_INPUT
    {attribute 'parameterstringof' := 'iftDut_A'}
    sItfNameDut_A: STRING;
END_VAR
VAR_IN_OUT
     iftDut_A : DUT_A;
END_VAR
```

See also

- *Chapter 1.4.5.15.2 “Calling a Visualization with an Interface” on page 1332*
- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*

1.4.5.19 Reference, user interface

1.4.5.19.1 Keyboard Shortcuts for Default Keyboard Action

Requirement: The *“Activate default keyboard handling”* option is activated in the *“Visualization Manager”* object.

The keyboard shortcuts for default keyboard action make it possible for users to operate the visualization with the keyboard only. Elements that respond to user input can process a keyboard event instead of a mouse event. You do **not** have to change their input configuration for this purpose. The universal keyboard shortcuts are supported by all devices and are available on all display variants when needed.

Keyboard shortcuts	
[Tab]	Focus jumps to the next element. The next element that responds to a configured or preconfigured user input receives the focus. The order of elements corresponds to the order that the elements were added to the editor. If the focused element is a table, then the upper left cell in the table is the next focus. After that, each next cell until all cells have been focused. It also applies here that only cells that require input are focused. If the focused element is a frame, then an element of the referenced visualization is set next in focus in the frame. After that, each next element until all elements have been focused. It also applies here that only elements that require input are focused.
[Shift]+[Tab]	Focus jumps to the previous element. The element is focused that is before the currently focused element in the added order. Therefore, the order is the opposite as for *“Tab”*.
[Arrow]	The focus jumps to the element that is in the direction as indicated by the arrow.
[Input]	The visualization detects the input at the focused element and triggers the input action.

1.4.5.19.2 Commands

Command 'Interface Editor'

Symbol: ; keyboard shortcut: *[Alt]+[F6]*.

Function: The command opens and closes the *"Interface Editor"* tab above the visualization editor.

Call: Menu bar: *"Visualization ➔ Interface Editor"* Also by clicking on the small down arrow at the top of the visualization editor

Tab 'Interface Editor'

Symbol:

The tab contains an editor for the declaration of interface variables. The editor behaves in a similar way to the declaration editor of a function block, however interface variables are not initialized.

Syntax

```
<scope>
    ( {attribute '<attribute name>' ( := '<expression>' )? } )?
    <identifier> : <data type>;
END_VAR

<scope> : VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT
// (...)? : Optional
```

Example

Declaration in the interface editor

```
VAR_INPUT
    {attribute 'parameterstringof'}
    sIdentifier : STRING; // String for instance name
    iCounter : INT;
END_VAR
VAR_IN_OUT
    {attribute 'VAR_IN_OUT_AS_POINTER'}
    fbController: FB_Controller;
END_VAR
```

Scopes

Possible scopes for interfaces of visualizations or dialogs

VAR_IN_OUT	• When transferring a structure When the visualization is instanced, it gets a reference to the current application data. • When transferring a control variable, if the variable is written to when a user input is made. Only then can the visualization write to it. Note: In the case of dialogs, the data is written back only when the dialog is closed. Hint: We strongly recommend that you use this scope so that the return of values is possible. Moreover, no data needs to be copied.
VAR_IN_OUT Pragma `{attribute 'VAR_IN_OUT_AS_POINTER' }`	When transferring a pointer to a data object In contrast to the VAR_IN_OUT scope (without an attribute), the variable changes are effective immediately and not just when the dialog is closed. Note: Use this scope only if the visualization implements a **Dialog**.

VAR_INPUT	When transferring data that will only be read Note: ● If the visualization is executed as an integrated visualization, then only input variables of a basic data type (scalar type) are permitted to be transferred. ● If the visualization is executed as a CODESYS TargetVisu or a CODESYS WebVisu, then input variables of any data type (including POUs) can also be transferred.
VAR_INPUT Pragma `{attribute 'parameterstringof'}`	When transferring a variable (data type `STRING`) for the instance name of the transfer parameter specified in the attribute

See also

- *Chapter 1.4.1.8.2 “Declaration of Variables ” on page 222*
- *Chapter 1.4.1.19.1.1 “Declaration Editor” on page 461*
- *Chapter 1.4.5.15.2 “Calling a Visualization with an Interface” on page 1332*
- *Chapter 1.4.5.15.4 “Calling a Dialog with an Interface” on page 1343*
- *Chapter 1.4.5.18.5 “Attribute 'parameterstringof'” on page 1717*
- *Chapter 1.4.5.18.4 “Attribute 'VAR_IN_OUT_AS_POINTER'” on page 1716*

Command 'Keyboard Configuration'

Symbol: ; keyboard shortcut: *[Alt]+[F6]*.

Function: This command opens and closes the *“Keyboard Configuration”* tab above the visualization editor.

Call: Menu bar: *“Visualization”*.

Requirement: A visualization is open and active in the visualization editor.

See also

- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Tab 'Keyboard configuration'

Symbol:

This tab contains a list of keyboard shortcuts with an editing option.

A keyboard shortcut can refer specifically to an element. Then the configuration appears here and in the *“Input configuration”* property of the associated element.

A keyboard shortcut can also have several configurations. If a keyboard shortcut has multiple keyboard configurations, then its input actions are executed in the order listed here.

Keyboard shortcuts of the default keyboard action are **not** listed here.

“Key”	Key that a keyboard configuration is defined. Example: *[M]* Note: You can combine the key with *[Ctrl]*, *[Alt]*, and/or *[Shift]*.
“Key down”	☑: The input action is executed when the user presses the key. ☐: The input action is executed when the user releases the key. Double-click: Drop-down list of all keys. Note: If the input action should be executed for both pressing the key (`KeyDown`) and releasing the key (`KeyUp`), then you must define a keyboard configuration for both input actions.
“Shift”	☑: The input event is triggered for *[Shift]+[key]*.

“Ctrl”	☑: The input event is triggered for *[Ctrl]+[key]*.
“Alt”	☑: The input event is triggered for *[Alt]+[key]*.
“Action type”	input action Double-click: Drop-down list of input actions. Tip: For a description of input actions, refer to the *“Input configuration”* dialog box.
“Action”	Configuration of the input action that was selected next. Double-click: A dialog box opens that varies according to the input action. It allows the user-prompted customization of the settings. Tip: For a description of dialog boxes, refer to the *“Input configuration”* dialog box. The input action is configured in the same way here.
“Element ID”	ID of the visualization element where the user can execute the key event. The ID is relevant only if the event is also assigned to an element. Tip: The assignment of ID to element name is listed in the *“Element list”*.
“Access rights”	Access privileges of the action per user group Requirement: The visualization has a user management.

↓	Clicking the symbol on the right of the list moves the selected row one line down.
↑	Clicking the symbol on the right of the list moves the selected row one line up.
Blank line	Allows adding a new keyboard configuration.

See also

- *Chapter 1.4.5.19.2.3 “Command 'Visualization Element List'” on page 1721*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Command 'Visualization Element List'

Symbol: 

Function: The command opens the *“Visualization Element List”* tab for the current visualization. It is displayed in the upper part of the visualization editor.

Call: Menu bar: *“Visualization”*

Requirement: A visualization is open in the editor.

Tab 'Visualization Element List'

This view contains a list of the visualization elements in the open visualization. Grouped elements are displayed in a tree structure and have their own order within the group (other hierarchy level).

The current selection in the list is always synchronized with the selection in the main window of the editor.

The order in the element list from top to bottom describes the order of the elements on the display layers of the visualization from back to front. When you insert elements consecutively, they are arranged starting from the back (position 0) on one layer forward. When you use the commands in the menu *“Visualization ➔ Order”* to move an element from front to back in the editor window, the element list refreshes accordingly.

"Type"	Element type and symbol, as used in the *"Visualization Toolbox"* view, as well as the element number that specifies the display layer. `#0` = layer furthest back.
"X",	Position of the upper left corner of the element (0,0 = upper left corner of the visualization area).
"Y"	
"Width"	Dimensions of the element (in pixels).
"Height"	
"ID"	Internally assigned element identifier
"Name"	Element name as defined in *"Properties ➔ Element name"*.
"Access Rights"	The lock symbol indicates the restricted behavior of an element for some user groups.
"Tab Order"	Position within the order in which you can jump from element to element in the editor by means of the tab key when the default keyboard usage is activated. The activation is done in the visualization manager, on the settings tab. Note that elements within a group or group box have their own order (different hierarchy level). The tab order initially corresponds to the order in which the elements are arranged on the layers from back to front (*"Type"* above). To change the position in the order for an element, you can specify a different number directly in the table field. You can also use the *"Move to Position"* context menu command to open a dialog for specifying a new position. Bold fonts indicate changed position specifications. By removing the displayed value, you exclude the element from the selection using tab or arrow keys. You can use the *"Reset to Default"* context menu command to reset a changed position to the original position. This can be done simultaneously for a multiselection of elements when they do not belong to different hierarchy levels (groupings).

See also

- *Chapter 1.4.5.19.4.1.1 "Visualization Editor" on page 1772*
- *Chapter 1.4.5.19.2.5 "Command 'Order'" on page 1723*
- *"Moving the visualization element forward and back" on page 1257*

Command 'Activate Keyboard Usage'

Symbol:

Function: This command activates and deactivates the keyboard usage when a visualization is executed in online mode (integrated in CODESYS).

Call: Menu bar: *"Visualization"*; context menu.

Requirement: A visualization is open.

When this command is active, the visualization executes the keyboard events that you specified as a visualization user.

When the command is inactive, CODESYS executes the keyboard events that you specify.

See also

- *Chapter 1.4.5.4.4 "Configuring Keyboard Shortcuts" on page 1274*

Command 'Order'

Function: The command makes further commands available. They are for specifying the order of the elements in levels, since elements in the rear levels are concealed by those in the front levels.

Call: Menu *"Visualization"*, context menu

Requirement: The visualization elements are positioned behind one another.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Command 'Bring to Front'

Symbol: 

Function: The command positions the selected visualization element in the front level. The element becomes completely visible.

Call: Menu *"Visualization ➔ Order"*, context menu

Command 'Bring One to Front'

Symbol: 

Function: The command positions the selected visualization element one level further forwards.

Call: Menu *"Visualization ➔ Order"*, context menu

Command 'Send to Back'

Symbol: 

Function: The command positions the selected visualization element in the back level.

Call: Menu *"Visualization ➔ Order"*, context menu

Command 'Send One to Back'

Symbol: 

Function: The command positions the selected visualization element one level further backwards.

Call: Menu *"Visualization ➔ Order"*, context menu

Command 'Alignment'

Function: the command makes further commands available. It is used for the alignment of visualization elements in the window area of the visualization.

Call: Menu *"Visualization"*, context menu

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Command 'Align Left'

Symbol: 

Function: the command aligns the selected visualization elements along a line through the left-hand edge of the element that is positioned furthest left.

Call: Menu *"Visualization ➔ Alignment"*, context menu

Requirement: Several elements are selected.

Command 'Align Top'

Symbol: 

Function: the command aligns the selected visualization elements along a line through the upper edge of the element that is positioned highest.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Align Right'

Symbol: 

Function: the command aligns the selected visualization elements along a line through the right-hand edge of the element that is positioned furthest right.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Align Bottom'

Symbol: 

Function: the command aligns the selected visualization elements along a line through the lower edge of the element that is positioned lowest.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Align Vertical Center'

Symbol: 

Function: the command aligns the selected visualization elements to their common vertical center.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Align Horizontal Center'

Symbol: 

Function: The command aligns the selected visualization elements to their common horizontal center.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Make Horizontal Spacing Equal'

Symbol: 

Function: The command aligns the selected visualization elements so that the elements positioned furthest left and furthest right retain their position and the elements between them are positioned with the same horizontal spacing.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: 3 or more elements are selected. The first element is blue, while the other elements are displayed in grey.

Command 'Increase Horizontal Spacing'

Symbol: 

Function: The command aligns the selected visualization elements so that the blue element retains its position and the other elements are positioned with a larger horizontal spacing. The spacing increases by 1 pixel each time.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Decrease Horizontal Spacing'

Symbol: 

Function: The command aligns the selected visualization elements so that the blue element retains its position and the other elements are positioned with a smaller horizontal spacing. The spacing decreases by 1 pixel each time.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Remove Horizontal Spacing'

Symbol:

Function: The command aligns the selected visualization elements so that the blue element retains its position and the other elements are positioned with no horizontal spacing between them.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Make Vertical Spacing Equal'

Symbol:

Function: The command aligns the selected visualization elements so that the uppermost and lowermost elements retain their position and the elements between them are positioned with the same vertical spacing.

Call: Menu *“Visualization”*, context menu

Requirement: 3 or more elements are selected. The first element is blue, while the other elements are displayed in grey.

Command 'Increase Vertical Spacing'

Symbol:

Function: The command aligns the selected visualization elements so that the blue element retains its position and the other elements are positioned with a larger vertical spacing. The spacing increases by 1 pixel each time.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Decrease Vertical Spacing'

Symbol:

Function: The command aligns the selected visualization elements so that the blue element retains its position and the other elements are positioned with a smaller vertical spacing. The spacing decreases by 1 pixel each time.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Remove Vertical Spacing'

Symbol:

Function: The command aligns the selected visualization elements so that the blue element retains its position and the other elements are positioned with no horizontal spacing between them.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected.

Command 'Make Same Width'

Symbol:

Function: The command makes the width of the selected visualization elements the same as the width of the blue selected element.

Call: Menu *“Visualization ➔ Alignment”*, context menu

Requirement: Several elements are selected. The first element is blue, while the other elements are displayed in grey.

> *The command does not work with lines or polygons.*

Command 'Make Same Height'

Symbol: 

Function: The command makes the height of the selected visualization elements the same as the height of the blue selected element.

Call: Menu *"Visualization ➔ Alignment"*, context menu

Requirement: Several elements are selected. The first element is blue, while the other elements are displayed in grey.

> *The command does not work with lines or polygons.*

Command 'Make Same Size'

Symbol: 

Function: The command makes the size of the selected visualization elements the same as the size of the blue selected element.

Call: Menu *"Visualization ➔ Alignment"*, context menu

Requirement: Several elements are selected. The first element is blue, while the other elements are displayed in grey.

> *The command does not work with lines or polygons.*

Command 'Size to Grid'

Symbol: 

Function: The command aligns the size and position of the selected visualization elements to the grid.

Call: Menu *"Visualization ➔ Alignment"*, context menu

Requirement: Several elements are selected.

> *The command does not work with lines or polygons.*

Command 'Group'

Symbol: 

Function: The command groups the selected visualization elements and displays them as one.

Call: Menu *"Visualization"*, context menu

Requirement: At least 2 elements are selected.

To select more elements you can drag a window around the desired elements with the mouse. Alternatively you can click on the desired elements while keeping the *[Shift]* key pressed.

To select all elements you can open the context menu of the visualization editor and choose the *"Select All"* command.

You can also drag and drop elements to a group. For that, press the [Shift] key while dragging the element to the group. Meanwhile the cursor changes its appearance (display a small plus sign).

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*
- *Chapter 1.4.5.19.2.8 "Command 'Ungroup'" on page 1727*
- *Chapter 1.4.5.19.2.22 "Command 'Select None'" on page 1744*

Command 'Ungroup'

Symbol: 

Function: The command ungroups elements again.

Call: Menu *"Visualization"*, context menu

Requirement: A grouping is selected.

See also

- *Chapter 1.4.5.19.2.7 "Command 'Group'" on page 1726*

Command 'Frame Selection'

Function: The command opens the *"Frame Configuration"* dialog.

Call:

- Menu bar: *"Visualization"*
- Click the *"Configure"* button in the *"References"* property.

Requirement: A *"Frame"* element or *"Tabs"* element is selected in the editor. The *"Element Properties"* view is open.

Dialog 'Frame Configuration'

The dialog allows you to select one or more of all available visualizations. The selected visualizations are displayed at runtime in the window area of the *"Frame"* element or *"Tabs"* element.

NOTICE!

Visualizations can be nested at any depth by means of *"Frame"* elements. In order to use the *"Switch to any visualization"* frame selection type without any problems, a *"Frame"* must not contain more than 21 referenced visualizations.

For more information, see also the description for the *"Input configuration"* of an element: Action *"Switch frame visualization"*.

Table 297: “Available Visualizations”

“By Visualization Name”	The list of available visualizations of the project and libraries is sorted alphabetically.
“By Type or Instance”	The list of available visualizations of the project and libraries is sorted by type or instance.
Input field for a filter	If a filter text is specified, then only those visualizations whose names contain the filter text are listed.
	Project with project visualizations below it
{}	Library with project visualizations below it

Table 298: “Selected Visualizations”

 *“Add”*	Click the symbol to add a visualization to the list of selected visualizations. Requirement: This is selected in *“Available Visualizations”*. Hint: To add a visualization, double-click a visualization in *“Available Visualizations”*.
 *“Delete”*	Click the symbol to delete a visualization from the list. Requirement: This is selected in *“Selected Visualizations”*.
The visualizations are automatically numerically indexed via the order in the list. The top visualization has the index `0`. The next visualization has the index `1` and so on. Note: A *“Frame”* and a *“Tabs”* element use the variables specified in the index of the *“Switch frame variable”* property.	
↑ *“Move Up”*	Click the symbol to move a visualization up in the list. Requirement: This is selected in *“Selected Visualizations”*.
↓ *“Move Down”*	Click the symbol to move a visualization down in the list. Requirement: This is selected in *“Selected Visualizations”*.

See also

- *Chapter 1.4.5.19.5.6 “Visualization Element 'Frame'” on page 1856*
- *Chapter 1.4.5.19.5.10 “Visualization Element 'Tabs'” on page 1887*
- *“Element property 'Switch frame variable'” on page 1671*

Command 'Background'

Symbol: 

Function: The dialog *“Background”* opens. You can define here whether the background of the visualization is colored or displayed with an image.

Call: Menu *“Visualization”*, context menu

See also

- *Chapter 1.4.5.3.7 “Designing a background” on page 1266*

Dialog 'Background'

Table 299: “Color Settings”

“Use Color”	☑: Background in color Color defined as a style color or as a fixed value.

Table 300: “Image Setting”

“Use Image”	☑: Display of a background image Reference to an image from an image pool in the project, formally specified as an instance path: `<Name of the image pool>.<ID>` Example: ● `ImagePool_A.Factory` ● `ImagePool_B.ID_B`

Command 'Multiply Visu Element'

Symbol: 

Function: The command opens the *“Multiply Visu Element”* dialog, which contains a configuration derived from the template element and the array declaration. You can rearrange the elements here, as well as their quantity and the index access to the array data. When you exit the dialog, a field of similar elements is created from the template element. In the properties of the new elements, array variables are now configured with precise array indexes. These new elements are those in which you have configured an array variable with index access placeholders in the template.

Call: Menu bar: *“Visualization”*; context menu

Requirement: The visualization is active and a configured template element is selected.

Dialog 'Multiply Visu Element'

Table 301: Tab “Basic Settings”

“Total number of elements”	The total number is determined by the index range of the placeholders, including the setting on the *“Advanced Settings”* tab. The layout of the elements can be one-dimensional (as a column or row) or two-dimensional (as a table field).
“Horizontal”	Number of elements per row Default: Number of array components (index range) of the placeholder `$FIRSTDIM$` Example for array: `axLampIsOn: ARRAY[0..4] OF BOOL; = 5`
“Vertical”	Number of rows required for the layout of all elements Default ● When using index access placeholder `$FIRSTDIM$`: If the index range of the placeholder is less than five, then the layout of elements is horizontal. If the index range is greater than five, then the layout the elements is quadratic whenever possible. ● When using index access placeholders `$FIRSTDIM$` and `$SECONDDIM$`: The number of horizontal elements is equal to the number of index ranges specified by the placeholder `$FIRSTDIM$`. The number of vertical elements is equal to the number of index ranges specified by the placeholder `$SECONDDIM$`.
“Offset between elements”	Distance between the new elements; affects the positions of the new elements ● *“ 0 ”*: The frames of the elements overlap by one pixel. ● *“1 ”*: The elements touch. ● *“<n> ”*: A distance of n-1 pixel is visible between the elements.
“Horizontal”	Distance between the elements within a row (in pixels) Example: `2` for a distance of one pixel

“Vertical”	Distance between the elements within the columns (in pixels) Example for a distance of three pixels: `4`
“Arrangement of elements”	Origin from which the new elements are positioned and arranged If *“Vertical”* or *“Horizontal”* <> 1 ● *“From top left”* ● *“From top right”* ● *“From bottom left”* ● *“From bottom right”* If *“Horizontal”* or *“Vertical”* = 1 ● *“From top”* ● *“From bottom”*
“Orientation”	Determines the layout of the elements in the field (row by row, or column by column) ● *“Line by line”* ● *“Column by column”*
“Preview”	Displays the set layout and orientation of the elements as an arrow

Table 302: Tab “Advanced Settings”

“Array access”	Based on the template element, the precise index for accessing the array variable is calculated for each new element. The calculation is based on the array index limits as specified in the array declaration. The settings are also taken into account here.
“1st dimension”	Calculation guideline for the index of the first dimension that replaces `$FIRSTDIM$` The first new element gets the value specified below in *“Start index”* in the first dimension. The other elements each get an index incremented by *“Increment”* until an index is calculated for all elements. Example ● *“Start index”*: `1` ● *“Increment”*: `1`
“2nd dimension”	Calculation guideline for the index of the second dimension that replaces `$SECONDDIM$` The first new element gets the value specified below in *“Start index”* in the second dimension. The other elements each get an index incremented by *“Increment”*. Example ● *“Start index”*: `1` ● *“Increment”*: `1`

“OK”	First, it is validated whether the calculated indices are in the index range of the array variable. If so, then the elements that match the template element are created and arranged as a field (row, column, or table). The placeholder indexes are replaced by the calculated indexes.

Example

Declaration of array variables

```
VAR
asTexts_Example: ARRAY[1..2,1..2] OF STRING :=
    [
        '1A Text', '2A Text',
        '1B Text', '2B Text'
    ];
    asToolTips_Example: ARRAY[1..2,1..2] OF STRING :=
     [
        '1A Tooltip', '2A Tooltip',
        '1B Tooltip', '2B Tooltip'
    ];

    axUserInput_Example: ARRAY[1..2,1..2] OF BOOL;
END_VAR
```

Visualization with template element and its property configuration

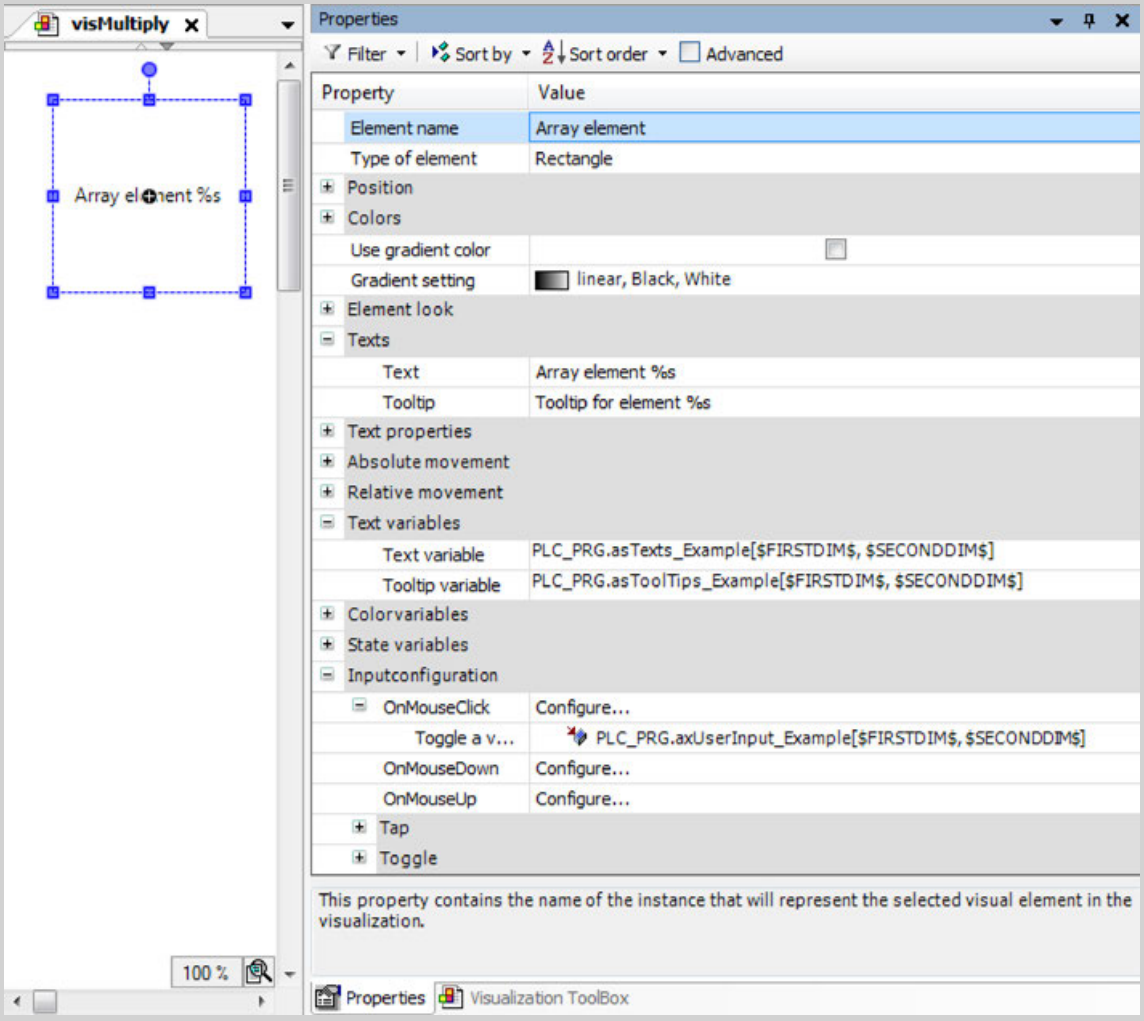


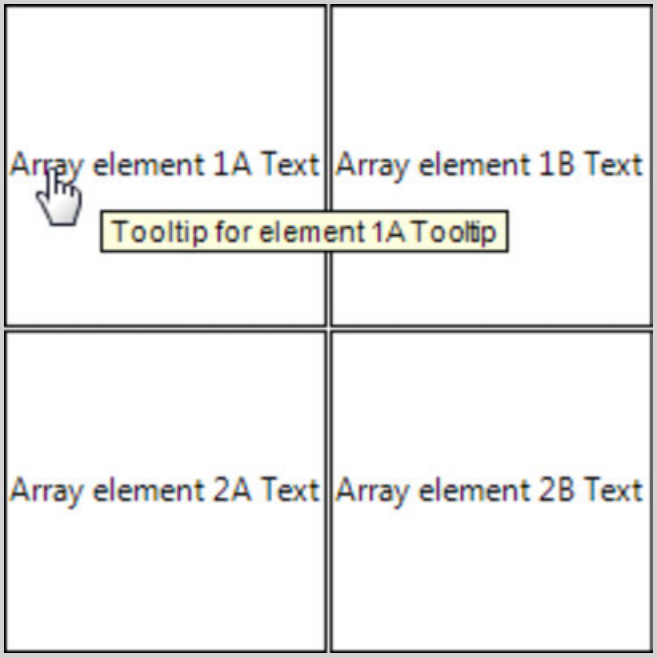
Table 303: Dialog 'Multiply Visu Element'

Tab *“Basic Settings”*	
“Total number of elements”	
“Horizontal”	2
“Vertical”	2

“Offset between elements”	
“Horizontal”	2
“Vertical”	2
“Arrangement of elements”	*“From top left”*
“Orientation”	*“Line by line”*

Tab *“Advanced Settings”*	
“Array access”	
“1st dimension”	
“Start index”	1
“Increment”	1
“2nd dimension”	
“Start index”	1
“Increment”	1

Visualization at runtime:



See also

- *Chapter 1.4.5.9.2 “Configuring and Multiplying Visualization Elements as Templates” on page 1299*

Command 'Configure Display Settings of Trend'

Symbol:

Function: When you execute this command in *“Visualization”* or in the context menu, the *“Edit Display Settings”* dialog opens.

Call:

- Menu bar: *“Visualization”*
- Context menu of a *“Trend”* element in the visualization editor
- Property *“Diagram”*

Requirement: A trend is selected in the active visualization editor.

Tab *“X Axis”*

“Grid”	☑: Trend diagram with grid lines in the X-direction in the selected color
“Font”	Font for the axis label

Tab *“Y axis”*

Table 304: “Display mode”

• *“Auto”*:	◉: The visualization scales automatically.
• *“ Fixed”*	◉: Fixed range from *“Minimum”* to *“Maximum”*
“Minimum”	Literal, variable (integer data type), or constant variable (integer data type). It contains the initial value of the segment. Requirement: The *“Display Mode”* is *“Fixed”*. Examples: `20`,`PLC_PRG.iLimit_Min`, `GVL.c_iLimit_Min` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Min : INT := 20`
“Maximum”	Literal, variable (integer data type), or constant variable (integer data type). It contains the end value of the segment. Requirement: The *“Display Mode”* is *“Fixed”*. Examples: `80`,`PLC_PRG.iLimit_Max`, `GVL.c_iLimit_Max` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Max : INT := 80`
“Grid”	☑: Trend diagram with grid lines in the Y-direction in the selected color
“Description”	☑: Text for labeling the Y-axis (for example, `DC/mA`)

Table 305: “Tick marks”

“Fixed spacing”	☑: Axis scale with tick marks for *“Distance”* and *“Subdivisions”*
“Distance”	Distance between the tick marks (example: `2`)
“Subdivisions”	Number of subdivisions between tick marks (example: `4`)

“Font”	Font for the axis label

Table 306: “Background”

“From visualization style”	Background color as defined in the visualization style
“Draw background”	Background color which is selected in the lower input field
“No background”	Trend diagram with transparent background
Background color of the trend diagram	Requirement: *“Draw background”* is activated.

"Reset"	Resets the settings to the default settings
"Use as default"	Saves the settings as default

"Add Y-axis"	Extends the trend diagram by one Y-axis Result: The *"Trend Recording"* editor contains an extended selection of Y-axes in the *"Additional axes"* option of the *"Variable Settings"*.
"Delete Y-axis"	Deletes the Y-axis of the visible tab.

See also

- *Chapter 1.4.5.19.2.16 "Command 'Configure Display Settings of Trend'" on page 1738*
- *Editor 'Trend Recording'*

Command 'Configure Trace'

Symbol:

Function: This command opens the *"Trace Configuration"* dialog box.

Call: Context menu of the visualization element; *"Trace"* property of the visualization element.

Requirement: An element of type *"Trace"* is open in the editor.

Dialog box 'Trace Configuration'

The tree view shows the trace configuration and allows navigation.

The top entry contains the trace name. When this entry is selected, the *"Record Settings"* group appears in the adjacent view.

An entry is located here for each variable that data was recorded continuously. When a variable is selected, the *"Variable Settings"* group appears in the adjacent view.

"Add variable"	Adds a new entry to the trace configuration. Result: A blank configuration appears next to the new variable under *"Variable Settings"*. You configure the variable there.
"Delete variable"	Removes the selected variable.

'Recording Settings'

A trigger can be configured in the trace only.	
"Task"	Task where data was recorded.
"Record condition"	Recording condition for which the application records data in runtime mode: Variable (BOOL)
"Comment"	Example: `Acquiring data only when all conditions are true.`
"Resolution"	Measure for the time stamp that is recorded per data set. ● *"ms"*: Time stamp (in milliseconds). ● *"µs"*: Time stamp (in microseconds) for a task cycle time of 1 ms or less
"Automatic restart"	☑: Recording starts automatically as soon as the trace has been started one time and then the controller was restarted. The trace configuration and the contents of the trace buffer are saved persistently to a file on the target system. Format: `.trace.csv`
"Display"	The *"Edit Appearance"* dialog box opens.

"Advanced"	The *"Advanced Trace Settings"* dialog box opens.
"Copy from Trace"	The *"Copy Settings from Trace Instance"* dialog box opens. If you have already created an existing trace configuration from a trace object, then you can copy the configuration data to the visualization element. To do this, select the respective object.

See also

- *Chapter 1.4.5.19.3.19 "Dialog 'Display Settings'" on page 1770*
- *Chapter 1.4.5.19.3.18 "Dialog 'Advanced Trace Settings'" on page 1770*
- *Chapter 1.4.5.18.1.34 "Visualization Element 'Trace'" on page 1619*

'Variable Settings'

"Variable"	Variable for recorded value. ● Variable (valid data type) ● property ● Reference ● Contents of the pointer ● Array element (base type with valid data type) ● Enumeration (base type with valid data type) Valid data types are all standard types, **except** `STRING`, `WSTRING`, and `ARRAY`.
"Parameters"	Parameter whose value is acquired. ...: Input assistance lists ale valid parameters of the PLC.
	Enables toggling between *"Variable"* and *"Parameter"*
"Attached axis"	Y-axis of the trace diagram for the *"Variable"*. : Selection of the standard y-axis and the additional configured y-axes Note: The additional configured y-axes are configured in the *"Edit Display Settings"* dialog box.
"Display variable name"	: The trace graphs are displayed in tooltip with their variable names. If a text is also specified in *"Description"*, then the text is displayed first with the variable names in parentheses. Example: `Sensor A (PLC_PRG.iSensor_A)` If *"Description"* does not contain any text, then the *"Display Variable Name"* property is activated automatically. Then only the name is displayed (example: `PLC_PRG.iSensor_A`). : The trace graphs are displayed in tooltip without their variable names. Only the text in *"Description"* is displayed.
"Description"	Text for the tooltip. It is displayed when a visualization user moves the cursor in the trace diagram. Example: `Sensor A` The text is also entered into the *"GlobalTextList"* object and can be translated there.
"Color"	Color of the graph in the diagram.
"Line type"	Representation of the graph as a line chart ● *"Line"*: Values are linked to form a line. ● *"Step"*: Values are linked in the form of steps. ● *"None"*: Values are not linked.

"Line width"	In pixels Example: `1`
"Line style"	The display of the line is solid, dash, dot, dash-dot, or dash-dot-dot.
"Dot type"	Representation of the graph as a scatter chart. This configuration entry with the *"Line type"* determines the appearance of the graph. ● *"Dot"*: Each value as a dot. ● *"Cross"*: Each value as a cross. ● *"None"* Note: For *"Dot"* or *"Cross"*, a paint buffer overflow can result from many recorded variables.
"Warning at minimum"	☑: When below the lower limit, the visualization shows the trace graphs in the alert color.
"Critical lower limit"	Minimum Value Example: `10.`
"Color"	Warning color on falling below the limit
"Warning at maximum"	☑: When above the upper limit, the visualization shows the trace graphs in the alert color.
"Critical upper limit"	Maximum value Example: `90`
"Color"	Warning color on exceeding the limit
"Dynamic appearance options"	
"Variable for visibility"	Variable (`BOOL`) or as bit access. This controls the visibility of the variables in the trace diagram. ● `TRUE`: Visible ● `FALSE`: Invisible

See also

- *Chapter 1.4.5.19.2.13 "Command 'Configure Trace'" on page 1734*

Command 'Export Trace Configuration'

Function: This command opens the *"Export Trace Configuration"* dialog box.

Call: context menu (right-click) the upper node in the tree view of the trace configuration.

Requirement: The dialog box *"Trace Configuration"* is active and the name of the trace configuration is selected in the tree view (example: `Visu_Trace1`).

Dialog box 'Export Trace Configuration'

This dialog is used for saving the trace configuration to a text file that can be read by the runtime system.

"File name"	Name of text file to be created.
"File type"	*"Trace file (*.trace)"*: Format that the runtime system component `CmpTraceMgr` expects for reading.

See also

- *Chapter 1.4.5.19.2.13 “Command 'Configure Trace'” on page 1734*

Command 'Insert Elements for Controlling Trace'

Symbol: 

Function: The command opens the *“Trace Wizard”* dialog. In this dialog, you select predefined visualization elements for controlling the trace recording. These elements are then inserted as configured into the visualization editor.

Call: Menu bar: *“Visualization”*; context menu of the trace element.

Requirement: The view is active and a trace element is selected.

Dialog 'Trace wizard'

“Control variable”	Corresponds to the *“Control variables”* property that is available in the element properties of the trace element. ☑: The control element for this trace control variable is created in the visualization editor.
“Variable”	Project variables that are assigned to the control element below the *“Input configuration”* property. In addition, the project variables are declared as local variables in the visualization when needed (in the interface editor). This list corresponds to the assignments that are defined in the element properties of the trace element. If nothing is configured in the properties of the trace element (no project variables assigned as control variables), then a pre-allocation is offered with default variable names.
“Type of element to insert”	For a Boolean variable, this element can be inserted as a button or rectangle. For a string variable, a rectangle or a text field is provided.
“OK”	At the closing of the dialog, the selected control elements are inserted into the visualization editor and (when needed) its control variables are created as local variables of the visualization. They are declared in the interface editor and they are used by the control element (property *“Input configuration ➔ Toggle ➔ Variable”*) and by the trace element (*“Control variables”* property). The control element writes to the variable and the trace element reads the variable.

Example

Standard control variables:

```
VAR
    bResetTrigger : BOOL;
    bStart : BOOL;
    bStop : BOOL;
    bStore : BOOL;
    sStoreFilename : STRING;
    bRestore : BOOL;
    sRestoreFilename : STRING;
END_VAR
```

See also

- *Chapter 1.4.5.10.1 “Getting started with trace” on page 1307*

Command 'Configure Display Settings of Trend'

Symbol: 

Function: When you execute this command in *"Visualization"* or in the context menu, the *"Edit Display Settings"* dialog opens.

Call:

- Menu bar: *"Visualization"*
- Context menu of a *"Trend"* element in the visualization editor
- Property *"Diagram"*

Requirement: A trend is selected in the active visualization editor.

Tab *"X Axis"*

"Grid"	☑: Trend diagram with grid lines in the X-direction in the selected color
"Font"	Font for the axis label

Tab *"Y axis"*

Table 307: "Display mode"

• *"Auto"*:	◉: The visualization scales automatically.
• *" Fixed"*	◉: Fixed range from *"Minimum"* to *"Maximum"*
"Minimum"	Literal, variable (integer data type), or constant variable (integer data type). It contains the initial value of the segment. Requirement: The *"Display Mode"* is *"Fixed"*. Examples: `20`,`PLC_PRG.iLimit_Min`, `GVL.c_iLimit_Min` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Min : INT := 20`
"Maximum"	Literal, variable (integer data type), or constant variable (integer data type). It contains the end value of the segment. Requirement: The *"Display Mode"* is *"Fixed"*. Examples: `80`,`PLC_PRG.iLimit_Max`, `GVL.c_iLimit_Max` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Max : INT := 80`
"Grid"	☑: Trend diagram with grid lines in the Y-direction in the selected color
"Description"	☑: Text for labeling the Y-axis (for example, `DC/mA`)

Table 308: "Tick marks"

"Fixed spacing"	☑: Axis scale with tick marks for *"Distance"* and *"Subdivisions"*
"Distance"	Distance between the tick marks (example: `2`)
"Subdivisions"	Number of subdivisions between tick marks (example: `4`)

"Font"	Font for the axis label

Table 309: "Background"

"From visualization style"	Background color as defined in the visualization style
"Draw background"	Background color which is selected in the lower input field

“No background”	Trend diagram with transparent background
Background color of the trend diagram	Requirement: *“Draw background”* is activated.

“Reset”	Resets the settings to the default settings
“Use as default”	Saves the settings as default

“Add Y-axis”	Extends the trend diagram by one Y-axis Result: The *“Trend Recording”* editor contains an extended selection of Y-axes in the *“Additional axes”* option of the *“Variable Settings”*.
“Delete Y-axis”	Deletes the Y-axis of the visible tab.

See also

- *Chapter 1.4.5.19.2.12 “Command 'Configure Display Settings of Trend'” on page 1732*
- *Editor 'Trend Recording'*

Command 'Edit Trend Recording'

Symbol:

Function: This command opens the *“Trend Recording”* object.

Call:

- Menu bar: *“Visualization”*
- Context menu of a *“Trend”* element in the visualization editor
- Property *“Trend recording”*

Requirement: An element of type *“Trend recording”* is selected in the visualization editor.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.1.20.2.31 “Object 'Trend Recording'” on page 949*

Command 'Insert Elements for Controlling the Trend'

Symbol:

Function: When you execute this command in *“Visualization”*, the *“Trend Wizard”* dialog box opens.

Call: Menu bar: *“Visualization”*; context menu of a *“Trend”* element in the visualization editor.

Requirement: A trend is selected in the active visualization editor.

Dialog 'Trend wizard'

Each row of the table contains a control element that can assigned to the trend. The elements are placed in the visualization next to the trend. The control elements are saved in the *“Assigned control elements”* property and can be modified there.

“Attached control element”	☑: The associated element is available in the visualization and connected with the trend via the property *“Assigned Visu element”*. The element is inserted into the visualization. ☐: Deactivating the option does **not** cause the element to be deleted from the visualization.
“Position”	Position of the control element in relation to the trend.
“Type of element to insert”	Drop-down list with the installed types of the control element
“Instance name”	Instance name of the control element

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Command 'Visualization Element Repository'

Symbol: 

Function: This command opens the *“Visualization Element Repository”* dialog box opens for editing the storage location and visualization profile.

Call: Menu bar: *“Tools”*.

Requirement: No project is open.

The visualization element repository is used for creating a visualization profile or visualization extension. This is necessary when developing you own visualization elements with the CODESYS VisuElement Toolkit. The CODESYS VisuElement Toolkit is required for this with a valid license. Users who do not wish to create their own visualization elements can use this dialog to find out which elements are included in which visualization profile. A reconfiguration of the storage location for a repository is also important only for element developers.

NOTICE!

1. Only an empty directory can be selected as a new storage location for a repository.

2. The "System" repository cannot be modified. This is indicated by the entry in italics in the repository list.

Currently only a single version of an element can be installed.

Dialog box 'Visualization Element Repository'

Table 310: Editing the repository

“Location”	Storage location for the repository in the file system. The drop-down list contains the configured repositories for visualization elements.
“Edit locations”	Opens the *“Edit Repositories”* dialog box for modifying the repository currently selected in *“Location”* or for creating a new repository.

Table 311: “Profile or extension selection”

A profile is a collection of visualization elements in a specific version. These elements originate from one or more libraries. They are available in the toolbox of the visualization editor when the profile is used in the project. You can use an extension to add a specific selection of elements to an existing profile. Creating and editing a profile is possible only if the CODESYS VisuElement Toolkit is installed. In this case, the buttons on the right side of the dialog box can be used.	
“Create or update profile”	You can configure a new profile in the dialog or modify an existing one. Then, the *“New”*, *“Copy”*, and *“Delete”* buttons are operable, as well as the *“Installed Elements”* and *“Available Elements”* views.
“Create or update extension”	In the dialog, you can configure an extension for the selected profile. Another drop-down list *“Extension”* appears with all currently available extensions. To configure a new extension, use the *“New”* or *“Copy”* buttons (see below). In both cases, the *“Specify Visualization Extension”* dialog box opens for you to define a new extension. In this dialog box, the *“Name”*, *“Company”*, and *“Version”* of the extension are displayed. Version syntax: Sequence of numbers and points with a number at the end.
“Profile”	Currently selected profile. The drop-down list provides all profiles available in the repository set above.
“Extension”	The extension that is currently selected for the specified profile. The drop-down list provides all extensions available for the profile.
“New”	Pressing the button opens the *“Specify Name of Visualization Profile”* dialog or the *“Specify Visualization Extension”* dialog. Specify a unique name for the new profile, or for an extension also the company name and the version. CODESYS automatically enters the previously used name, appended with "_0". The *“Installed elements”* list is empty.
“Copy”	Pressing the button opens the *“Specify Name of Visualization Profile”* dialog or the *“Specify Visualization Extension”* dialog (see above: *“New”*). The elements of the selected profile are accepted and they appear in the *“Installed elements”* view.
“Delete”	The currently set profile or the extension is deleted, and then the drop-down list is removed.

Table 312: “Installed Elements”

“Name, Vendor, Library”	Elements that are assigned to the selected profile.
“Uninstall”	All elements currently selected in the list are uninstalled and removed from the *“Available Elements”* list.
“Update code”	The list is refreshed with any changes in the implementation code of the library POUs.
“Update all”	The list is refreshed with any changes in the implementation code and in the interfaces (declaration part) of the library POUs.

Table 313: “Available Elements”

“Name” *“Library”* *“Vendor”* *“Version”* *“Repositories”* *“Profiles”*	Elements that are available in the system and can be installed into the current profile or extension. The selection depends on the installed element libraries and element packages. The tree structure displays the libraries with the contained elements below them. Elements display in green are already installed for the specified profile or extension. *“Profiles”* shows the elements installed for the profiles.
“Install element”	The elements selected in the list are added to the *“Installed Elements”* view. Existing elements are overwritten.
“Install library”	The *“Library Repository”* dialog box opens where another library can be installed in order to accept its elements in the *“Available Elements”* view.

“Note current library versions only”	☑: When refreshing the list, only the most current version of the library is searched, not all libraries.
“Overwrite profiles without prompting”	☑: For actions that change the profile, the usual prompt does not appear for confirming the change.

Dialog box 'Edit Repositories'

“Storage location, Name”	For managing the visualization elements, one or more repositories can be used. All currently defined storage locations are listed here with file path and name. The order from top to bottom is also the search order for the visualization elements. File path and name of the storage location selected previously in the *“Repository for Visualization Elements”* dialog. Note: A storage location "System" is always defined automatically, which cannot be modified or deleted.
“Add” *“Edit”*	Opens the *“Storage Location for Repository”* dialog for creating a new storage location or for editing the current storage location. Specify: *“Storage location”* (file path of an empty directory) and *“Name”*. The name is symbolic (example: "Elements category 1").
“Remove”	Deletes the repository currently selected in the repository list.
“Move Up, Move Down”	Moves the entries within the list. Note: The repositories are searched from top to bottom.

Command 'Visualization Style Repository'

Symbol: 

Function: This command opens the *“Visualization Styles”* dialog box. It makes it possible to edit visualization style repositories.

Call: Menu bar: *“Tools”*.

See also

- ↳ *Chapter 1.4.5.17.2 “Managing visualization styles in repositories” on page 1365*

Dialog Box 'Visualization Styles'

"Storage location"	Name of the currently selected repository Preset: *"System"* ▾: Lists the repositories installed in the development system.
"(...)"	Storage location of the repository Example: `(C:\ProgramData\CODESYS\Visualization Styles)`
"Edit locations"	The *"Edit Repository Locations"* dialog box opens.

Table 314: "Installed Visualization Styles"

"Company"	When a company name is specified here, the tree view is filtered and only the styles of the selected company are listed. Preset: *"(All companies)"*. It is not filtered. ▾: Lists all companies that are specified in the styles.
Windows with styles	Tree view of all versions of the installed visualization styles in the selected repository
"Display localized names"	☑: The style name is localized and displayed in the language that is set in CODESYS. ☐: The style is display as the source name.
"Install"	The *"Select Visualization Style(s)"* dialog box opens.
"Uninstall"	The selected style version is removed from the repository.
"Preview"	The windows closes. A preview is displayed of the selected style in the selected version. Specific elements are displayed in the style.

Dialog box 'Edit Repository Locations'

Table 315: "Repositories (elements are searched in that order)"

"Location"	Storage location of the configured repository on the development system Example: `C:\ProgramData\CODESYS\Visualization Styles`
"Name"	Preset: `System`
"Add"	The *"Repository Locations"* dialog box opens. It makes it possible to manage other repositories.
"Edit"	
"Remove"	
"Move Up"	The order in the list of repositories is adapted. It defines the processing order when searching for elements.
"Move Down"	

Command 'Add Visual Element'

Function: The command opens a menu containing all available visualization elements as menu items.

Requirement: You have configured the command in the dialog box *"Customize"* in a way that you have a call in a (any) menu.

When you select an element in the menu, the element is added in the visualization editor in the upper left corner.

See also

- *Chapter 1.4.5.3.1 “Select Element” on page 1255*
- *Chapter 1.4.1.20.3.8.16 “Command 'Customize'” on page 1071*

Command 'Select None'

Function: The command cancels at once any selection in the current visualization editor.

Requirement: You have configured the command in the dialog box *“Customize”* in a way that you have a call in a (any) menu.

See also

- *Chapter 1.4.5.3.1 “Select Element” on page 1255*
- *Chapter 1.4.5.19.2.7 “Command 'Group'” on page 1726*
- *Chapter 1.4.1.20.3.8.16 “Command 'Customize'” on page 1071*

Command 'Add Elements for Alarm Acknowledgement'

Symbol: 

Function: This command adds buttons automatically to the visualization for acknowledging alarms. It opens an assistant for inserting controls below the table.

Call: Menu bar: *“Visualization”*; context menu of visualization element "Alarm table"

Requirement: An "Alarm table" visualization element is selected.

Dialog box 'Alarm Table Wizard'

“Type of element(s) to insert”	● *“Button”* ● *“Rectangle”*
“ Action”	☑: A button or a rectangle with the selected function is added to the visualization.
“Variable”	If you have already specified a variable for an action, then this is displayed here in the *“Variable”* column. If you have not defined a variable yet, then a local visualization variable is created automatically.

See also

- *Chapter 1.4.5.19.5.22 “Visualization Element 'Alarm Table'” on page 1969*

1.4.5.19.3 Dialog Boxes

Dialog 'Access Rights'

Function: This dialog defines the permissions of user groups for a visualization element.

Call: Click in the *“Value”* field of the *“Access Rights”* element property of a visualization element.

Requirement: A visualization element is selected in a visualization element and the *“Properties”* is open.

“User Groups”	Groups that were configured in the *“Visualization Manager”* (tab *“User Management ➔ Groups”*).
“Operable”	☑: The visualization element is available with full functionality.
“Only Visible”	☑: The visualization element is visible only and does not provide any functionality.
“Invisible”	☑: The visualization element is not displayed.
“Group hierarchy is used”	Display whether the option *“Use group hierarchy”* is activated in the *“Visualization Manager”* (tab *“User Management ➔ Settings”*). A group of a higher hierarchy cannot have fewer permissions for an element than an element of a lower hierarchy.

If no user is logged in, then the permissions apply for the visualization elements that are configured for the user group “None”. If the permissions for a visualization element is restricted, then the group “None” should be granted the lowest permissions.

See also

- *Chapter 1.4.5.19.4.5 “Tab 'Visualization manager' - 'User management'” on page 1782*

Dialog 'Add Visualization'

Function: The dialog is used to create a new object of type *"Visualization"*.

Call: Menu bar: *"Project* ➔ *Add Object* ➔ *Visualization"*; context menu of an application

Requirement: An application is selected in the device tree.

"Name"	Name of the visualization Example: `Visu_A`

The following settings are displayed only when you add a *"Visualization"* object to the project for the first time.	
"Symbol library"	List of all installed symbol libraries
"Assigned"	: Symbol library is selected Hint: CODESYS manages this setting in the project settings.

"Add"	CODESYS creates a new visualization, assigns the selected symbol libraries to the project, and lists them in the *"Visualization Toolbox"* view.

See also

- *Chapter 1.4.5.3.1 "Select Element" on page 1255*
- *Dialog 'Project Settings' - 'Visualization'*
- *Command 'Add Object'*

Dialog 'Update Frame Parameters'

Function: The dialog requests you, after changing an interface in the visualization references concerned, to re-assign the variables for the parameter transfer.

Call: The dialog appears automatically.

Requirement: You have changed the interface of a visualization, for example by adding an additional variable. After that, you have clicked either *"File* ➔ *Save Project"* or *"Build* ➔ *Generate Code"*, or opened a visualization.

"Parameter"	Hierarchical structure of the interface parameters as a tree view
	Top node of the visualization hierarchy with the name of the visualization. This contains an element of type *"Frame"* or *"Tabs"*.
	Name of the element (*"Frame"* or *"Tabs"* type)
	Name of the referenced visualization
"(Recent)"	Interface of the referenced visualization with the new parameters You can edit the parameter transfer here.
"(Previous)"	Interface of the referenced visualization with the previously valid parameters. You cannot edit the parameter transfer, but you can use it as a template.
`<name>`	Variable for the parameter transfer (`VAR_INPUT` scope)
`<name>`	Variable for the parameter transfer (`VAR_IN_OUT` scope)

"Type"	Data type of the variable Example: `INT`
"Value"	Variable that is transferred as a parameter and with whose value the visualization is initialized during instantiation. Example: `PLC_PRG.iVisNr` If the variable lies under the current interface, which is marked in the tree view with *"(Current)"*, then you can edit the parameter transfer. ● Click in the field to open the input field. ● Double-click in the field to open the Input Assistant. ● Accept the settings by copying assignments in the *"Value"* column and pasting them into another cell. Use the *"Copy"* and *"Paste"* links to do this.

"Copy"	Link for copying an assignment from the *"Value"* column. Requirement: An assignment is selected.
"Paste"	Link for inserting an assignment Requirement: You have copied an assignment.

"OK"	Click the button to close the dialog and confirm the changes made under *"(Recent)"*. Result: The assignment is entered in the *"References"* property and on the *"Interface Editor"* tab.

See also

- *Chapter 1.4.5.19.5.6 "Visualization Element 'Frame'" on page 1856*
- *Chapter 1.4.5.19.2.1 "Command 'Interface Editor'" on page 1719*

Dialog 'Configure Categories and Items'

Function: The dialog is used to manage the categories in a tree view. The assigned elements are listed below a category. You can create custom categories and edit the assignment to the visualization elements. The name of the category is displayed in the *"Visualization Toolbox"* view as a label of the button to open the element selection.

Call: Click the symbol in the *"Visualization Toolbox"* view.

See also

- *Chapter 1.4.5.3.1 "Select Element" on page 1255*
- *Chapter 1.4.5.19.4.1.2 "View 'Visualization Toolbox'" on page 1773*

Tree view

"Category"	Tree view ● *"<name>"*: Default category ● *"<name>"*: Custom category Example: *"Favorite"*
+	Lists the assigned visualization elements. To remove a selected visualization element, click the *[Del]* key. Hint: The assignment is created in the *"Visualization Toolbox"* view with the help of the context menu of a selected element.
"Active"	: A button for the category is visible in the *"Visualization Toolbox"* view.

Toolbar

+	The *"Add Category"* dialog opens.
− or *[Del]*	The category selected in the tree view is removed. After you click *"OK"* to close the dialog, the button is also removed from the *"Visualization Toolbox"* view.

Dialog 'Add Category'

Call: Click the + symbol in the *"Configure Categories and Items"* dialog.

"Name"	Name of the category Example: `tagA`
"Description"	Example: `Tagged with A`

Dialog 'Gradient Editor'

Function: The dialog is for setting the color gradient of visualization elements. If you define two colors, the color graduates between them. If you only select 1 color, the color graduates within this color through its brightness. The detailed settings are for a special specification of the initial position and the angle of the color gradient.

Call: Click in the value field of the property *"Gradient settings"*

Requirement: You have selected a visualization element in the editor that has the property *"Gradient settings"*.

"Gradient type"	● *"Linear"* ● *"Radial"* ● *"Axial"*: The color gradient runs along an axis, with the colors extending perpendicular to the axis on both sides.
"Color 1"	First color of the gradient.
"Color 2"	Second color of the gradient.
"Transparency"	Transparency of the associated color. Permissible values: Integers in the range of values from `255` to `0`. 255: The color is opaque. 0: The color is fully transparent.
"Standard linear"	Requirement: *"Linear"* color gradient. Standard direction of the linear color gradient.
"Standard radial"	Requirement: *"Radial"* color gradient. Standard setting.

“Standard axial”	Requirement: *“Axial”* color gradient. Direction of the color gradient
“Angle (degrees)”:	Requirement: *“Linear”* or *“Axial”* color gradient.
“Center X (%):”	Requirement: *“Radial”* color gradient. X-position of the center point (0 – 100%)
“Center Y (%):”	Requirement: *“Radial”* color gradient. Y-position of the center point (0 – 100%)
“Use one color”	Color gradient between *“Color 1”* and the same color with a different brightness.
“Brightness”	Requirement: The option *“Use one color”* is selected. Setting from 0 (black) to 100 (white)
“Use two colors”	Color gradient between the two selected colors *“Color 1”* and *“Color 2”*.

See also

- *Chapter 1.4.5.3.3 “Assigning a color” on page 1258*
- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Dialog 'Input Configuration'

Symbol: 

Function: The dialog is used to assign input actions to specific input events. It also includes specific settings for the selected input action.

Call: In the *“Input configuration”* property, click *“Configure”*.

Requirement: An element is selected in the editor.

Input action 'User Management'

“Dialogs and actions”	Configures which one of the possible user management dialogs or which action follows the input event Note: The dialog used at runtime is configured in the *“Dialog Settings”* tab of the Visualization Manager. See also ● *“ Login dialog”* ● *“Change password dialog”* ● *“Change configuration dialog”* Default: Dialogs from the `VisuUserManagement` library
“Login”	The login prompt opens. Default: `VisuUserManagement.VUM_Login` in *“Login dialog”*
“Logout”	The current user is logged out.
“Change User Password”	The dialog for changing the password opens. Default: `VisuUserManagement.VUM_ChangePassword` in *“Change password dialog”*
“Open User Configuration”	The dialog opens for changing the configuration. Default: `VisuUserManagement.VUM_UserManagement` in *“Change configuration dialog”*

See also

- *Chapter 1.4.5.5 “Setting Up User Management” on page 1282*
- *“Tab 'Visualization manager' – 'Settings'” on page 1777*

Input action 'Close Dialog'

“Dialog”	The visualization of type *“Dialog”* that will be closed. : List box with all *“Dialog”* type visualizations available in the project. Example: Default dialogs of the `VisuDialogs` library, which is usually integrated in the project. • `FileOpenSave` • `Keypad` • `Login` • `Numpad` • `NumpadExtended` • `TextinputWithLimits` Note: The setting in the object property (*“Visualization”* tab) of a visualization determines whether or not a visualization can be used as a dialog.

“Result”	Return value for closing the dialog. Note: If there are more input actions after closing, then they configured in the *“Input configuration ➔ OnDialogClosed”* property of the element.
“None”	: No return value
“OK”	: The set return value is returned. The return value refers to the button in the dialog. The value `OK` is returned for the OK button. The value `Cancel` is returned for the cancel button.
“Cancel”	
“Abort”	
“Retry”	
“Ignore”	
“Yes”	
“No”	

See also

- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*

Input action 'Open Dialog'

“Dialog”	Visualization (type *“Dialog”*). The dialog opens. : List box with all dialogs available in the project. Note: The *“VisuDialogs”* library provides visualizations (type *“Dialog”*). • `VisuDialogs.FileOpenSave` • `VisuDialogs.Login`

Transfer parameters of the dialog	
"Parameter"	Interface parameter as declared in the interface editor of the visualization Example: `filelistProvider`
"Type"	Data type of the parameter as declared in the interface editor of the visualization. Example: `VISU_FBFILELISTPROVIDER`
"Value"	Variable (data type corresponds to the data type of the parameter). The value of the variable is read when the dialog opens and passed to the parameter. Example: `PLC_PRG.fileListProvider // Instance of function block VisuDialogs.Visu_FbFileListProvider` : The input assistance offers all variables available in the entire project.

Here the return value of the dialog is activated for which the `Var_OUTPUT` variable and `VAR_IN_OUT` variable are written. The dialog closes afterwards.	
"Update" "and" "parameter in case of result"	Note: The parameters are updated before the dialog is closed. Until then, the values are stored temporarily. They are stored as a copy, not as a reference.
"None"	: No return value
"OK"	: Defines the return value for which the transfer parameter is written
"Cancel"	
"Abort"	
"Retry"	
"Ignore"	
"Yes"	
"No"	

"Open dialog modal"	: Only the dialog processes user inputs. The remaining visualizations are blocked to user input.

"Position to open"	
"Centered"	The dialog opens in the center of the visualization.
"Position"	The dialog opens at the position defined by *"X"* and *"Y"*.
"X"	Position (in pixels) or variable (integer data type)
"Y"	Position (in pixels) or variable (integer data type)

See also

- *Chapter 1.4.5.15.3 "Calling a dialog in a visualization" on page 1338*

Input action 'Change Language'

"Language"	Language to be switched Example: `en` : The input assistance offers all languages available in the project.

Input action 'Change Shown Visualization'

Table 316: “Zoom to visualization”

Visualization that is shown at the user input	
“Assign”	Visualization that is selected from all available visualizations in the project or libraries. Example: `visMain`
“Assign expression”	Variable (`STRING`) that contains the name of the visualization Example: `PLC_PRG.strVisu` for the following application code: `strVisu: STRING := 'visMain';`
The order in which visualizations are displayed by user inputs is saved internally. The following options use this information.	
“Previous shown visualization”	☑: Visualization that has already been shown before the current one Requirement: A visualization switch has already occurred.
“Next shown visualization”	☑: Visualization that is next in the call order after the current one. Requirement: A visualization switch has already occurred which was called by *“Previous shown visualization”*.

Input action 'Execute Command' Commands are listed here with transfer parameters that the visualization processes when an input event occurs.

“Configure commands”	▾ ● *“Execute program on the plc”* ● *“Execute program on client”* ● *“Print”* ● *“Navigate to URL (WebVisu)”* ● *“Create Recipe”* ● *“Read Recipe”* ● *“Write Recipe”* ● *“Write Recipe in File”* ● *“Load Recipe from File”* ● *“Delete Recipe ”* Click + to add the selected command to the lower command list.

+	The command in *“Configure commands”* is added to the list.
−	The command is removed. Requirement: A command is selected.
The order in the list defines the order of execution.	
▼	The selected command is moved down one position in the list.
▲	The selected command is moved up one position in the list.

Table 317: Command “Execute program on the plc”

“Command”	*“1st parameter”*	*“2nd parameter”*
`ExecutePlcProgram`	`'C:\programs\notepad.exe'`	`'Notes_A.txt'`
EXE file that is executed on the controller The program is executed on the PLC and therefore it must not be interactive or have any user interfaces. It is possible, for example, for a program to copy a file.	Program name with directory as `STRING` in single straight quotation marks	Arguments of the program as `STRING` in single straight quotation marks Example: Name of the file that the program opens

Table 318: Command “Execute program on client”

“Command”	*“1st parameter”*	*“2nd parameter”*
`ExecuteClientProgram`	`'C:\programs\notepad.exe'`	`'Notes_A.txt'`
EXE file that is executed on the display variant. Exception: WebVisu. The program is executed within the context of the display variant. After this, the program may be interactive and have a user interface.	Program name with directory as `STRING` in single straight quotation marks	Arguments of the program as `STRING` in single straight quotation marks Example: Name of the file that the program opens

NOTICE!
If the visualization is displayed as a CODESYS WebVisu, then no program (EXE file) can be started.

Table 319: Command “Navigate to URL (WebVisu) ”

“Command”	*“1st parameter”*	*“2nd parameter”*
`NavigateURL`	`'http://en.wikipedia.org'` `PLC_PRG.stURL`	`'replace'`
The visualization navigates to the web page of the URL. Requirement: The visualization is executed as a CODESYS WebVisu.	URL ● As a literal in single straight quotation marks ● As a variable (`STRING`)	If a parameter is not specified, then the web page is displayed in a new window or a new tab. If `'replace'` is specified, then the CODESYS WebVisu is replaced by the web page.

Table 320: Command “Read Recipe”

“Command”	*“1st parameter”*	*“2nd parameter”*
`ReadRecipe`	`'RecipeDefinitionForModules'`	`'RecipeModuleA'`

	Name of the recipe definition ● As a literal ● As a variable (STRING)	Name of the recipe ● As a literal ● As a variable (STRING)
At visualization runtime, the controller reads the actual values from the variables of the recipe definition and writes them to the specified recipe. The values are saved implicitly (to a file on the controller) and shown in the recipe definition in the Recipe Manager of CODESYS. In other words, the recipe that is managed in CODESYS is updated with values from the controller.		

Table 321: Command “Write Recipe”

“Command”	*“1st parameter”*	*“2nd parameter”*
`WriteRecipe`	`PLC_PRG.stRecipeDef`	`PLC_PRG.stRecipe`
	Name of the affected recipe definition ● As a literal ● As a variable (STRING)	Name of the recipe (from the recipe definition) ● As a literal ● As a variable (STRING)
At visualization runtime, the values of the recipe are written to the variables on the controller as they are in the Recipe Manager.		

Table 322: Command “Save Recipe in File”

“Command”	*“1st parameter”*	*“2nd parameter”*
`SaveRecipeAs`	`PLC_PRG.stRecipeDef`	`PLC_PRG.stRecipe`
	Name of the affected recipe definition ● As a literal ● As a variable (STRING)	Name of the affected recipe that is updated and saved to a file ● As a literal ● As a variable (STRING) Optional parameter: If you do not specify a transfer parameter here, then the values from the recipe variables are saved only the file that is specified later. The implicit recipe files are not updated.
At visualization runtime, the *“Save Recipe as”* dialog opens and prompts the user for a file name and a storage location on the controller. The file name must not be `<recipe>.<recipe definition>`. The file extension is `.txtrecipe`. The user can then save the file that includes the actual values from the recipe variables. If a transfer parameter is **not** specified in the 2nd parameter, then the file is saved without changing an implicit recipe file. If a transfer parameter is given in the 2nd parameter, then the implicit recipe file is also updated. Note: If the *“Save recipe changes to recipe files automatically”* option is selected in the *“Recipe Manager - General”* tab, then the recipe definition in CODESYS and the implicit recipe files are kept the same automatically. Note: Implicit (automatically generated) recipe files exist on the controller with names in the following syntax: `<recipe>.<recipe definition>.txtrecipe`. These are typically used in the application as a buffer when reading and writing recipe variables.		

Table 323: Command "Load Recipe from File"

"Command"	*"1st parameter"*	*"2nd parameter"*
`LoadRecipeFrom`	`PLC_PRG.stRecipeDef`	`PLC_PRG.stRecipe`
	Name of the affected recipe definition ● As a literal ● As a variable (STRING)	Name of the affected recipe ● As a literal ● As a variable (STRING)
At visualization runtime, the *"Load Recipe"* dialog opens. It provides the visualization user with a file list that is located in the file system of the controller and filters by the extension `txtrecipe`. The selected file is downloaded. Then the recipes from the file are written to the implicit files and read to the given recipe in the recipe definition of the Recipe Manager. Requirement: The file was created with the `SaveRecipeAs` command.		

Table 324: Command "Create Recipe"

"Command"	*"1st parameter"*	*"2nd parameter"*
`CreateRecipe`	`PLC_PRG.stRecipeDef`	`PLC_PRG.stRecipe_New`
	Name of the affected recipe definition ● As a literal ● As a variable (STRING)	Name of the new recipe ● As a literal ● As a variable (STRING)
At visualization runtime, a new recipe is created in the given recipe definition.		

Table 325: Command "Delete Recipe"

"Command"	*"1st parameter"*	*"2nd parameter"*
`DeleteRecipe`	`PLC_PRG.stRecipeDef`	`PLC_PRG.stRecipe`
	Name of the affected recipe definition ● As a literal ● As a variable (STRING)	Name of the recipe ● As a literal ● As a variable (STRING)
At visualization runtime, the specified recipe is deleted from the recipe definition.		

Table 326: Command "Print"

"Command"	*"1st parameter"*	*"2nd parameter"*
`Print`	Optional: File name for the visualization screen to be printed (example: 'Start screen')	A second parameter cannot be specified for the `Print` command.
The default *"Printer"* dialog opens while the visualization is running. In the dialog, you select a printer and configure additional print settings. When you confirm the dialog, the currently displayed visualization screen is printed. Note: The command can be executed in the TargetVisu only.		

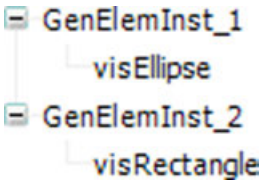
See also

- *Changing Values with Recipes*
- *Object 'Recipe Definition'*

Input action 'Switch Frame Visualization'

When the input event occurs, the display switches to another visualization within one frame.

"Frame selection type"	
"Switch local visualization"	The *"Frame Selection"* group is visible.
"Switch to any visualization"	The *"Frame and visualization selection"* group is visible.

Requirement: *"Switch local visualization"* is selected.	
"Frame selection"	List of all frames that contain the active visualization. The referenced visualizations are listed below each frame, as determined in the *"References"* property of the respective frame. Example: 
"Assign selection"	The selection in the *"Frame selection"* input field is accepted. Then it appears in the *"Selected frame"* and *"Selected visualization"* settings. Requirement: A visualization is selected in the *"Frame selection"* input field.
"Selected Frame"	Name of the frame to be switched to Example: `MainArea` Hint: Use the *"Assign selection"* command for changing the setting here.
"Selected Visu"	Name of the switched visualization. Example: `visMainArea` Hint: Use the *"Assign selection"* command for changing the setting here.

Requirement: The *"Switch to any visualization"* option is selected.	
"Frame and visualization selection"	Contains the frame to be switched to
"Assign"	Frame to be switched to (with complete path). The index determines the visualization. Example: `visMain.frameA.visB.frameB` The path is specified in the following syntax: `<visualization name>.<frame name> { <visualization name>.<frame name> }` Caution: Visualizations can be nested at any depth by means of frame elements. In order to use the *"Switch to any visualization"* frame selection type without any problems, a frame must not contain more than 21 referenced visualizations.

“Assign expression”	Variable (`STRING`). Contains the path of the frame to be switched to Example: `strFrane: STRING := 'visMain.frameA.visB.frameB';`
“Index to select”	Index that determines which of the referenced visualizations is displayed ● As an integer ● As a variable (integer data type) Example :`PLC_PRG.iIndex` Note: The referenced visualizations of a frame are indexed automatically according to their order.

Requirement: The project contains visualizations that form a structure.

See also

- *“Dialog 'Frame Configuration'” on page 1727*

Input action 'Write Variable'

The configuration of the input action defines how a visualization user specifies a value and to which variable the value is written.

Check all inputs for there validity. Be sure that only values within the range can be added to a numeric field. Depending on the datatype of the input value, the limitations can be different.

“Input type” How the input is prompted.	
“Default”:	An input field also opens, or if necessary a virtual keyboard (when the display variant does not have a physical keyboard). Note: The default option for text input at runtime is set in the Visualization Manager: *“Dialog Settings”* tab, *“Settings for Default Text Input”*.
“Text input”	An input field appears. You use the keyboard to specify a number or a text. Requirement: The display variant has a keyboard as input device.
“Text input with limits”	An input field appears. You use the keyboard to specify a number or a text. The field also shows the range of values for the input. When a limit is passed, the input value is displayed in red. Requirement: The display variant has a keyboard as input device.
“VisuDialogs.Keypad”	A virtual keyboard opens. You use it to specify a number or a text.
“VisuDialogs.Numpad”	A virtual keyboard opens. You use it to specify a number.
“VisuDialogs.NumpadExtended”	A virtual keyboard opens. You use it to specify a number. Hexadecimal and exponential notation are also permitted here.

“Choose variable to edit”	
“Use text output variable”	: The input value is written to the text output variable of the element. This is the variable that is assigned in the *“Text variable ➔ Text”* property.
“Use another variable”	: Variable where the input value is written. Example: `PLC_PRG.iVariable`
“Initial display format”	Placeholder with format definition. It defines the output format for the variable value and the input limits. Example: `%2.3f` for displaying the value as a decimal fraction.

"Min"	Minimum value of the input limit. If a user specifies a lesser value, then it is not accepted. ● As a fixed value ● As a variable (data type corresponds to selected variable)
"Max"	Maximum value of the input limit. If a user specifies a greater value, then it is not accepted. ● As a fixed value ● As a variable (data type corresponds to selected variable)
"Dialog title"	Text displayed in the title bar of the dialog. Optional. ● As a fixed string Example: `Insert value` ● As a variable (`STRING`) Example: `PLC_PRG.stTitle : STRING := 'Insert value';`

"Password field"	☑: Unseen text input. `***` is shown instead of the input text.

Table 327: "Position to open input dialog"

"Use global setting (from the Visualization Manager)"	◉: This option is applies only for use in a TargetVisu or WebVisu. The settings are used which are available in the *"Dialog Settings"* tab of the Visualization Manager.
"Centered"	◉: The dialog opens in the center of the visualization window.
"Position"	◉: The dialog opens in the visualization at the position defined here. *"X"*, *"Y"*: Variable or explicit number (in pixels) for the definition of the upper left corner of the dialog in the coordinate system of the visualization window. You can use the placeholders `ElementRectangle.ptTopLeft.iX` and `.iY` `ElementRectangle.ptBottomRight.iY`. It is replaced at runtime by the coordinates of the calling element.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *Chapter 1.4.5.19.3.15 "Dialog 'Properties' of Visualization Objects" on page 1767*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Tab 'Visualization manager' – 'Settings'" on page 1777*

Input action 'Execute ST Code'

Input field	Editor for code as Structured Text

Input action 'Toggle Variable'

"Variable"	Variable (`BOOL`). It toggles between `TRUE` and `FALSE` for an input event. Example: `PLC_PRG.bSwitch`

Input action 'File Transfer' With the *"File Transfer"* input action, a file can be transferred from an operating variant (target or web visualization) to the PLC as well as to and from the PLC. This works either by means of a file transfer (*"Type"*: *"File"*) or streaming (*"Type"*: *"Streaming"*).

The action has the effect that a file selection dialog is displayed in the visualization at runtime. There the visualization user can select a file which will be transferred either to or from the PLC: For a transfer from a PLC to the visualization, the *“Save File”* dialog opens. For the transfer from the visualization to the PLC, the *“Open File”* dialog opens.

“Transfer”

“Direction”	Direction of file transfer
“From PLC to Visualization”	The object specified in *“File name”* or *“Streaming instance name”* is transferred from the PLC to the visualization. The *“Save File”* file selection dialog is displayed in the visualization at runtime.
“From Visualization to PLC”	The file specified by the visualization user is transferred to the PLC and saved in the file path specified in *“File name”* or *“Streaming instance name”*. The *“Open File”* file selection dialog is displayed in the visualization at runtime.

“Type”	Determines how the file is transferred ● Transfer type *“File”*: By file transfer ● Transfer type *“Streaming”*: By streaming

“Type”	*“File”* The data transfer is done by file transfer.
“File name”	File path (type `STRING`) which describes the file in the file system ● Variable Example: `strTransferFile: STRING;` ● Literal with relative path Example: `'/Recipes/Recipe_1.txt'` saves the file in the directory `Recipes`. ● Literal with placeholder `$PLCLOGIC$` `PlcLogic` is the default resolution for the directory placeholder `$PLCLOGIC$`. Example: `'$$PLCLOGIC$$/test.txt'` saves the file in the directory `PlcLogic`. Example: `'$$PLCLOGIC$$/MyData/test.txt'` saves the file in the directory `PlcLogic/MyData`. ● Literal with placeholder `$VISU$` `visu` is the default resolution for the placeholder `$VISU$`. Example: `'$$VISU$$/test.txt'` save the file in the subdirectory `PlcLogic/visu`. Alternatively, `'visu/test.txt'` can also be specified. ● Literal with absolute path Example: `'E:\temp\test.txt'` Note: These kinds of file paths are not always supported. Note: If a user specifies the file path in the visualization by means of a *“Text Field”* element, the masking character `$` must not be included: `$VISU$/dummy.txt` Note: In the case that the file path is specified by the user, it should be checked by the application in order to prevent files from being read or overwritten accidentally.

"Type"	*"Streaming"* The data transfer is done by streaming.
"Streaming instance name"	Instance path (type `IVisuStreamWriter` or `IVisuStreamReader`) which describes the object in the file system of the controller Type `IVisuStreamReader` for transfer direction *"From PLC to Visualization"* Type `IVisuStreamWriter` for transfer direction *"From Visualization to PLC"*

"Control flags"	Note: The variable is evaluated only for transfer direction *"From Visualization to PLC"*. Variable (type `DWORD`) Determines how the object (file or instance object) is handled on the file system of the PLC. Two flags are provided for this with which the variable can be set. ● Flag 1: `VisuElems.VisuEnumFileTransferControlFlags.UseOriginalFileName` ● Flag 2: `VisuElems.VisuEnumFileTransferControlFlags.ConfirmFileOverwriteInPlc` Options ● No flag set: The user selects a file which is saved in the path specified in *"File name"* or *"Streaming instance name"*. ● Flag 1 is set: The path, which is specified by the user at visualization runtime, is applied and used as the path in the PLC file system. ● Both flags are set: The path is also checked. If an object already exists in the path specified on the client side, then a message prompt is displayed in the visualization. There the visualization user can confirm that the file will be overwritten. Example: `dwControlFlag`

Example

The transfer direction is *"From Visualization to PLC"* (write).

Example: A new recipe file `Recipes/Recipe_2021.txt` has been created in the visualization device. The visualization user selects this file and wants to save the file on the PLC under the same name. Because the control flags are set accordingly, a message window opens and the visualization user can confirm that the file will be overwritten.

```
PROGRAM PLC_PRG
VAR
    xVisuToggle : BOOL;
    dwControlFlag : DWORD:=
VisuElems.VisuEnumFileTransferControlFlags.UseOriginalFileName +
VisuElems.VisuEnumFileTransferControlFlags.ConfirmFileOverwriteInPlc
;
    strFileName: STRING := '/Recipes/Recipe_new.txt';
END_VAR
```

Table 328: “Status Variables”

“Transfer active”	Boolean variable (optional) `TRUE`: The transfer is in progress.
“Transfer successful”	Boolean variable (optional) `TRUE`: The transfer has completed successfully.
“Error code”	● `0`: No errors ● `1`: Unspecified error ● `2`: Cancellation of file dialog ● `3`: Other file transfer in progress ● `4`: Error during file transfer ● `5`: Cancellation by timeout ● `6`: File read error – The file is not available or cannot be read. ● `7`: No device support for file transfer Possible causes: – CODESYS WebVisu: File transfer is not possible by default. – Communication with a controller of a version < 3.5.11: Functionality not implemented. – Communication with a controller of a version >= 3.5.11: File transfer not activated (device description). Note: In this case, contact the CODESYS support team.

Dialog 'Options' - 'Visualization Styles'

Symbol: 

Function: This dialog is used for configuring the display of library visualizations and visualizations in the POUs view in the visualization editor. In addition, it is used for configuring the tab *“Visualization Manager”* - *“Settings”* (group *“Style Settings”*).

Call: Menu bar: *“Tools* ➔ *Options”* (*“Visualization Styles”* category).

These settings are not applied at visualization runtime. In runtime mode, only the settings of the visualization manager are available in the *“Settings”* tab.

See also

- *Chapter 1.4.5.17 “Applying Visualization Styles” on page 1360*

'Style Configuration for Libraries and Global Visualizations'

These settings are applied for library visualizations and for visualizations in the POUs view.	
“Use no visualization style”	: Display without using style properties. Elements are displayed as defined by presets.
“Use the following visualization style”	: Style with style properties used for displaying visualizations.

"Derive visualization style automatically"	: Display with the style that was selected in the application in the visualization manager (when possible). Therefore, the display is derived from this style. It is actually possible for this to cause an incorrect display. Then the fallback solution is used.
"Fallback if no visualization style could be derived "	Another style that is applied after the selected style. Then a style property is assigned from the style specified here. This is done for element properties that could not be assigned style properties. Requirement: The selected style causes a device-specific, deficient display on the display variant.

'Style Selection'

The drop-down list of *"Selected style"* can be configured in the visualization manager (*"Settings"* tab, *"Style settings"* group).	
"Display all versions"	: All other styles of the repository, including the selected style, are listed for selection, but only in the latest version. If newer versions are installed for the selected style, then these are also listed. : All installed styles in all installed versions are available for selection.

'Style for New Visualization Managers'

"Last used: <style, version, vendor>"	Style that is selected automatically when you add a new visualization application. Note: It is actually possible that a display variant is displayed another way depending on the device despite this setting.
"Preset: <style, version, vendor>"	
"<style, version, vendor>"	

Dialog 'Options' - 'Visualization User Management'

Symbol:

Function: The options define the use of visualization user management for global visualizations in the *"POUs"* view and for visualizations that are linked from libraries.

Call: Menu bar: *"Tools"*.

Requirement: A visualization user management exists.

Table 329: "User Management Configuration for Libraries and Global Visualizations"

"Do not use visualization user management"	The affected visualizations behave as when no user management is configured.
"Use the following visualization user group list"	• You can edit the list. • The list is created in the *"Visualization manager"* (*"User management ➔ Groups"*) by clicking *"Export groups for global visualizations"*.
" Derive visualization user management automatically"	The affected visualizations use the user management configuration of the visualization manager selected here. The drop-down list shows all visualization managers of the project. If this is not possible, then the user groups are used from the option *"Use the following user group list for the visualization"*.

The user management for a visualization in the "Devices" view is configured in the "Visualization Manager" (tab "User Management").

See also

- *Chapter 1.4.5.19.4.5 "Tab 'Visualization manager' - 'User management'" on page 1782*

Dialog Box 'Options' - 'Visualization'

Symbol: 

Function: The dialog serves for the configuration of the visualization editor and during runtime it serves the configuration of the Integrated Visualization.

Call: Main menu *"Tools ➔ Options"*, category *"Visualization"*

Tab 'General'

*These settings will **not** be applied for the following visualization clients: CODESYS TargetVisu, CODESYS WebVisu.*

Table 330: "Presentation options (visualization editor in the programming system)"

"Fixed"	The visualization maintains its original size
"Isotropic"	The visualization maintains its proportions
"Anisotropic"	The visualization adapts to the size of the visualization window
"Antialiased Drawing"	☑: The visualization is drawn with the help of antialiasing methods. This applies while you are editing and also when the visualization is running as Diagnosis Visualization.

Table 331: "Editing options"

"Link to toggle/tap variable when appropriate"	☑: The placeholder *"<toggle/tap variable>"* in the visualization element properties is enabled. Effect: If you drag an element having the property *"Color variable ➔ Toggle color"* in the visualization editor, this property will be configured with the placeholder *"<toggle/tap variable>"*. The following elements are affected: *"Button"*, *"Frame"*, *"Image"*, *"Line"*, *"Pie"*, *"Polygon"*, *"Rectangle"*, *"Text field"*, *"Scrollbar"*.

See also

- *Chapter 1.4.5.19.5.11 "Visualization Element 'Button'" on page 1892*
- *Chapter 1.4.5.19.5.6 "Visualization Element 'Frame'" on page 1856*
- *Chapter 1.4.5.19.5.5 "Visualization Element 'Image'" on page 1842*
- *Chapter 1.4.5.19.5.2 "Visualization Element 'Line'" on page 1804*
- *Chapter 1.4.5.19.5.4 "Visualization Element 'Pie'" on page 1829*
- *Chapter 1.4.5.19.5.3 "Visualization Element 'Polygon', 'Polyline', 'Bézier Curve'" on page 1816*
- *Chapter 1.4.5.19.5.1 "Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'" on page 1792*
- *Chapter 1.4.5.19.5.14 "Visualization Element 'Text Field'" on page 1916*
- *Chapter 1.4.5.19.5.15 "Visualization Element 'Scroll Bar'" on page 1928*

Tab 'Grid'

"Visible"	☑: The visualization editor contains a grid. The spacing of the grid lines is defined by *"Size"*.
"Active"	☑: The visualization elements get aligned to the grid, defined by *"Size"*, even if the grid lines are not visible. When you insert or move an element, its center will be positioned on the grid. When you modify the size of an element, you can move the position markers onto grid lines only. Elements already available in a visualization, will not be aligned automatically, until you change their position.
"Size"	Spacing of the grid lines in pixel.

Tab 'File options'

"Text list files for textual "IntelliSense"	File name and path of a file of type `.csv`. The file contains texts in the format of a text list. The file entries will be available when using the function "List Components" as input assistance. Note: You can create this file as an export file of the global text list. For this purpose use the command *"Import/Export Text Lists"*.
"Visualization Directories"	
"Text list files"	Storage path for text lists. Note: This setting will be used in CODESYS only if no storage path for *"Text list files"* is defined in the *"Project Settings"*, category*"Visualization"*.
"Image files"	Storage path for image files. Multiple paths get separated by semicolons. CODESYS uses this path for example when exporting or importing image files. Note: This setting will be used in CODESYS only if no storage path for *"Image files"* is defined in the *"Project Settings"*, category*"Visualization"*.

See also

- *Chapter 1.4.1.8.8 "Managing text in text lists" on page 266*
- *Chapter 1.4.1.20.3.20.6 "Command 'Import/Export Text Lists'" on page 1133*

See also

- *Chapter 1.4.5.19.3.13 "Dialog 'Project Settings' - 'Visualization'" on page 1766*
- *Chapter 1.4.5.19.4.7 "Object 'TargetVisu'" on page 1787*

Dialog 'Project Environment' - 'Visualization Profile'

Function: The dialog displays the current visualization profile of the project. The profile can be updated here.

Call: Main menu *"Project ➔ Project Environment"*, Tab *"Visualization Profile"*.

"Current visualization profile in project"	The currently set visualization profile of the opened project.
"Recommended, newest profile"	The newest profile
"Action"	
"Do not update"	The visualization profile of the project remains unchanged.

“Update to x.x.x.x”	CODESYS updates the project to the chosen visualization profile.
“Check for updates when loading this project”	☑: CODESYS checks for new profiles when the project is opened. If there are updates available an update dialog opens automatically. ☐: No check of the profile when loading the project. The update dialogs do not open automatically any longer.
“Set All to Newest”	CODESYS updates the profile.

Dialog 'Project Environment' - 'Visualization Styles'

Function: The dialog displays the currently used visualization style of the project. The visualization style can be updated here.

Call: Main menu *“Project ➔ Project Environment”*, tab *“Visualization Styles”*

“For the following visualization styles currently in use, newer versions are available:”	
“Visualization Styles”	Version of the currently set visualization style of the opened project.
“Current”	Current version of the visualization style, for example `3.5.6.0`
“Recommended”	Recommenden version of the visualization style, for example `3.5.7.0`
“Action”	
“Do not update”	The visualization style of the project remains unchanged.
“Update to x.x.x.x”	CODESYS updates the project to the version of the chosen visualization style.
“ Check for updates when loading the project”	☑: CODESYS checks for new versions when the project is opened. If there are updates available an update dialog opens automatically. ☐: No check of the version. The update dialogs do not open automatically any longer.
“Set All to Newest”	CODESYS updates the version.

See also

- *Chapter 1.4.5.17 “Applying Visualization Styles” on page 1360*

Dialog 'Project Environment' – 'Visualization Symbols'

Function: The dialog lists installed symbol libraries and allows for you to assign symbol libraries to a project.

Call: Menu bar: *“Project ➔ Project Environment”*, *“Visualization Symbols”* tab

Requirement: The open project contains a visualization and has been saved with a compiler version `< 3.5.7.0`. CODESYS recognizes symbol libraries in compiler version `3.5.7.0` and higher.

“Symbol library”	List of all installed symbol libraries
“Active”	☑: Symbol library is selected for the project. CODESYS provides its symbols in the *“Visualization Toolbox”* view. ☐: Symbol library has been previously installed only in the library repository.

See also

- *Chapter 1.4.5.3.1 “Select Element” on page 1255*

Dialog 'Project Settings' - 'Visualization'

Symbol:

Function: The dialog is used to configure the project-wide settings for objects of type *"Visualization"*.

Call: Menu bar: *"Project ➔ Project Settings"*, *"Visualization"* category

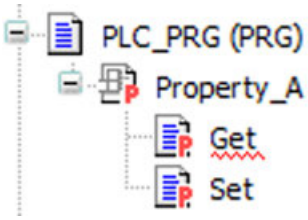
Requirement: A project is open.

Tab 'General'

Table 332: "Visualization Directories"

"Text list files"	Directory which contains text lists that are available in the project to configure texts for different languages. CODESYS uses the directory, for example to import or export text lists. After clicking ..., the *"Select Directory"* dialog opens which allows for the selection of a directory in the file system.
"Image files"	Directory which contains image files that are available in the project. Multiple folders are separated with a semicolon. CODESYS uses the directory, for example to import or export image files. After clicking ..., the *"Select Directory"* dialog opens which allows for the selection of a directory in the file system.

Table 333: "Advanced"

"Activate property handling in all element properties"	☑: You can also configure a visualization element with a property in those of its properties in which you select an IEC variable. Then CODESYS creates additional code for the property handling when a visualization is compiled. Requirement: Its IEC code contains at least an object of type *"Interface property"* (a property).  Requirement: *"Visible"* is selected.

See also

- *Object 'Property'*

Tab 'Symbol Libraries'

Table 334: "Visualization Symbol Libraries"

"Symbol libraries"	List of all installed symbol libraries Example: `VisuSymbols`
"Assigned"	☑: Symbol library is selected in the project and CODESYS makes it available in the *"Visualization ToolBox"* view of a visualization. ☐: Symbol library is installed in the library repository, but CODESYS does **not** make it available in the *"Visualization ToolBox"* view of a visualization.

See also

- *Chapter 1.4.5.19.3.2 "Dialog 'Add Visualization'" on page 1746*

Dialog ‘Project Settings’ - ‘Visualization Profile’

Symbol: 

Function: The dialog box enables the setting of the visualization profile.

Call: Menu *“Project ➔ Project Settings”*, category *“Visualization Profile”*

Requirement: A project is open.

Table 335: “Visualization Profile”

“Specific profile”	Profile that CODESYS uses in the project and that determines the visualization elements that are available in the project. The selection list contains all the profiles installed so far.

Dialog 'Properties' of Visualization Objects

Function: This dialog is used for configuring object-dependent properties.

Call: Menu bar: *“View”*; context menu of the visualization object in the *“Devices”* view or *“ POUs”* view.

Tab 'General'

“Name ”	Example: `visMain`
“Object type ”	visualization
“Open with ”	visualization

See also

- *Chapter 1.4.1.20.4.10.1 “Dialog Box 'Properties' - 'Common'” on page 1157*

Tab 'Access Control'

This tab is used for defining which user group can execute which actions on the object.

See also

- *Chapter 1.4.1.20.4.10.6 “Dialog 'Properties' - 'Access Control'” on page 1161*

Tab 'Visualization'

This tab assigns a visualization type to a visualization.

In addition, it includes settings for window size that are used at runtime.

Table 336: “Use visualization as”

“Visualization”	Visualization type for an ordinary or referenced visualization. Preset.
“Dialog ”	Visualization type for a visualization that opens as a dialog in its own window for an input event. The input action for this is *“Open dialog”*. The *“Close dialog”* input action closes the window. Tip: A dialog usually includes an *“OK”* button or *“Cancel”* button at the bottom edge for confirming or rejecting user input, and for closing the dialog. A simple dialog or a dialog prompt includes only a question or information and buttons for closing the dialog with either *“Yes”* or *“No”*. A dialog is part of a user interface. While a dialog is open, the rest of the user interface is usually disabled.
“Numpad / keypad / dialog for input configuration”	Visualization type for a visualization that displays a virtual numeric keypad or a virtual keyboard. It appears when the user is prompted to specify text. The input action for this is *“Write variable”*. Note: The interface of this visualization must also conform with the interfaces for the standard visualizations for the numeric keypad or that keyboard that provides the `VisuDialogs` library: `Numpad`, `Keypad`, `NumpadExtended`, or `TextinputwithLimits`. Tip: The `VisuDialogs` library contains templates for virtual keyboards or numeric keypads.

“Dialog is opaque”	☑: The screen area that is covered by the dialog is not refreshed. This has a positive effect on the character and input performance. Use this option when your drawn dialog is rectangular and opaque, containing no transparent parts.
“Use automatic detected visualization size”	◉: The size is determined so that all visualization elements are enclosed.
“Include background image”	☑: All elements and the background image are completely visible. ☐: All elements are visible, but a larger background image is truncated.
“Use specified visualization size”	◉: The values *“Height”* and *“Width”* define the window size of the visualization (in pixels).

“Internal”	☑: The visualization is internal. It is used exclusively as an internal module of a complete visualization in a library. When editing as a library project while the project is open in CODESYS, an internal visualization is handled like all visualizations. The internal visualization appears in drop-down lists. Or in the visualization manager (*“Visualizations”* tab). The internal visualizations that include a linked library are not visible to you.

See also

- *“Dialog 'Frame Configuration'” on page 1727*

Tab 'Build'

This tab includes options for compiling the object.

See also

- *Chapter 1.4.1.20.4.10.4 “Dialog 'Properties' - 'Build'” on page 1159*

Dialog 'Selected Alarm Class'

Function: In this dialog box, you define the alarm classes that are considered for the alarm table or alarm banner.

Call: Property *“Alarm configuration”* / *“Alarm classes”* of the alarm table or alarm banner visualization element.

Requirement: An alarm table visualization element or alarm banner visualization element is added to the visualization.

“Available Alarm Classes”	Shows all alarm classes created in the project.
“Selected Alarm Classes”	The alarm classes in this column are displayed in the alarm table.
“All”	☑: All alarm classes are listed in an alarm table.
>>	Moves all available alarm classes to the *“Selected Alarm Classes”* column.
>	Moves the selected alarm classes to the *“Selected Alarm Classes”* column.
<	Removes the selected alarm classes from the *“Selected Alarm Classes”* column.
<<	Removes all selected alarm classes from the *“Selected Alarm Classes”* column.

See also

- *Chapter 1.4.5.19.5.22 “Visualization Element 'Alarm Table'” on page 1969*
- *Chapter 1.4.5.19.5.22 “Visualization Element 'Alarm Table'” on page 1969*
- *Chapter 1.4.5.7 “Visualizing alarm management” on page 1289*

Dialog 'Selected Alarm Group'

Function: In this dialog box, you define the alarm groups that are considered for the alarm table or alarm banner.

Call: Property *“Alarm configuration”* / *“Alarm groups”* of the alarm table or alarm banner visualization element.

Requirement: An alarm table visualization element or alarm banner visualization element is added to the visualization.

“Available Alarm Groups”	Shows all alarm groups created in the project.
“Selected Alarm Groups”	The alarm groups in this column are displayed in the alarm table.
“All”	☑: All alarm groups are listed in an alarm table.
>>	Moves all available alarm groups to the *“Selected Alarm Groups”* column.
>	Moves the selected alarm groups to the *“Selected Alarm Groups”* column.
<	Removes the selected alarm groups from the *“Selected Alarm Groups”* column.
<<	Removes all alarm groups from the *“Selected Alarm Groups”* column.

See also

- *Chapter 1.4.5.19.5.22 “Visualization Element 'Alarm Table'” on page 1969*
- *Chapter 1.4.5.19.5.22 “Visualization Element 'Alarm Table'” on page 1969*
- *Chapter 1.4.5.7 “Visualizing alarm management” on page 1289*

Dialog 'Advanced Trace Settings'

Function: The recording rate of the *"Trace"* visualization element is configured in this dialog box.

Call

- Properties: *"Trace"* , *"Advanced"*
- Context menu: *"Configure trace"*, *"Advanced"*

Requirement: A trace is selected in the active visualization editor.

Table 337

"Measurement in every nth cycle"	The task where the trace is running is the basis for the measurement. The measurement interval is a multiple of the trace task according to the selected value. The measurement interval is displayed on the right side.
"Buffer size (samples)".	The number of measurements is calculated according to the time range of the x-axis.

See also

- *Chapter 1.4.5.19.2.13 "Command 'Configure Trace'" on page 1734*

Dialog 'Display Settings'

Function: The dialog includes the configuration for the display settings of the trace diagram (for both the X-axis and Y-axis) and provides a preview in the trace diagram.

Call: *"Display"* button in *"Trace Configuration"* dialog.

"Add Y-Axis"	Extends the trace diagram by one Y-axis. Result: The *"Trace Configuration"* dialog contains an extended selection of Y-axes in the *"Additional axes"* option of the variable settings.
"Delete Y-axis"	Deletes the Y-axis with the visible tab.

Tab 'X-axis'

"Display Mode"	Scaling ● *"Auto"*: ☑: CODESYS Scales automatically. ● *"Fixed length"*: ☑: CODESYS displays a segment of constant *"Length"*. ● *"Fixed"*: ☑: CODESYS displays a segment from *"Minimum"* to *"Maximum"*.
"Minimum"	Initial value of the segment. Requirement: The *"Display Mode"* is *"Fixed"*.
"Maximum"	End value of the segment. Requirement: The *"Display Mode"* is *"Fixed"*.
"Length"	Constant length of the segment.
"Grid"	☑: Diagram with vertical grid lines. Select the line color from the list box of colors.

Table 338: "Tick marks"

"Fixed spacing"	☑: CODESYS draws tick marks with *"Distance"* and *"Subdivisions"*.
"Distance"	Distance between tick marks
"Subdivisions"	Number of subdivisions between two tick marks

“Font”	Font for the X-axis

Tab 'Y-Axis'

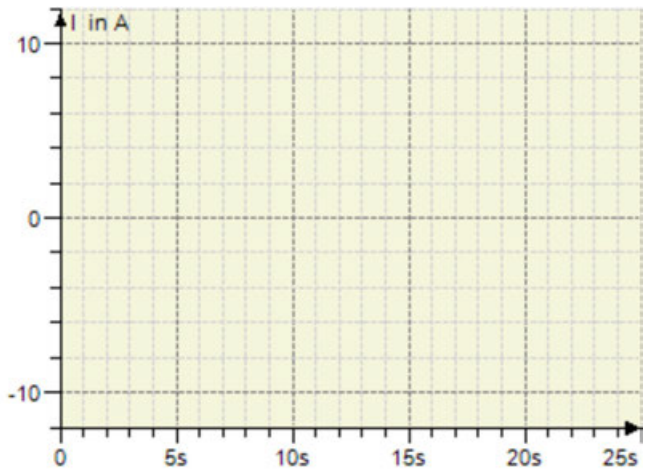
“Display Mode”	Scaling ● *“Auto”*: ☑: CODESYS Scales automatically. ● *“Fixed”*: ☑: CODESYS displays a segment from *“Minimum”* to *“Maximum”*.
“Minimum”	Literal, variable (integer data type), or constant variable (integer data type). It contains the initial value of the segment. Requirement: The *“Display Mode”* is *“Fixed”*. Examples: `20,PLC_PRG.iLimit_Min, GVL.c_iLimit_Min` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Min : INT := 20`
“Maximum”	Literal, variable (integer data type), or constant variable (integer data type). It contains the end value of the segment. Requirement: The *“Display Mode”* is *“Fixed”*. Examples: `80,PLC_PRG.iLimit_Max, GVL.c_iLimit_Max` Note: The variable has to have an initial value. This is important for the offline display and the scaling subdivision. Example: `iLimit_Max : INT := 80`
“Grid”	☑: Diagram with a grid line. Select the line color from the list box of colors.
“Label”	☑: The description is displayed on the axis.

Table 339: “Tick marks”

“Fixed spacing”	☑: CODESYS draws tick marks with *“Distance”* and *“Subdivisions”*.
“Distance”	Distance between tick marks
“Subdivisions”	Number of subdivisions between two tick marks

“Font”	Font for the Y-axis

Preview of the trace diagram



“Background color”	● *“No background”*: Transparent display without background color. ● *“Draw background”*: Background color according to selection below. ● *“From visualization style”*: Background color as defined in the visualization style.

“Reset”	CODESYS resets all settings to the defaults.
“Use as default”	CODESYS saves the settings as default

1.4.5.19.4 Objects

Object 'Visualization' and visualization editor

Symbol: 

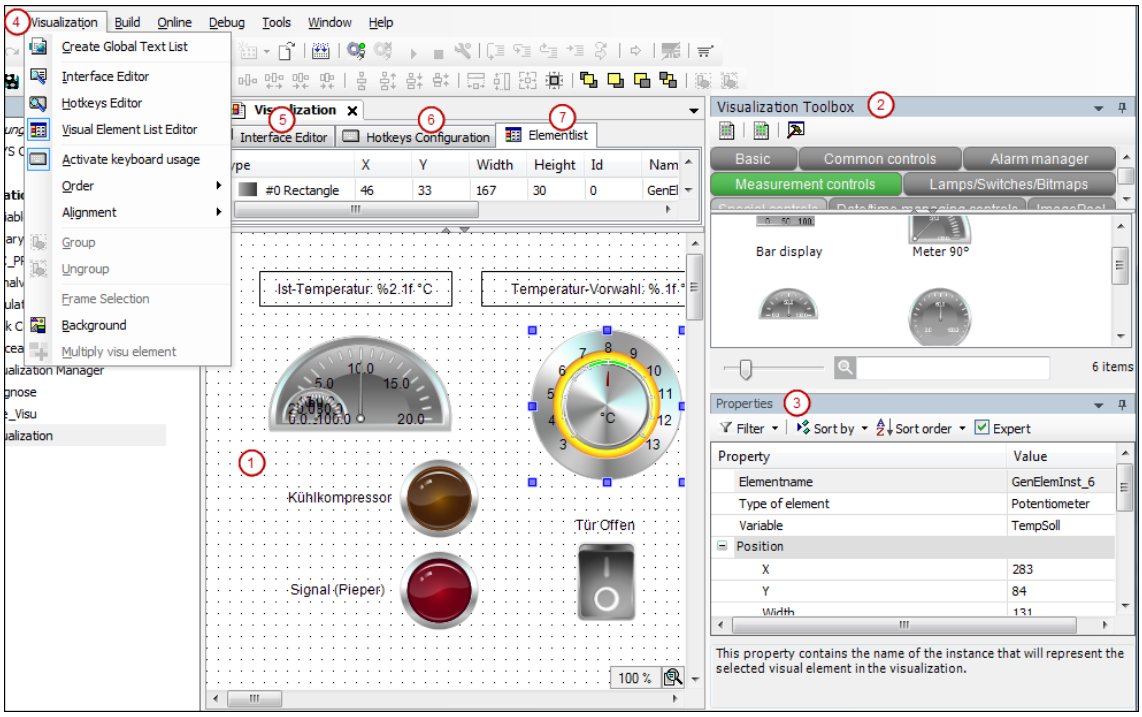
The object represents a single visualization. You can insert a visualization under an application or, so that it is available project-wide, under the root node of the view *“Devices”* or directly in the view *“POUs”*. You can open the visualization editor for editing by double-clicking on the object entry in the device tree or in the view POUs.

See also

- *Chapter 1.4.5.19.3.15 “Dialog 'Properties' of Visualization Objects” on page 1767*
- *Chapter 1.4.5.19.4.1.1 “Visualization Editor” on page 1772*

Visualization Editor

The visualization editor opens when you double-click a visualization object.



- (1) Graphical editor: Here you create the visualization from the visualization elements which are provided in the visualization toolbox view.
- (2) View *“Visualization Toolbox”*: available visualization elements
- (3) View *“Properties”*: Configuration editor for the visualization element currently selected in the editor area
- (4) Menu *“Visualization”*: Commands for working in the visualization editor

The *“Visualization”* menu contains, for example, commands for opening additional editors.

- (5) *“Interface Editor”*: Declaration of variables which can be used to parameterize references of the visualization.
- (6) *“Hotkeys Configuration”*: Definition of keyboard shortcuts for inputs on the visualization in online mode.
- (7) *“Element List”*: List of all elements used in the visualization; possibility to change their position on the Z-axis.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.4.4 “Configuring Keyboard Shortcuts” on page 1274*
- *Chapter 1.4.5.15.2 “Calling a Visualization with an Interface” on page 1332*
- *Chapter 1.4.5.19.4.1.2 “View 'Visualization Toolbox'” on page 1773*
- *Chapter 1.4.5.19.4.1.3 “View 'Properties' of a visualization element” on page 1775*
- *Chapter 1.4.5.19.2 “Commands” on page 1718*
- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*
- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.2.3 “Command 'Visualization Element List'” on page 1721*

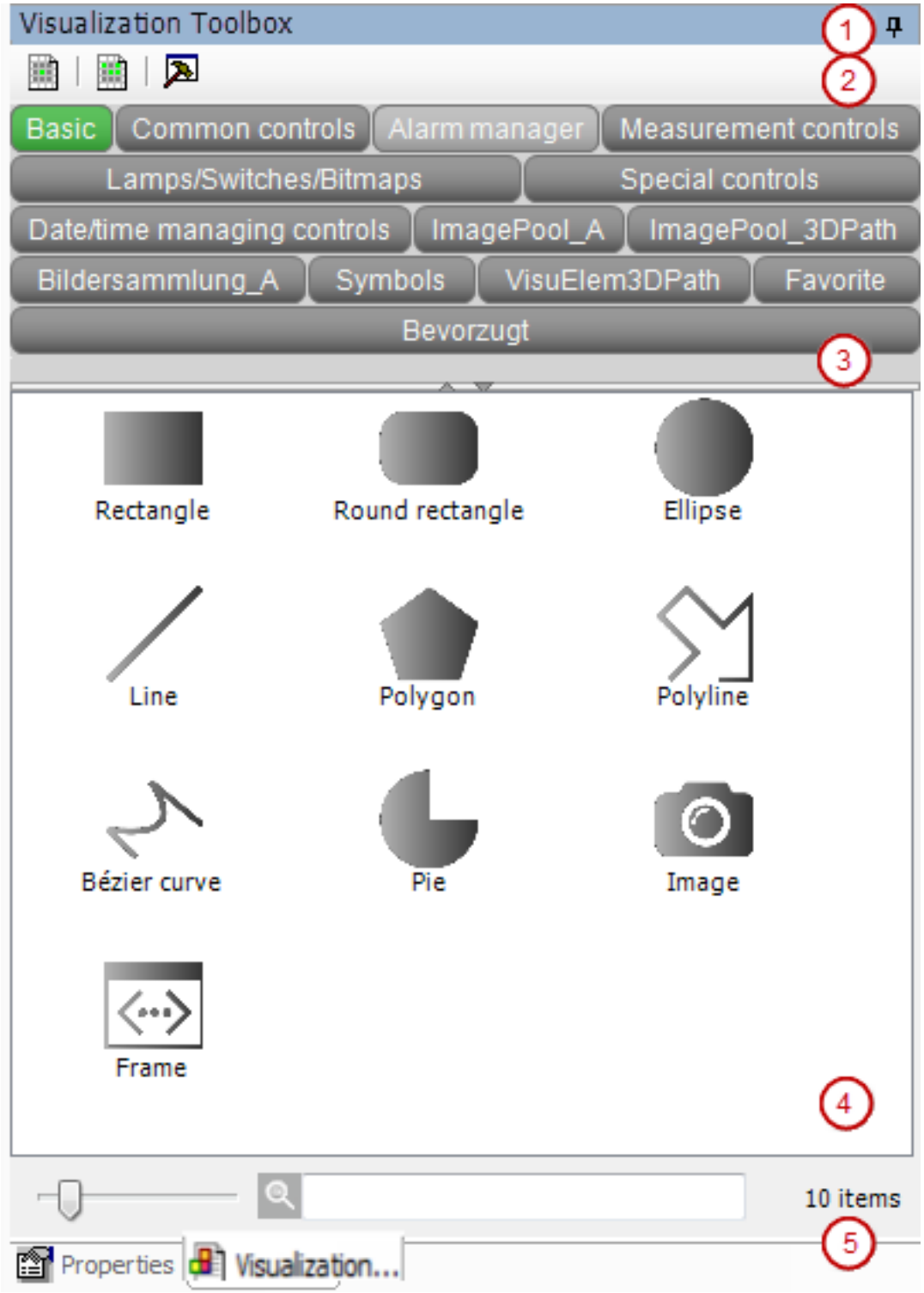
View 'Visualization Toolbox'

Symbol:

Function: The view provides the elements that can be used in the editor. The individual elements are assigned with specific categories. There is a button for each category. The elements of selected categories are displayed with thumbnails which can be dragged into the editor. In addition to the standard categories, you can also define your own categories. You can resize the thumbnails with the slider or perform a full-text search of element names.

Call: Menu bar: *"View ➔ "*

Requirement: A visualization is active.



- (1) *"Visualization Toolbox"* view
- (2) Toolbar with commands
- (3): Buttons fro selecting element categories
- (4) Selection of individual visualization elements
- (5) Controls

See also

- *Command 'Toolbox'*

Toolbar with commands

Symbol: 	Only one button can be selected.
Symbol: 	Multiple buttons can be selected.
Symbol: 	The *“Configure Categories and Items”* dialog opens.

See also

- *Chapter 1.4.5.19.3.4 “Dialog 'Configure Categories and Items'” on page 1747*

Buttons for selecting element categories

A button is displayed for each defined element category. A selected button is displayed in green.

[Shift] + click a button	Changes the selection of the category and the selection type (single or multiple selection possible)
Right-click a button	The context menu opens.

Table 340: Context menu of a button

“Hide Category”	Removes the button. Then the category is removed from view.
“Enable Category”	The button turns green and the category is enabled, irrespective of the selection type.
“Disable Category”	The button turns gray and the category is disabled, irrespective of the selection type.

See also

- *Chapter 1.4.5.3.1 “Select Element” on page 1255*
- *Chapter 1.4.5.19.3.4 “Dialog 'Configure Categories and Items'” on page 1747*
- *Command 'Toolbox'*

Selection of visualization elements

The visualization elements are displayed as thumbnails and labeled with names. The selection depends on the search query in  or on the chosen buttons.

Slider 	To resize of the thumbnails.
 with input field	For a full-text search by element name of all available elements
“<number> items”	Number of visualization element items that are currently displayed as a result of the selected buttons and the search query in  .

View 'Properties' of a visualization element

Symbol: 

Function: This view is used for configuring the element properties of the selected visualization element.

Call: Menu bar: *“View ➔ Element Properties”*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Menu bar

Table 341: “Filters”

“All categories”	List of all element properties
“Default”	List of the most frequently used element properties
“Simple”	List of certain basic element properties, such as *“Texts”*, *“Colors”*, and *“Input configuration”*
“Animation”	List of element properties for animation with variables
“Colors”	List of element properties for designing with color
“Texts”	List of element properties for designing with text
“Input”	List of element properties for configuring user input

Table 342: “Sort”

“Sort by type”	☑: Element properties are sorted by the original order of categories.
“Sort by name”	☑: Element properties are sorted in alphabetical order.

Table 343: “Order”

“Sort ascending”	☑: The properties are sorted from A to Z.
“Sort descending”	☑: The properties are sorted from Z to A.

“Expert”	☑: The table includes all properties. The menu command *“Filter ➔ Show all categories”* is enabled at the same time.

Element properties display in a table

Column *“Property”*	Element properties of the selected element
Column *“Value”*	The assigned value is applied in the editor view.

Double-click in the *“Value”* column	A line editor, drop-down list, or dialog opens for editing the value. ... opens the *“Input Assistant”* dialog for help, for example when assigning variables or image references.
Single-click for a selected field	
[Blank] for a selected field	

A style selected in the visualization manager can include single, predefined element properties. As a result, these do not appear in this view because a fixed value is already assigned to them. They do not have to be configured anymore.

Visualizations can be configured with device-specific restrictions that block the availability of element properties.

Device-specific restrictions :

- Elements with restricted availability
- Fonts with restricted availability
- Colors with restricted display
- Image formats with restricted display
- Maximum number of visualization elements
- Maximum number of visualizations below the device

Object 'Visualization manager'

Symbol: 

The visualization manager manages the configuration settings for all display variants of the visualizations of the current application.

The object is automatically inserted when a visualization object is inserted below the application. On a double-click the configuration dialog opens with several tabs.

If the device employed supports display variants of the visualization, the visualization manager automatically brings along the corresponding objects (CODESYS WebVisu, CODESYS TargetVisu).

If the device employed supports CODESYS TargetVisu, the visualization manager automatically brings along the corresponding object CODESYS TargetVisu.

See also

- *Chapter 1.4.5.16 “Configuring and executing display variants” on page 1354*
- *Chapter 1.4.5.19.4.7 “Object 'TargetVisu'” on page 1787*
- *Chapter 1.4.5.19.4.8 “Object 'WebVisu'” on page 1788*

Tab 'Visualization manager' – 'Settings'

Symbol: 

Function: the tab contains settings for all visualizations that are available application-wide.

'General settings'

“Use unicode strings”	☑: The visualization codes character strings as Unicode.
“Use CurrentVisu variable”	☑: the application knows and uses the global variable `VisuElems.CurrentVisu` of the type `STRING`. It contains the name of the currently active visualization at the runtime of the application. The application can read from the variable in order to obtain the name of the currently active visualization. The application can cause a visualization change by a write access. Requirement: the application contains a visualization that calls further visualizations. Example ● Variable assignment: `VisuElems.CurrentVisu:=strVisuName;` ● Visualization name assignment: `VisuElems.CurrentVisu:='visu1';`

'Style settings'

The “Visualization Style Editor” enables new styles to be generated, checked and installed in the visualization styles repository.

“Selected style”	• Style from the visualization styles repository that every visualization in the application uses, for example *“Flat Style”*. • *“<None>”*: The visualization displays its elements without style or according to the internal default. A standard dialog appears instead of a selection list for selection in the element properties *“Color”* und *“Font”*.
	Opens a selection list with styles that are installed in the visualization styles repository.
“Display all versions (for experts only)”	 : The selection list contains only the latest version of each selected style and all other styles. If a newer version of the selected style is installed it is also displayed.  : The selection list contains all versions of all installed styles.
Button 	Opens a selection list with commands for the use of the *“Visualization Styles Editor”*.
“Open Style Editor”	The *“Visualization Styles Editor”* opens.
“Create and edit derived style”	The *“Visualization Styles Editor”* opens with the dialog *“Create a new visualization style”*. The dialog contains the settings for the first configuration step. Requirement: a style is selected in *“Selected style”*.
“Copy and edit the selected style”	The *“Visualization Style Editor”* opens with the dialog *“Open existing style as a copy”*. The dialog contains the settings for the first configuration step. Requirement: a style is selected in *“Selected style”*.
“Preview”	The elements displayed represent the style specified in *“Selected style”*.

See also

- *Chapter 1.4.5.17 “Applying Visualization Styles” on page 1360*
- *Chapter 1.4.5.17.2 “Managing visualization styles in repositories” on page 1365*
- *Chapter 1.4.5.20.1 “Dialog 'Create a New Visualization Style'” on page 2127*
- *Chapter 1.4.5.20.2 “Dialog 'Open Existing Style as a Copy'” on page 2127*
- *Chapter 1.4.5.19.3.7 “Dialog 'Options' - 'Visualization Styles'” on page 1761*

'Language settings'

“Selected language”	Language used by the display variants at the start of a visualization.

'Settings for default text input'

For an element with standard text input, a dialog that supports the input appears at runtime. You can specify which dialog appears.

"Numpad"	Dialog that calls the visualization if a user activates the input field for a number at runtime. The dialog represents a numeric keypad. Default: *"VisuDialogs.Numpad"*
"Keypad"	Dialog that calls the visualization if a user activates the input field for a text at runtime. The dialog represents a keyboard. Preset: *"VisuDialogs.Keypad"*
"Use text input with limits"	Requirement: CODESYS TargetVisu or CODESYS WebVisu are configured as display variants and the *"standard text input"* is *"keyboard"*. The visualization then supports input via keyboard at runtime. The input thus generally takes place via an input field. ☑: Instead of the input field you can call a dialog that displays the value range for inputs with a limited value range. Default: *"VisuDialogs.TextinputWithLimits"*. This dialog displays the value range and doesn't accept any value outside these limits.

See also

- *Chapter 1.4.5.19.4.7 "Object 'TargetVisu'" on page 1787*
- *Chapter 1.4.5.19.4.8 "Object 'WebVisu'" on page 1788*

'Settings for user management dialogs'

You can configure your visualization with a user management. To do this, configure an input to an element that causes a user management dialog to appear. The `VisuUserManagement` library contains ready-to-use dialog visualizations for this purpose. The library is located in the installation directory, for example in `C:\Program Files (x86)\3S CODESYS\CODESYS\Projects\Visu\Dialogs\VisuUserMgmtDialogs.library`. You can also use other visualizations as user management dialogs. To do that you have to change the defaults here.	
"Login dialog"	User management dialog that enables logging in; typically a request to enter a username and a password. It appears upon an input event on an element that executes as a consequential action *"User management"*, action *"Login"*. Preset: `VisuUserManagement.VUM_Login`
"Change password dialog"	User management dialog that enables a password to be changed; typically a request to enter the current password and a new one. It appears upon an input event on an element that executes as a consequential action *"User management"*, action *"Change user password"*. Preset: `VisuUserManagement.ChangePassword`
"Change configuration dialog"	User management dialog that enables a configuration change of the user management, i.e. typically a display of the current user configuration and a possibility to change it. It appears upon an input event on an element that executes as a consequential action *"User management"*, action *"Open user configuration"*. Preset: `VisuUserManagement.VUM_UserManagement`

See also

- *Chapter 1.4.5.5 "Setting Up User Management" on page 1282*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

'Additional settings'

"Activate multitouch handling"	☑: At runtime the visualization expects user inputs via gestures and touch events. Elements concerned ● Elements with input configuration ● Element of the type *"Frame"* ● Component of the type *"Tab control"*
"Activate semi-transparent drawing"	☑: The visualization draws the elements in a semi-transparent color. To do this you can additionally specify a graduation value for the transparency when defining a color. The transparency is defined in the *"Transparency"* property. The leading byte is evaluated in color variables. Preset: Activated. Requirement: you create a new visualization and the display variants can paint semi-transparently.
"Activate standard keyboard handling"	● *[Tab]* ● *[Shift] + [Tab]* ● *[Input]* ● *[Up arrow]* ● *[Down arrow]* ● *[Right arrow]* ● *[Left arrow]*

See also

- *Chapter 1.4.5.3.3 "Assigning a color" on page 1258*
- *Chapter 1.4.5.19.1 "Keyboard Shortcuts for Default Keyboard Action" on page 1717*

'Extended settings'

Table 344: "Memory settings"

"Size of memory for visualization"	Memory size in bytes allocated by the visualization at runtime. Preset: *"400000"*
"Size of the paintbuffer (per client)"	Memory size in bytes allocated by the visualization per display variant and used for painting actions. Preset: *"50000"*

Table 345: "File transfer mode"

"Transfer visualization files to the PLC"	◉: When downloading the application from the visualization directories to the controller, CODESYS copies image files and text list files that the visualization references. A CODESYS TargetVisu needs the files on the PLC and similarly the dialogs that support a numerical input or a keyboard input.
Use local visualization files	◉: The visualization uses image files and text list files from local directories Note: In order to achieve that the visualization can access the files stored locally, it is necessary thatthat the file paths are **relative**. The paths are given in dialog box *"Project ➔ Project settings"* in tab *"Visualization"*. Note: It is also necessary that the link type of a image is *"Link to file"*. The link type is specified in the image pool.

See also

- *Chapter 1.4.5.16.3 "Configure File Transfer Mode" on page 1359*
- *Chapter 1.4.5.19.3.13 "Dialog 'Project Settings' - 'Visualization'" on page 1766*

- *Chapter 1.4.5.19.3.9 “Dialog Box 'Options' - 'Visualization'” on page 1763*
- *Chapter 1.4.1.20.2.13 “Object 'Image Pool'” on page 873*

Display variant denotes the type of visualization, for example WebVisu, TargetVisu. A visualization client is a currently connected display medium. Thus, several browsers (clients) can be connected in parallel for the display variant WebVisu.

Table 346: “Client settings”

“Maximum number of visualization clients”	Limits the number of visualization clients that are executed at the same time. If you configure the elements so that they vary depending on the display variant, then you have to limit the number of display variants. A visualization is given an ID at runtime that identifies the display variant and then processes data accordingly. CODESYS can the query the ID using the system variable `CURRENTCLIENTID` and thus obtains the information as to which of the running variants is concerned. Example: `arr[CURRENTCLIENTID].dwColor` Requirement: `VisuGlobalClientManager` library is integrated in the project. Tip: You can find in the CODESYS store. example "Global Client Manager"

“Transfer both svg images and converted images”	This option is visible only if both a WebVisu and a TargetVisu exist. It concerns images in svg format only. The option is available if the device description for the controller of the TargetVisu does not support the svg (full) format. ☑: The images are transmitted in the png or bmp formats (for TargetVisu) and additionally in svg format (for WebVisu).

Not all settings are available with an integrated CODESYS visualization.

Tab 'Visualization Manager' - 'Default Hotkeys'

Symbol: 

The tab includes a list of configured keyboard shortcuts that are valid for all visualizations available throughout the application. Therefore, the tab is the central location for defining keyboard shortcuts for all visualizations that are below an application.

Keyboard shortcuts of the default keyboard action are **not** listed here.

The tab is similar to the *“Keyboard Configuration”* tab and provides the same editing options.

See also

- *“Tab 'Keyboard configuration'” on page 1720*

Tab 'Visualization manager' – 'Visualizations'

Symbol: 

Function: The tab lists all visualizations that are available project-wide and enables an assignment of the visualizations for the loading behaviour, depending on the display variants.

Tab 'Flags'

"Standard behaviour"	☑: The visualizations of the application and the actually referenced visualizations are automatically loaded to the target system. The activated checkboxes show which one that is. ☐: The loading behaviour is explicitly defined for each visualization. Hint: use the explicit selection if you reference visualizations indirectly via IEC variables.
"Visualizations"	The list contains all created visualizations from the device tree and the POU view.
"Dialogs"	The list contains all the referenceable visualizations that are available via the libraries of the library management.

Only those visualizations selected here using checkboxes are loaded.	
"Remote target visualization, target visualization, web visualization,"	The column settings affect the loading behaviour for the display variants *"remote target visualization"*, *"target visualization"* and *"web visualization"*.

Tab 'Visualization manager' - 'User management'

Symbol: 

The *"User management"* tab is used for creating and configuring the user management for visualizations and their users and groups.

If a user management has not been configured yet, then the following buttons are available:	
"Create empty user management"	The user management opens. The *"None"* group is created.
"Create user management with default groups and users"	The user management opens. The following groups and users are created: ● *"Admin"* group with *"Admin"* user ● *"Service"* group with *"Service"* user ● *"Operator"* group *"Operator"* user ● *"None"* group

Project with multiple visualization user managements

Table 347: "Choose between local and remote user management"

Requirement: The project includes several devices with a visualization user management.	
"Use local user management"	The user management of this visualization manager is used for the visualization.
"Use remote user management"	Drop-down list with all devices of the project that have their own visualization user management.

Visualization user management on the controller for a visualization without a display variant

Even if there are no display variants of the visualization in the application, it may be required that the visualization user management is located on the controller. This is the case, for example, when HMIs connect to the controller.

Requirement: The visualization does not have any display variants. This means that the objects *"Web visualization"*, *"Target visualization"*, or *"Remote target visualization"* are inserted below the visualization manager.

See also

- ↳ *Chapter 1.4.5.5.2 "Configuring users and groups" on page 1283*

Tab 'Groups'

"Group name"	When you click the node, all users are listed that belong to the group.
"Automatic logout"	☑: The *"Logout time"* input field is active and editable.
"Logout time"	Input field for integer value Drop-down list for time unit *"Min"*, *"Sec"*, or *"Hr"*
"Permission to change user data"	☑: The group is granted permission to edit user data when the visualization is in online mode.
"Description"	The text is visible in the development system only. It is not downloaded to the controller.
"ID"	Unique ID for each group. Assigned automatically by the system.
Add a new group	In the last row of the table, click in the *"Group name"* field and specify the name for the new group.
Delete a group	Select a group and press *[Del]*. The *"None"* group cannot be deleted.

Table 348: Buttons

"Update visualizations / hotkeys"	Opens the *"Update visualizations and hotkeys"* dialog box. Update, if groups were changed at a time when visualizations or keyboard shortcuts already had restricted permissions.
"List usage of groups"	List of visualizations and keyboard shortcuts with restricted permissions. The list is displayed in the *"Messages"* view.
"Export groups for global visualizations"	The defined group names are transmitted to *"Tools ➔ Options ➔ Visualization user management"*. They are then listed in *"Use the following user group list for the visualization"*. The list can be changed there as well.
"Delete complete user management"	The user management is deleted and the start view is shown with the following buttons: *"Create empty user management"* and *"Create user management with default groups and users"*.
"Export user management"	The drop-down list opens. ● *"Before V3.5 SP6"* ● *"V3.5 SP6 and later"* A standard dialog opens for saving the user management as a CSV file with any name in any directory.

“Import user management”	A standard dialog opens for importing a user management. the user management must be a CSV file.
↑, ↓	Moves the selected group one line up/down, thus changing the hierarchy of the group. A group of a higher hierarchy cannot have fewer permissions for an element than a group of a lower hierarchy.

Table 349: Dialog box “Update visualizations and hotkeys”

This dialog updates only visualization elements and keyboard shortcuts with configured permissions.	
“Add new group”	Drop-down list with all new created groups of this user management. Requirement: A new user group was created.
“Setting for new group”	● *“new group in visualization / hotkey will get the right like group”*: Drop-down list with all existing groups of this user management ● *“new group should get the following right”* – *“for visualization elements”*: Drop-down list with the permissions: *“Operable”*, *“Only visible”*, and *“Invisible”*. – *“for hotkeys”*: Drop-down list with the permissions: *“Operable”*, *“Not operable”*.
“Delete not existing groups”	If no affected visualization elements or keyboard shortcuts are found for updating, then this is displayed as a message in the *“Messages”* view (*“Visualization”* category).
“Rename groups”	
“Update”	Updates the permissions of the affected visualization elements and keyboard shortcuts

Tab 'User'

“Login name”	Name for the user to log in to the visualization at runtime. This name is unique.
“Full name”	This name may exist more than one time in the user management.
“ Password”	Encrypted by CODESYS. By default, the *“Login name”* is displayed here. If you click the *“Password”* field of a selected line, then the *“Change password”* dialog box opens.
“User group”	Group(s) that the user belongs to. Clicking the *“User group”* field of a selected user opens the dialog box *“User groups the user belongs to”*. ● *“Groups”* ● *“Assigned”*: ☑: The user is assigned to this group.
“Deactivate”	☑: The user is deactivated.
“Description”	Descriptive text is available in the development system only and is not downloaded to the controller.

Table 350: Buttons

“Upload user from device”	The data of the user management is uploaded from the controller. If user data is already configured. then it is overwritten.
“Download user to device”	The data of the user management is downloaded to the controller. The existing user management on the controller is overwritten.

“Export user management”	The drop-down list opens. ● *“Before V3.5 SP6”* ● *“V3.5 SP6 and later”* A standard dialog opens for saving the user management as a CSV file with any name in any directory.
“Import user management”	A standard dialog opens for selecting the user management (in CSV format) from the file system.

Tab 'Settings'

“Settings for download of user data”	
“Download user data on every login”	The data of the user management is downloaded to the controller at login. Existing data is overwritten.
“Never download user data on login”	The data of the user management is never downloaded to the controller, even if it changes.
“Allow decision on every download”	A *“Warning”* dialog box opens for you to accept or refuse the download.
“Access rights for elements”	
“Use group hierarchy”	☑: The permissions can be granted to the group hierarchy of the *“Groups”* tab only. The group in the first line of the *“Group”* list is the highest in the hierarchy. A group of a higher hierarchy cannot have fewer permissions for an element than a group of a lower hierarchy.
“Logout behavior”	
“Change to start visualization at logout”	☑: Switches at logout to the visualization that is configured as the *“Start visualization”* in the respective display variant.

CSV file with the data for user management

The data for user management is saved to a CSV file in the following format:

- User groups: `ID;group name; automatic logoff TRUE/FALSE;logoff time;unit logoff time;permission to change user date TRUE/FALSE`
- Users: `login name;full name;password encrypt TRUE/FALSE;password;group ID;user deactivated TRUE/FALSE`

Use this format when you want to edit data for user management by means of any tool. If you set `password encrypt` to `FALSE`, then an unencrypted password can be used. In the example, the unencrypted password `Yellow` was specified for the user `Hugo`. If you import the CSV file with the command *“Import user management”*, then the password is encrypted automatically.

Example

```
V1.0.0.1
Usergroups:
1;Admin;TRUE;1;Minute;TRUE
3;Operator;FALSE;1;Minute;FALSE
7;Service;FALSE;1;Minute;FALSE
0;None;FALSE;1;Minute;FALSE
4;Early and late shift;FALSE;1;Minute;FALSE
2;Early shift;TRUE;1;Minute;FALSE
6;Late shift;FALSE;1;Minute;FALSE
User:
Service;Service;TRUE;C08298D42A35732CFFB7DF43771B7607;2;FALSE
Operator;Operator;TRUE;3D94AB9540B025B07773DE7037F19837;3;FALSE
John;Blue;TRUE;62ED5DE29E5DD4164A01F3AF1B81EFA0;4;FALSE
Paul;White;TRUE;01E2CBD4AE5442D9EACE33669549A3CC;2;FALSE
Hugo;Green;FALSE;Yellow;6;FALSE
```

See also

- *Chapter 1.4.5.5.2 “Configuring users and groups” on page 1283*
- *Chapter 1.4.5.5.4 “Configuring permissions for groups” on page 1285*

Tab 'Visualization Manager' - 'Font'

Symbol: 

Function: This tab provides settings for adapting the font and font size in the visualization according to the language. The settings apply to all visualizations of the application, including the visualization manager.

Table 351: “Language Specific Font Settings”

“Language”	Language used in the project. A column is created for each language. All text lists, including those from integrated libraries, are scanned for this.
“Font”	Font used by the visualization depending on the language.
“Size factor”	The factor affects the type size of all texts in the visualization. Preset: 1 If the factor is smaller than 1, this leads to a reduction of the type size. If the factor is 1, all texts are displayed unchanged as defined in *“Properties”*.
Red highlighting of a cell	The highlighted language is no longer present in the text lists of the project or the libraries. This highlighting is not available in runtime mode.

Context menu of a selected table row	
“Delete Language”	The associated column is removed. This is advisable above all if settings in the column are highlighted in red.
“Copy Language Settings”	All settings in the column are copied to the clipboard.
“Paste Language Settings”	All settings in the column are overwritten with the values from the clipboard.

Table 352: “General Font Settings”

“Automatic decrease of font size”	☑: If the text to be displayed does not fit in the text field in the set format, then the font size is decreased automatically until the text fits completely in the text field. Tip: This prevents a text from being truncated when changing to a language that needs more space. The requirement is that a font is available which has a sufficiently small font.

See also

- *Chapter 1.4.5.6 “Setting Up Multiple Languages” on page 1286*

Object 'TargetVisu'

Symbol:

Function: The object is used for configuring CODESYS TargetVisu in order to display the visualization directly on the controller of an integrated or connected panel.

Requirement: The CODESYS control runtime environment is equipped with the CODESYS TargetVisu component. The object itself is inserted below the visualization manager.

CODESYS TargetVisu can be executed on different platforms, from embedded controllers to powerful PC-based systems on different operating systems. Therefore, it can be run on Windows, Windows Embedded CE, Linux, QNX, or VxWorks. A ready-made adaptation to the graphics interface of the systems is available on these operating systems. An adaptation is required for embedded controllers or other operating systems. In addition, there are device manufacturers that integrate visualizations into external applications by means of ActiveX controls.

“Start Visualization”	Name of the visualization where the start is displayed as CODESYS TargetVisu. Hint: Use input assistance for selecting another visualization.
“Update rate (ms)”	Refresh rate (in milliseconds) in the visualization Example: `200`
“Show used visualizations”	The link opens the *“Visualizations”* tab in the *“Visualization manager”* editor. The tab provides information of the visualizations loaded on the display variants.

See also

- *Chapter 1.4.5.19.4.4 “Tab 'Visualization manager' – 'Visualizations'” on page 1781*

Table 353: “Scaling Options”

“Fixed”	◉: Fixed size of the visualization (original size).
“Isotropic”:	◉: The size of the visualization is adapted to the dimensions of the display device, retaining the proportions of the visualization.
“Anisotropic”:	◉: The size of the visualization is adjusted to the size of the display device, for example a screen.
“Use scaling options for dialogs”	☑ The dialogs, also for keypad and numpad, are scaled like the visualization (drawn with the same scaling factor). This is an advantage when a dialog was created to match the visualization because then they are scaled together.
“Use automatically detected client size”	☑: The visualization fills the screen of the display device completely.
“Use specified client size”	☑: The values in *“Client height”* and *“Client width”* are used for the size of the visualization. The visualization fills this screen area only.

“Client height”	Height of the visualization (in pixels).
“Client width”	Width of the visualization (in pixels).

Table 354: “Presentation Options”

“Antialiased drawing”	☑: Antialiasing is used in the visualization editor for drawing a visualization as a TargetVisu and a TargetVisu variant. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness (see element property *“Absolute movement”*, option *“Use REAL values”*). Requirement: The platform in use supports using REAL coordinates.

Table 355: “Default Text Input”

“Input with”	
“Touchscreen”	Text input on the display variant with touchscreen. The keypad or numpad dialog opens.
“Keyboard”	Text input on the display variant with an ordinary keyboard or a virtual keyboard (on Linux for example)
Effect: When you configure a user input for default text input, select an input configuration for input action *“Write variable”*, and configure the *“Input type”* as *“Default”*, then the settings are used here.	

See also

- *Chapter 1.4.5.4 “Configuring user inputs” on page 1267*
- *“Input action 'Write Variable'” on page 1757*

Object 'WebVisu'

Symbol: 

Function: The object is used to configure the web-based display variant for remote display of the visualization of the controller in a web browser. This allows for remote access, remote monitoring, as well as service and diagnostics of an application over the Internet.

Requirement: The object is inserted below the Visualization Manager, and the target system has a web server with CODESYS WebVisu support. The web server allows for the communication between the target system and the web browser.

See also

- *Chapter 1.4.5.16.1 “Executing as CODESYS WebVisu” on page 1355*

“Start Visualization”	Name of the visualization where the start is displayed as CODESYS WebVisu. Hint: Use the Input Assistant to select another visualization.
“Name of the .htm file”	Base URL of the web page. The URL is also specified as the address in the web browser. Example: `http://localhost:8080/webvisu.htm` Note: If you use a BeagleBone Black as a visualization device, then you have to note that a BeagleBone Black uses port 9090 for its web server. A valid IP address is as follows: `http://192.168.7.2:9090/webvisu.htm`

"Use as default page"	☑: The page specified in *"Name of .htm file"* is preset as the default page. Now this page will always open when a user specified in the web browser the IP address and port of the web server that is running on the controller: `http://<IP address web server>:<port web server>`. Example: `http://localhost:8080` Notice: Even if you have created multiple web visualizations, you can activate this option for exactly one web page only and therefore preset only one page as the default page.
"Update rate (ms)"	Refresh rate (in milliseconds) in the web browser
"Default communication buffer size"	Default size for communication buffer (in bytes). Defines the maximum available memory for data transfer between the web client and the web server. Example: `50000`
"Show Used Visualizations"	The link opens the *"Visualizations"* tab in the *"Visualization Manager"* editor. The tab provides information about the visualizations downloaded to the display variants.

See also

- *Chapter 1.4.5.19.4.4 "Tab 'Visualization manager' – 'Visualizations'" on page 1781*
- *Chapter 1.4.5.16.1 "Executing as CODESYS WebVisu" on page 1355*

Table 356: "Scaling Options"

"Fixed"	◉: Fixed size of the visualization. The values used are *"Client height"* and *"Client width"*.
"Isotropic"	◉: The size of the visualization is adapted to the dimensions of the web browser, retaining the proportions of the visualization.
"Anisotropic"	◉: The size of the visualization is adapted to the web browser.
"Use scaling options for dialogs"	☑ The dialogs (also for keypad and numpad) are scaled as the visualization (drawn with the same scaling factor). This is an advantage when a dialog was created to match the visualization because then they are scaled together.
"Client height"	Height of the visualization (in pixels).
"Client width"	Width of the visualization (in pixels).

Table 357: "Presentation Options"

"Antialiased drawing"	☑: Antialiasing is used when drawing the visualization in the web browser.

Table 358: "Input handing options"

"Standard text input with"	• *"Touchscreen"*: Text input on the WebVisu with touchscreen. The keypad or numpad dialog opens. • *"Keyboard"*: Text input on the WebVisu with an ordinary keyboard or a virtual keyboard (on Android OS for example) Effect: When you configure a user input for default text input, select an input configuration for input action *"Write Variable"*, and configure the *"Input type"* as *"Default"*, then the settings are used here.
"Treat touch as mouse actions"	☑: On devices with a touchscreen, gestures are treated as mouse actions. This option is required, for example, to operate a slider or scrollbar on a touch device.

See also

- *Chapter 1.4.5.4 “Configuring user inputs” on page 1267*
- *“Input action 'Write Variable'” on page 1757*

1.4.5.19.5 Visualization Elements

Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'

Symbol:

Category: *"Basic"*

The *"Rectangle"*, *"Rounded Rectangle"*, and *"Ellipse"* are the same type of element. They can be converted into another element type by changing the *"Element type"* property.

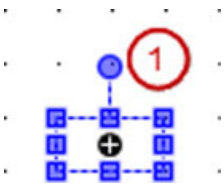
Element properties

"Element name"	Optional Example: `Werkstueck_3` Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Rectangle"*, *"Rounded Rectangle"*, *"Ellipse"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Y"	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Width"	Specified in pixels Example: `150`
"Height"	Specified in pixels Example: `30`
	Tip: You can change the values in *"X"*, *"Y"*, *"Width"*, and *"Height"* by dragging the corresponding symbols to another position in the editor.

"Angle"	Static angle of rotation (in degrees) Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Radius setting

Visible only when *"Rounded Rectangle"* is selected in the *"Type of element"* property.

"Radius"	Rounding of the corners. *"From style"* *"Relative to the element size"* *"Explicit"*: Allows for specifying a custom value in the *"Value"* setting.
"Value"	Radius of the rounded corners (in pixels) Example: `5` Requirement: *"Explicit"* is selected in the *"Radius"* setting.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

"Normal state"	The normal state is in effect if the variable in *"Color variables ➔ Toggle color"* is not defined or it has the value `FALSE`.
"Frame color"	Frame and fill color for the corresponding state of the variable.
"Fill color"	
"Transparency"	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
"Alarm state"	The alarm state is in effect if the variable in *"Color variables ➔ Toggle color"* has the value `TRUE`.
"Use gradient color"	☑: The element is displayed with a gradient of two colors.
"Gradient setting"	The *"Gradient editor"* dialog box opens.

See also

- ↳ *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Fill attributes"	The way in which the element is filled. ● *"Filled"*:The element is filled with the color from property *"Colors ➔ Fill color"*. ● *"Invisible"*: The fill color is invisible.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- ↳ *" Element property 'Appearance variables'" on page 1854*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1797*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

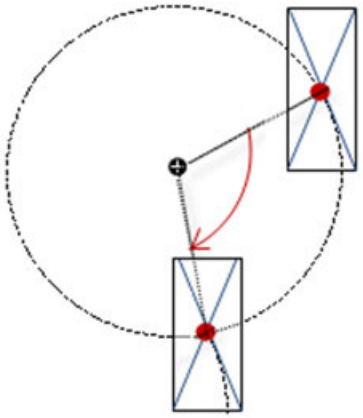
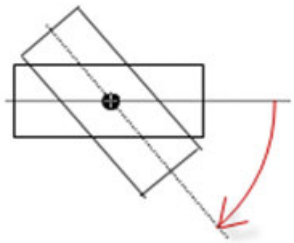
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 1795*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1794*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

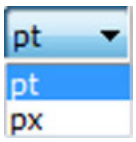
“Text list”	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
“Text index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
“Tooltip index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1795*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

“Toggle color”	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *“Color”* property. ● `TRUE`: The element is displayed with the color specified in the *“Alarm color”* property. Assignment options: ● Placeholder for the user input variable – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
“Normal state” *“Alarm state”*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *“Color variables”*, *“Toggle color”* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *“Colorvariables”*, *“Toggle color”* has the value `TRUE`.
“Frame color”	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
“Filling color”	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the “Activate semi-transparent drawing” option of the visualization manager is selected.

Select the “Advanced” option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Element property 'Appearance variables'

The properties contain IEC variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels).
“Fill attributes”	Variable (DWORD). Controls whether the fill color of the element is visible. ● Variable value = `0`: Filled ● Variable value > `0`: Invisible; no fill color
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible; no line

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1806*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *“Configure”* button opens the *“Input Configuration”* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *“Execute ST Code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog.
“OnMouseClick”	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the mouse button.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Line'

Symbol:

Category: *“Basic”*

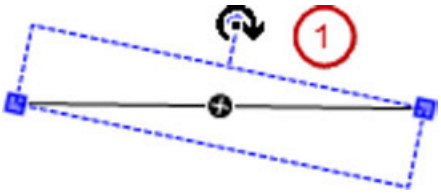
The element draws a simple line.

Element properties

“Element name”	Optional. Example: `Separator_Header` Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Line”*

Element property 'Position'

The following properties define the position and length of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“Dots”	*“[0]”*: Coordinates of the starting point *“[1]”*: Coordinate of the end point You can also change the values by dragging the box symbols () to other positions in the editor.
“Angle”	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow  , you can rotate the element about its center as a handle. Example:  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *“Absolute movement ➔ Internal rotation”*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the  symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color of the line in normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color of the line in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- ↳ *" Element property 'Appearance variables'" on page 1854*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1809*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

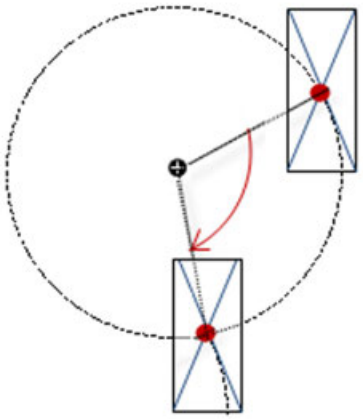
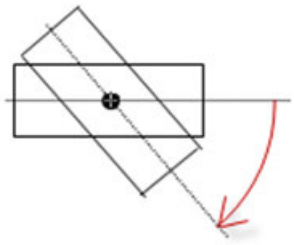
Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement point[0]” ● *“X”* ● *“Y”*	Variable (numeric data type). It contains the number (in pixels) that the starting point of the line is moved. Incrementing the X value moves the element to the right. Incrementing the Y value moves the element to the down.
“Movement point[1]” ● *“X”* ● *“Y”*	Variable (numeric data type). It contains the number (in pixels) that the end point of the line is moved. Incrementing the X value moves the element to the right. Incrementing the Y value moves the element to the down.

See also

- *“Element property 'Absolute movement'” on page 1807*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1807*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

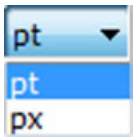
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1807*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable in the alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*
- *Chapter 1.4.5.19.4.2 "Object 'Visualization manager'" on page 1777*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

Element property 'Line width variable'

Dynamic definition of the weight of a line element using a variable.

"Integer value "	Variable (integer data type). Defines the line weight of the element (in pixels). This overwrites the fixed value that is defined in *"Appearance ➔ Line weight"*. Note: The value 0 codes the same as 1 and sets the line weight to one pixel.

Element property 'Line style variable'

"Integer value "	Variable (integer data type). Defines the appearance of the line at runtime. ● 1: Solid ● 2: Dashes ● 3: Dots ● 4: Dash Dot ● 5: Dash Dot Dot ● 6: Invisible: The line is not drawn.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Polygon', 'Polyline', 'Bézier Curve'

Symbol:

Category: *“Basic”*

The *“Polygon”*, *“Polyline”*, and *“Bézier Curve”* are the same element type. They can be converted into another type by changing the *“Element type”* property.

Elements can be dragged to the editor. The element is then drawn with five points: [0] to [4].

Other positions are added as follows: Move the mouse pointer over a corner point; the mouse pointer changes shape. Now if you press and hold *[Ctrl]* and click the left mouse button, another point is created. You can delete a point by pressing and holding *[Shift]+[Ctrl]* and click the selected point.

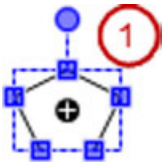
As an alternative, you can select the element in the toolbox area and in the editor click multiple times. At the same time, a connecting line is drawn from one point to the other. End by double-clicking the element or right-clicking it one time.

Element properties

“Element name”	Optional. Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Werkstueck_1`
“Type of element”	● *“Polygon”* ● *“Polyline”* ● *“Bézier Curve”*

Element property 'Position'

The following properties define the position of the corner points in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"Dots"	[0]..[n]: Coordinates of the corner points Specified in pixels You can also change the values by dragging the box symbols (⊞) to other positions in the editor.
"Angle"	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow ↻, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

"Normal state"	The normal state is in effect if the variable in *"Color variables ➔ Toggle color"* is not defined or it has the value `FALSE`.
"Frame color"	Frame and fill color for the corresponding state of the variable.
"Fill color"	
"Transparency"	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
"Alarm state"	The alarm state is in effect if the variable in *"Color variables ➔ Toggle color"* has the value `TRUE`.
"Use gradient color"	☑: The element is displayed with a gradient of two colors.
"Gradient setting"	The *"Gradient editor"* dialog box opens.

See also

- *Chapter 1.4.5.19.3.5 “Dialog 'Gradient Editor'” on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

“Line width”	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Fill attributes”	The way in which the element is filled. ● *“Filled”*:The element is filled with the color from property *“Colors ➔ Fill color”*. ● *“Invisible”*: The fill color is invisible.
“Line style”	Type of line representation ● *“Solid”* ● *“Dashes”* ● *“Dots”* ● *“Dash Dot”* ● *“Dash Dot Dot”* ● *“not visible”*

You can assign variables in the “Appearance variables” property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- *“ Element property 'Appearance variables'” on page 1854*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Text”*.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Tooltip”*.

See also

- *“Element property 'Text variables'” on page 1822*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

Element property 'Text properties'

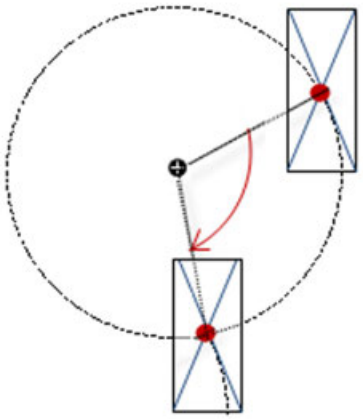
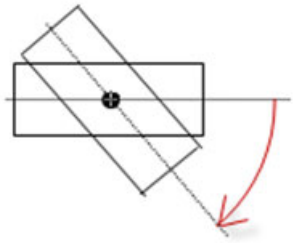
The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Dynamic points'

“Array of points”	Variable (`POINTER TO`). Points to an array of the structure `VisuElems.VisuStructPoint`. The elements `iX` and `iY` of `VisuStructPoint` contain the xy-coordinates of a point The current number of array elements implicitly contains the variable in the property *“Number of points”*. The variable that is assigned to the property *“Number of points”* contains the number of array elements and therefore the number of corner points. Example: `pPoints : POINTER TO ARRAY[0..100] OF VisuElems.VisuStructPoint;`
“Number of points”	Variable (integer data type): Contains the number of array elements and therefore the number of corner points for displaying the element. Example: `PLC_PRG.iNumberOfPoints := 24;` In the example, the element has 24 points. This definition is necessary because the individual points are defined by a pointer and this does not allow control over the number of points. Note: In this way, it is possible to adapt the display of the element dynamically by updating the number of corner points.

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1819*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

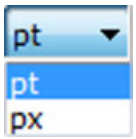
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in "Text properties".

See also

- *"Element property 'Text properties'" on page 1820*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assignment options: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Normal state" *"Alarm state"*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *"Color variables"*, *"Toggle color"* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *"Colorvariables"*, *"Toggle color"* has the value `TRUE`.
"Frame color"	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
"Filling color"	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Appearance variables'

The properties contain IEC variables for controlling the appearance of the element dynamically.

"Line width"	Variable (integer data type). Contains the line weight (in pixels).
"Fill attributes"	Variable (DWORD). Controls whether the fill color of the element is visible. ● Variable value = `0`: Filled ● Variable value > `0`: Invisible; no fill color
"Line style"	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible; no line

Fixed values can be set in the "Appearance" property. These values can be overwritten by dynamic variables at runtime.

See also

- *"Element property 'Appearance'" on page 1847*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *“Configure”* button opens the *“Input Configuration”* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *“Execute ST Code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog.
“OnMouseClick”	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the mouse button.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

“Hotkey”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Events”* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
“Key”	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
“Events”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the “Keyboard Configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Pie'

Symbol:

Category: *“Basic”*

The element draws a pie of any angle.

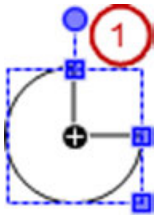
Element properties

“Element name”	Example: `Error_rate_part_1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Pie”*

Element property 'Position'

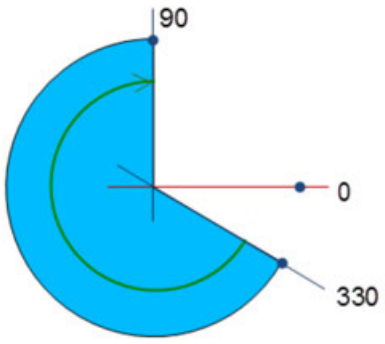
The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Y”	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
“Width”	Specified in pixels Example: `150`
“Height”	Specified in pixels Example: `30`
	Tip: You can change the values in *“X”*, *“Y”*, *“Width”*, and *“Height”* by dragging the corresponding symbols to another position in the editor.

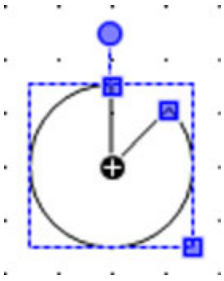
“Angle”	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *“Absolute movement ➔ Internal rotation”*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Begin”	Start angle of the pie. If you also define a variable for the start, then the start angle is calculated from the sum of the values for *“Begin”* and *“Variable for begin”*.	Example: • *“Begin”*: 330 • *“End”*: 90 
“End”	End angle of the pie. If you also define a variable for the end, then the end angle is calculated from the sum of the values for *“End”* and *“Variable for end”*. The pie is drawn clockwise from the start angle to the end angle.	
“Variable for begin”	The start of the sector is defined dynamically by a variable.	
“Variable for end”	The end of the sector is defined dynamically by a variable.	
“Only show circle line”	☑: The pie is drawn without the radius line or filling color.	

Element property 'Center'

"X" *"Y"*	Display of the center coordinates. You cannot modify these values here in the properties. If the Pie is selected in the editor, then the center of the Pie (as well as the center of the enveloping box) is visualized with the symbol ⊕. Moreover, the element is decorated with a position, begin, and end boxes that you can move.  The center coordinates change when you move the center symbol ⊕ in the editor. This also changes the size of the Pie so that the position box retains its position and the center remains in the middle of the element.

Element property 'Colors'

"Normal state"	The normal state is in effect if the variable in *"Color variables ➔ Toggle color"* is not defined or it has the value `FALSE`.
"Frame color" *"Fill color"*	Frame and fill color for the corresponding state of the variable.
"Transparency"	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
"Alarm state"	The alarm state is in effect if the variable in *"Color variables ➔ Toggle color"* has the value `TRUE`.
"Use gradient color"	☑: The element is displayed with a gradient of two colors.
"Gradient setting"	The *"Gradient editor"* dialog box opens.

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Fill attributes"	The way in which the element is filled. ● *"Filled"*:The element is filled with the color from property *"Colors ➔ Fill color"*. ● *"Invisible"*: The fill color is invisible.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- ↳ *" Element property 'Appearance variables'" on page 1854*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- ↳ *"Element property 'Text variables'" on page 1835*
- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- ↳ *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

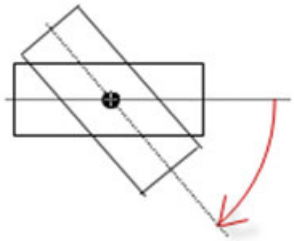
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (integer data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (integer data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.

“Interior rotation”	Variable (integer data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. If a static angle of rotation is specified in *“Position ➔ Angle”*, then the static angle of rotation and the angle of rotation are added.	

You can link the variables to a unit conversion.

The “X”, “Y”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1833*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

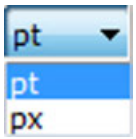
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1833*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assignment options: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Normal state" *"Alarm state"*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *"Color variables"*, *"Toggle color"* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *"Colorvariables"*, *"Toggle color"* has the value `TRUE`.
"Frame color"	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
"Filling color"	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Appearance variables'

The properties contain IEC variables for controlling the appearance of the element dynamically.

"Line width"	Variable (integer data type). Contains the line weight (in pixels).
"Fill attributes"	Variable (DWORD). Controls whether the fill color of the element is visible. ● Variable value = `0`: Filled ● Variable value > `0`: Invisible; no fill color
"Line style"	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible; no line

Fixed values can be set in the "Appearance" property. These values can be overwritten by dynamic variables at runtime.

See also

- *"Element property 'Appearance'" on page 1847*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Image'

Symbol:

Category: *"Basic"*

The element adds an image to the visualization. The displayed image is managed in the image pool and referenced in the visualization element by means of a static ID. You can also change the displayed image dynamically by using a variable instead of the static ID.

With the "Background" command, you can define a background for the entire visualization.

Directories that contain the images for use in visualizations can be defined in the project settings (category "Visualization").

Element properties

"Element name"	Example: `Status bar` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Image"*
"Static ID"	Identifier of the image file for a static assignment ID of the image file on, as it is defined in the corresponding image pool. If the image is not included in the global image pool in the POU view, then the instance path must be specified. Then the name of the image pool is preceded to make the entry unique. Example: `imagepool2.button_image`. When a new ID is specified, a file selection dialog opens. The selected file is saved to the *"GlobalImagePool"*. See also: Help for the *"Image Pool"* object.
"Show frame"	☑: The image file is displayed with a frame.
"Clipping"	Requirement: The *"Scaling type"* property is *"Fixed"*. ☑: Only part of the visualization is displayed that fits in the element frame.
"Transparent"	☑: The image pixels that have the *"Transparent color"* are displayed as transparent.
"Transparent color"	Effective only if the *"Transparent"* option is activated. The … button opens the color selection dialog. This is where you select the transparent color.

"Scaling type"	Definition of how an image fits in the element frame. ● *"Isotropic"*: The entire image is displayed in the element frame, either larger or smaller. As a result, the proportion of height and width are retained. If the alignment of the elements to each other should also be retained within a scaled frame element, then note the following. Unwanted horizontal or vertical offsets can be prevented by setting the properties *"Horizontal alignment"* and *"Vertical alignment"* to *"Centered"*. The alignment of the elements is retained and there are no resulting horizontal or vertical offsets. Example: A lamp is centered above a switch. The lamp should remain in the horizontally centered position, even if the frame is resized. ● *"Anisotropic"*: The image resizes automatically to the dimensions of the element frame, filling the entire element frame. As a result, the proportions are not retained. ● *"Fixed"*: The image retains its original size, even if the element frame is resized. Note also that the *"Clipping"* option is selected. For each reassignment of an image ID, the element size is adapted automatically to the image size.
"Horizontal alignment"	Horizontal alignment of the element within the element frame: ● *"Left"* ● *"Centered"* ● *"Right"* Requirement: The scaling type of the image is *"Isotropic"* or *"Fixed"*. Note: If the visualization is referenced, then the horizontal alignment takes effect within the frame position. : The *"Variable"* property is shown below this.
"Variable"	Enumeration variable (`ENUM VisuElemBase.VisuEnumVerticalAlignment`). Contains the horizontal alignment. Example: `PLC_PRG.eHorizontalAlignment`
"Vertical alignment"	Vertical alignment of the element within the element frame: ● *"Top"* ● *"Centered"* ● *"Bottom"* Requirement: The scaling type of the image is *"Isotropic"* or *"Fixed"*. Note: If the visualization is referenced, then the horizontal alignment takes effect within the frame position. : The *"Variable"* property is shown below this.
"Variable"	Enumeration variable (`ENUM VisuElemBase.VisuEnumVerticalAlignment`). Contains the vertical alignment. Example: `PLC_PRG.eVerticalAlignment`

Example A valid declaration is required for the variables used as an example in the table above.

Enumeration

```
TYPE VisuElemBase.VisuEnumHorizontalAlignment
    LEFT
    HCENTER
    RIGHT
END_TYPE

TYPE VisuElemBase.VisuEnumVerticalAlignment
    DOWN
    VCENTER
    BOTTOM
END_TYPE
```

Declaration

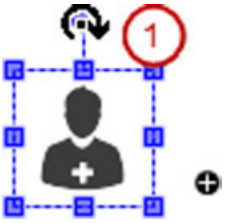
```
PROGRAM PLC_PRG
VAR
    eHorizontalAlignment :
VisuElemBase.VisuEnumHorizontalAlignment :=
VisuElemBase.VisuEnumHorizontalAlignment.HCENTER;
    eVerticalAlignment : VisuElemBase.VisuEnumVerticalAlignment :=
VisuElemBase.VisuEnumVerticalAlignment.VCENTER;
END_VAR
```

See also

- *Object 'Image Pool'*

Element property 'Position' The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Y"	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Width"	Specified in pixels Example: `150`
"Height"	Specified in pixels Example: `30`
	Tip: You can change the values in *"X"*, *"Y"*, *"Width"*, and *"Height"* by dragging the corresponding symbols to another position in the editor.

"Angle"	Static angle of rotation (in degrees). Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the frame Requirement: *"Show frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the frame in alarm state Requirement: *"Show frame"* property is activated. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- ↳ *" Element property 'Appearance variables'" on page 1854*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- ↳ *"Element property 'Text variables'" on page 1850*
- ↳ *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- ↳ *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Image ID variable'

"Image ID"	Variable (`STRING`). Contains the image ID. The contents of the string corresponds to the description of the *"Static ID"* property. Example: `PLC_PRG.stImageID := 'ImagePool_A.Image3';`

See also

- *Chapter 1.4.5.18.1.5 "Visualization Element 'Image'" on page 1418*
- *Chapter 1.4.1.20.2.13 "Object 'Image Pool'" on page 873*

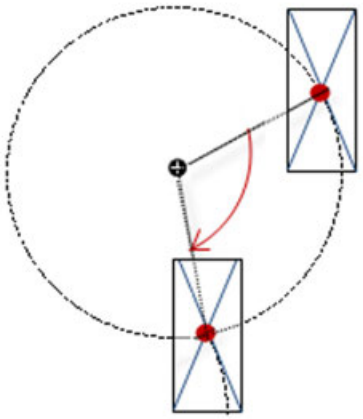
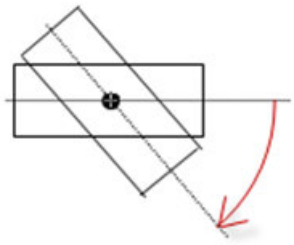
Element property 'Dynamic image'

You can use this element property for animating a series of image files.

"Bitmap version"	Variable (integer data type). Contains the version of the image. If the variable changes, then the visualization re-reads the image referenced in the *"Image ID"* property and displays it. The visualization displays animations when the image file on the controller is updated continuously, thus incrementing the version variable. The application must be programmed for this. Possible applications ● Displaying graphics that are generated by the application ● Displaying images that are refreshed by a camera

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed. 

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 1820*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1847*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

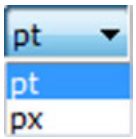
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1847*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	Color variable for the frame ● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Requirement: *"Show frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable for the frame in alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Element property 'Appearance variables'

The properties contain variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels). Note: The values 0 and 1 both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible: The line is not drawn.

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1847*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. User input is a user event from the perspective of the element.

The *"Configure"* button opens the *"Input configuration"* dialog box for creating or modifying a user input configuration.

A configuration contains one or more input actions for the respective input event. Existing input actions are displayed below it.

Example: *"Execute ST code"*: `PLC_PRG.i_x := 0;`

"OnDialogClosed"	Input event: The user closes the dialog box.
"OnMouseClick"	Input event: A user clicks the element completely. The mouse button is clicked and released.
"OnMouseDown"	Input event: A user clicks down on the element only.
"OnMouseEnter"	Input event: A user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: A user drags the mouse pointer away from the element.
"OnMouseMove"	Input event: A user moves the mouse pointer over the element area.
"OnMouseUp"	Input event: The user releases the mouse button over the element area.

See also

- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

"Hotkeys"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Event(s)"* property are triggered.
"Key"	Key pressed for input action. Example: *[T]*
"Event(s)"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed in the "Keyboard configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Project Settings - Visualization*
- *Chapter 1.4.5.19.2.10 "Command 'Background'" on page 1728*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Frame'

Symbol:

Category: *“Basic”*

The element serves as a frame in which to display one or more already existing visualizations. You get a structured user interface. The size of the frame can be fixed or scaled. The display area of the referenced visualization then adapts itself to the frame size.

Element properties

“Element name”	Example: `refVisUserInfo`
“Type of element”	*“Frame”*
“Clipping”	☑: Fixed size. Only that part of the referenced visualization that fits inside the frame is displayed. Requirement: The *“Scaling type”* property is *“Fixed”*.
“Show frame”	Displays the frame ● *“No frame”*: The displayed area of the frame does not have borders. ● *“Frame”*: The displayed area of the frame has borders. ● *“No frame with offset”*: The displayed area of the frame does not have a border and the displayed area of the referenced visualization is reduced inwards by one pixel as compared to the frame area. The gap prevents the referenced visualization from touching any adjacent elements.

“Scaling type”	The method with which the height and width of the **referenced** visualization are scaled. ● *“Isotropic”*: The visualization is scaled to the size of the element. The visualization retains its proportions with a fixed height/width ratio. ● *“Anisotropic”*: The visualization is scaled to the size of the element. The height and width are adapted to the element independently of each other. ● *“Fixed”*: the visualization is displayed in its original size without taking into account the size of the element. ● *“Fixed and scrollable”*: The visualization is displayed fixed in the element. If it is larger than the element, the element will be provided with scrollbars. Please note: assign variables to the properties *“Scroll position variable horizontal”* or *“Scroll position variable vertical”*. You can then edit the data of the scrollbar position in the application.

Element properties 'Scrollbar settings'

The properties contain variables for the position of the scrollboxes in the scrollbars. You can then edit the data of the scrollbox position in the application.

Requirement: the property *“Scaling type”* is *“fixed and scrollable”*.	

“Scroll position variable horizontal”
“Scroll position variable vertical”
	Variable (integer data type, also as array). Contains the position of the horizontal or vertical scrollbox. The array contains the position for every display variant. If the visualization runs on several display variants, then the position changes are decoupled from each other.
	Example:
	`PLC_PRG.iScrollHor[CURRENTCLIENTID]`
	`PLC_PRG.iScrollVer[CURRENTCLIENTID]`
	The variable is declared as an array in the example.
	`iScrollHor: ARRAY[0..20] OF INT;`
	`iScrollVer: ARRAY[0..20] OF INT;`
	`CURRENTCLIENTID` indexes the current display variant.

You can combine the variables with a unit conversion.

See also

- *Unit conversion*

“Deactivation of the background character”	☐: The background is drawn. The non-animated element of the referenced visualization is drawn as a background bitmap in order to optimize the performance of the visualization. Consequence: Elements can be displayed in an unexpected order at runtime. For example, an animated element can push itself behind the Frame at runtime. ☑: Background character is deactivated in order to avoid the behavior described above.

Element property 'References' Contains the currently configured visualization references as a subnode

“References”	Clicking *“Configure”* opens the *“Frame Configuration”* dialog. This is used to manage the referenced visualizations. Caution: Visualizations can be nested at any depth by means of Frame elements. In order to use the *“Switch to any visualization”* Frame selection type without any problems, a Frame must not contain more than 21 referenced visualizations. For more information, see also the description for the *“Input configuration”* of an element: Action *“Switch Frame visualization”*.
List of the currently referenced visualizations	Visualizations that have a button also have this displayed as a subnode. Each interface variable is listed with the currently assigned transfer parameters. Example: `vis_FormA` • `iDataToDisplay_1 : PLC_PRG.iVar1` • `iDataToDisplay_2 : PLC_PRG.iVar2` Hint: You can change the assignment of the variables to an interface variable here and edit the value field. Or click the *“Configure”* button instead.

See also

- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*
- *Chapter 1.4.5.15 “Creating a structured user interface” on page 1321*
- *“Input action 'Switch Frame Visualization'” on page 1756*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for the colors.

"Color"	Color of the frame ▼: Selection list with style colors appears ...: Standard dialog *"Color"* opens for selecting a color. Please note: the normal state is when the boolean variable in the property *"Color variables ➔ Toggle color"* is not defined or its value is `FALSE`.
"Alarm color"	Color with which the element is filled during the alarm state. Please note: Alarm state is when the value of the boolean variable in the property *"Color variables ➔ Toggle color"* is `FALSE`.
"Transparency"	Integer number (value range from `255` to `0`). Specifies the transparency of the associated color. `255`: The color is opaque. `0`: The color is fully transparent. Please note: If the color is a style color and already contains a transparency value, then this property is write-protected.

See also

- *Chapter 1.4.5.3.3 "Assigning a color" on page 1258*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values are defined here.

See also

- *" Element property 'Appearance variables'" on page 1867*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1863*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

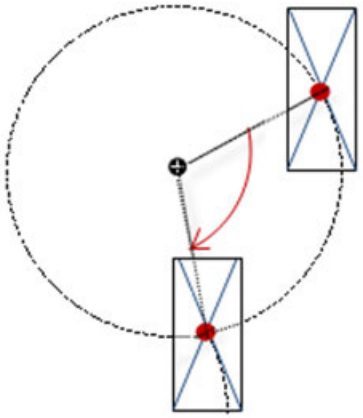
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.

You can link the variables to a unit conversion.

The properties "X", "Y", "Rotation", and "Interior rotation" are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*"Position"* property). The shape of the element can change.

"Movement top-left"	
"X"	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
"Y"	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
"Movement bottom-right"	

"X"	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
"Y"	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *"Element property 'Absolute movement'" on page 1820*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1860*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

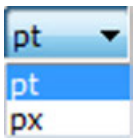
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1861*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	Color variable for the Frame ● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Requirement: *"Show Frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable for the Frame in alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Element property 'Appearance variables'

The properties contain variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels). Note: The values 0 and 1 both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible: The line is not drawn.

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 1860*

Element property 'Switch frame variable'

The variable controls the switching of the referenced visualizations. This variable indexes one of the referenced frame visualizations and this is displayed in the frame. When the value of the variable changes, it switches to the recently indexed visualization.

“Variable”	● Variable (integer data type) that contains the index of the active visualization Example: `PLC_PRG.uiIndexVisu` Hint: The *“Frame Configuration”* dialog includes a list of referenced visualizations. The visualizations are automatically numerically indexed via the order in the list. Note: This variant of switching usually affects all connected display variants. ● Array element (integer data type) for index access via `CURRENTCLIENTID` Example: `PLC_PRG.aIndexVisu[CURRENTCLIENTID]` Note: This variant of switching applies to the current client only, and therefore only on one display variant. That is the display variant where the value change was triggered (for example, by means of user input).

See also

- *Chapter 1.4.5.19.2.9 “Command 'Frame Selection'” on page 1727*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.
"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.15 "Creating a structured user interface" on page 1321*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Visualization Element 'Label'

Symbol:

Category: *"Common Controls"*

The element is used to label visualizations.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Header_Parameter`
"Type of element"	*"Label"*

Element property 'Texts'

The property requires a character string.

This text is entered automatically into the `GlobalTextList` text list and can be localized there.

"Text"	Character string (without single straight quotation marks) Example: `Main page`

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

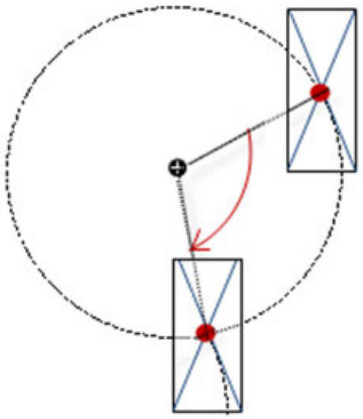
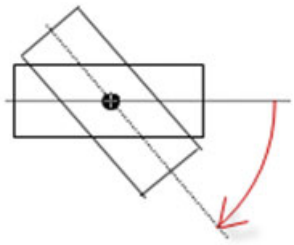
The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.

"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Combo Box, Array'

Symbol:

Category: *"Common Controls"*

The element shows values of an array as a list box. When the visualization user clicks an entry, the array index of the entry is written to an integer variable.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `List_Product_Number`
"Type of element"	*"Combo Box, Array"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	The array index of the list entry that the user clicks is saved at runtime. Property value ● Variable (integer data type) Example: `PLC_PRG.iIndexComboboxEntry` ● Enumeration variable with text list support Example: `PLC_PRG.eMyCombobox<COMBO>` Note: Value range of the enumeration value that lies within the `DWORD` or `DINT` value range
"Data array"	Displayed as a combo box. Every array component becomes a combo box entry. Property value ● Array variable (`ARRAY[...] OF`) Example: `PLC_PRG.astrCombobox` Declaration: `astrCombobox : ARRAY[0..4] OF STRING := ['First', 'Second', 'Third', 'Fourth'];`

See also

- *Enumerations*
- *Chapter 1.4.5.6 "Setting Up Multiple Languages" on page 1286*

Element property 'Columns'

The "Combo box – Array" element visualizes an array variable or structure variable in a tabular view. The index of array elements or structure members is shown in a column or row. Two-dimensional arrays or structure arrays are shown in several columns. You specify the visualized variable in the *"Data array"* property. If a variable is assigned there, then you can specify the display of the table columns where the array elements are shown. You can customize each column that is assigned to an index [<n>].

"Columns" ● [<n>]	Due to the structure of the variable that is defined in *"Data array"*, CODESYS determines the number of columns and defines them with the index <n>. Example: `StringTable : ARRAY [0..2, 0..4] OF STRING := ['BMW','Audi','Mercedes','VW','Fiat', '150','150','150','150','100','blue','gray','silver','blue','red'];`: three columns are formed `[0]`, `[1]` and `[2]`.
"Max. array index"	Optional. Variable (integer data type) or value. Defines up to which array index the data is displayed.
"Row height"	Height of the rows (in pixels).
"Number visible rows"	Optional. If the array is larger than the number of visible rows, then a scrollbar is included.
"Scrollbar size"	Width of the vertical scrollbar (in pixels).

Table 359: "Element property 'Columns: Column [<n>]'"

"Width"	Column width (in pixels).
"Image column"	☑: Images can be displayed in the column. Images are used from the global image pool or user-defined image pools. The image IDs are shown in the cells of the table as defined in the image pool.
"Image configuration"	

"Fill mode"	• *"Fill cell"* The image resizes to the dimensions of the cell without fixing the height/ width ratio. • *"Centered"* The image is centered in the cell and retains its proportions (height-width ratio).
"Transparency"	☑: The color that is specified in *"Transparent color"* is displayed as transparent.
"Transparent color"	When the *"Transparent"* property is enabled, the color specified here is not displayed. Pixels with this color are transparent.
"Text alignment in column"	• *"Left"* • *"Centered"* • *"Right"*

Element property 'Texts'

"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element in runtime mode Example: `Products of customer A` Hint: The text is accepted automatically into the *"GlobalTextList"* text list and can be localized there.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

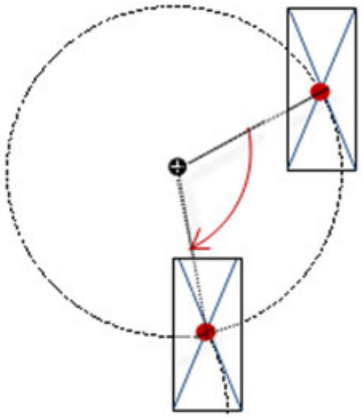
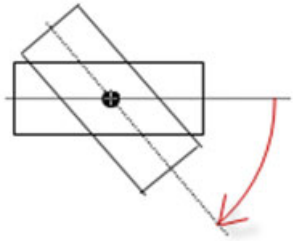
Element property 'Text properties'

The properties contain fixed values for the text properties.

"Usage of"	• *"Default style values"*: The values of the visualization style are used. • *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Font"	Example: *"Default"* ...: The *"Font"* dialog opens. ▼: List box with style fonts
"Font color"	Example: *"Black"* ...: The *"Color"* dialog opens. ▼: List box with style colors
"Transparency"	Integer (value range from `0` to `255`). This determines the transparency of the respective color. `255`: The color is opaque. `0`: The color is completely transparent. Note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Combo Box, Integer'

Symbol:

Category: *“Common Controls”*

The element shows values as a list box. When the user clicks an entry, the ID of the entry is written to an integer variable. The entries in the list box can be from a list and contain images from an image pool.

Element properties

“Element name”	Example: `List of product numbers` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Combo Box, Integer”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	At runtime, the text list ID of the list entry that the user clicks is saved at runtime. If only one image pool is displayed, then the image ID is saved. Property value ● Variable (integer data type) Example: `PLC_PRG.iIDComboboxEntry` ● Enumeration variable with text list support Example: `PLC_PRG.eMyCombobox<COMBO>`
"Text List"	Displayed as a combo box. Every text list entry becomes a combo box entry. Note: A maximum of 32766 entries can be displayed. Transfer value ● Text list identifier as string Example: `'TextList_A'` Note: The IDs of the text list have to be within the range of values of `DWORD` or `DINT`. ● Blank – When an enumeration variable with text list support is specified in the *"Variable"* property – When only one image pool is displayed
"Image Pool"	Displayed as a combo box. Every image in the image pool becomes a combo box entry. Example: `'ImagePool_A'`

See also

- *Enumerations*
- *Chapter 1.4.5.6 "Setting Up Multiple Languages" on page 1286*

Element property 'Settings of the list'

Displayed list that expands when a visualization user clicks into the element.

"Number of rows setting"	● *"From style"*: ● *"Explicit"*: Then the *"Number of visible rows"* property appears below it.
"Number of visible rows"	Number of visible lines of the combo box drop-down list defined here ● Integer literal Example: `5` ● Variable (integer data type) Example: `PLC_PRG.iNumberOfVisibleRows` Note: The property is available when the *"Number of rows setting"* property is set to *"Explicit"*.

“Row height”	● *“From style”*: ● Literal Example: `20`
“Height of image”	Image height (in pixels) of the image displayed in the drop-down list entry ● *“From style”*: ● Integer literal Example: `30` Note: Images are displayed only when a value is specified in the *“Image pool”* property.
“Width of image”	Image width (in pixels) of the image displayed in the drop-down list entry ● *“From style”*: ● Literal Example: `30` Note: Images are displayed only when a value is specified in the *“Image pool”* property.
“Offset of image”	Makes the images in the selection list appear offset (in pixels) from the left margin. An offset of `0` means that the images are displayed directly on the margin. ● *“From style”*: ● Literal Example: `4` Note: Images are displayed only when a value is specified in the *“Image pool”* property.
“Scrollbar size”	Size of the scrollbar (in pixels). The scrollbar is displayed when more entries are specified in the drop-down list than in *“Number of visible rows”*. Default: `20`

Element property 'Texts'

“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element in runtime mode Example: `Products of customer A` Hint: The text is accepted automatically into the *“GlobalTextList”* text list and can be localized there.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element property 'Value range'

"Limit valuerange"	Limits the text list to one subrange. This subrange is displayed by the combo box. Requirement: A value is specified in the *"Text list"* property. ☑: Only the subrange that is defined by the *"Minimum value"* *"Maximum value"* properties is displayed as a drop-down list.
"Minimum value"	ID of the text list entry from which a combo box entry is displayed ● Literal (`ANY_NUM`) Example: `5` ● Variable (integer data type) Example: `PLC_PRG.iFirstEntry`
"Maximum value"	ID of the text list entry up to which combo box entries are displayed ● Literal (`ANY_NUM`) Example: `10` ● Variable (integer data type) Example: `PLC_PRG.iLastEntry`
"Filter missing textentries"	☑: Text list is refreshed and any unused texts (IDs) are removed. Requirement: A value is specified in the *"Text list"* property.

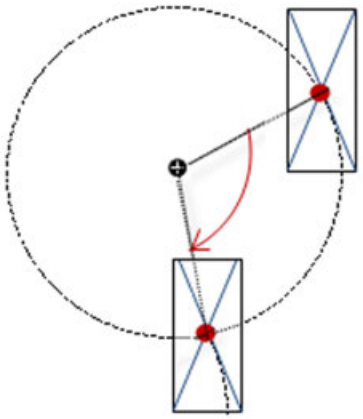
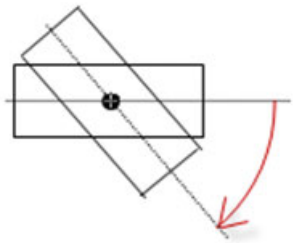
Element property 'Text properties'

The properties contain fixed values for the text properties.

"Usage of"	● *"Default style values"*: The values of the visualization style are used. ● *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- ↳ *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- ↳ *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Tabs'

Symbol:

Category: *“Common Controls”*

The element displays selected visualizations in tabs. The tabs can be used by means of the tab header without having to configure an input configuration. A visualization user switches between visualizations by clicking the tab header.

Element properties

“Element name”	Example: `Assembly A` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Tabs”*
“Tab width”	Width of the tab (in pixels). If there is not space for all tab headers, then a scroll bar is added. Example: `30`
“Tab height”	Height of the tab (in pixels) ● Integer literal Example: `15` ● *“From style”*

“Scaling type”	The method with which the height and width of the **referenced** visualization are scaled. ● *“Isotropic”*: The visualization is scaled to the size of the element. The visualization retains its proportions with a fixed height/width ratio. ● *“Anisotropic”*: The visualization is scaled to the size of the element. The height and width are adapted to the element independently of each other. ● *“Fixed”*: the visualization is displayed in its original size without taking into account the size of the element. ● *“Fixed and scrollable”*: The visualization is displayed fixed in the element. If it is larger than the element, the element will be provided with scrollbars. Please note: assign variables to the properties *“Scroll position variable horizontal”* or *“Scroll position variable vertical”*. You can then edit the data of the scrollbar position in the application.

"Deactivate background drawing"	☐: The non-animated elements of the referenced visualization are displayed as background images in order to optimize the performance of the visualization. Result: At runtime, the elements can be displayed in any order, for example when an element moves behind the frame at runtime. ☑: Deactivates the background display in order to prevent the behavior described above The property is not available for the following settings: ● The *"Scaling type"* property is set to *"Fixed and scrollable"* ● The client animation functionality is enabled.

Element property 'Scroll bar settings'

The properties include variables for the position of the scroll boxes in the scroll bars. You can process the data for the scroll box position in the application.

Requirement: The *"Scaling type"* property is *"Fixed and scrollable"*.	

"Scroll position variable horizontal"
"Scroll position variable, vertical"
	Variable (integer data type, also array). Includes the position of the horizontal or vertical scroll box. The array contains the position for each display variant. If the visualization is running on multiple display variants, then the position changes are disconnected from each other.
	Example:
	`PLC_PRG.iScrollHor[CURRENTCLIENTID]`
	`PLC_PRG.iScrollVer[CURRENTCLIENTID]`
	In this example, the variable is declared as an array:
	`iScrollHor: ARRAY[0..20] OF INT;`
	`iScrollVer: ARRAY[0..20] OF INT;`
	`CURRENTCLIENTID` indicates the current display variant.

See also

- *Unit conversion*

Element property 'References'

"References"	Clicking *"Configure"* opens the *"Frame Configuration"* dialog. You can select an existing visualization there. Selected visualization references are shown in the properties. Selected visualization references are listed here as subordinate properties.
Name pf the visualization reference (example: `PLC_PRG.S1`)	
"Heading"	Tab caption (example: `Panel`)
"Image ID"	Image ID in the theme `<image pool name>.<ID>` Example: `Imagepool_A.1` for the image with ID `1` in `Imagepool_A`
Interface parameter of the visualization reference Example: `iX`	If the visualization has an interface, then their parameters are displayed here as subordinate properties. Variable (data type conforms to data type of the interface parameter). Includes the initialization value for the instantiation of the visualization.

See also

- *Chapter 1.4.5.15 “Creating a structured user interface” on page 1321*
- *Chapter 1.4.5.19.2.1 “Command 'Interface Editor'” on page 1719*
- *Chapter 1.4.5.19.2.9 “Command 'Frame Selection'” on page 1727*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Switch frame variable'

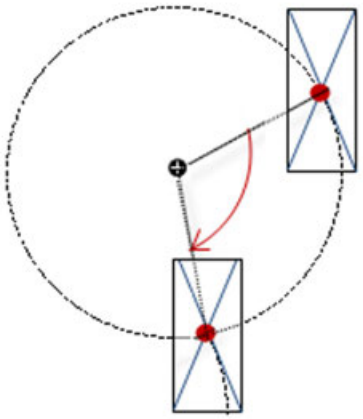
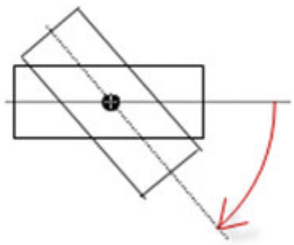
"Variable"	Variable (integer data type). Specifies the index of the active visualization. Example: `PLC_PRG.uiActiveVisuID`. Tip: The *"Frame Configuration"* dialog box includes a list of selected visualizations. The visualizations are ordered automatically in numeric order in the list.

See also

- *Chapter 1.4.5.19.2.9 "Command 'Frame Selection'" on page 1727*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Button'

Symbol:

Category: *"Common Controls"*

The element triggers an action, such as setting a variable.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Voltage_on`
"Type of element"	*"Button"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the element in its normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the element in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.
"Use gradient color"	☑: The element is displayed with a color gradient.
"Gradient setting"	The *"Color gradient editor"* dialog box opens.

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Image'

"Static ID"	Reference to an image in an image pool of the format <name of image pool>.<image ID> (example: `image_pool.GreenButton`). If the image is from the *"GlobalImagePool"*, then you can omit the name of the image pool because CODESYS always searches this pool first. ...: The *"Input Assistant"* dialog box opens and lists all available image pools and images in the entire project.
"Scale type"	Behavior of the image when resizing the button. ● *"Isotropic"*: The image retains its proportions. The ratio of height to width is retained, even if you change the height or width of the button separately. ● *"Anisotropic"*: The image resizes to the dimensions of the button. ● *"Fixed"*: The image retains its original size, even if you change the size of the button.
"Transparency"	The visualization displays the image with the transparency color that is selected in *"Transparency color"*.
"Transparency color"	Color that is transparent in the image (example: *"White"*). if the image background that is reference by *"Static ID"* is white, then this background is displayed transparent. Clicking ... opens a color selection dialog. Requirement: The *"Transparency"* option is activated.
"Horizontal alignment"	Horizontal alignment of the image ● *"Left"* ● *"Centered"* ● *"Right"*
"Vertical alignment"	Vertical alignment of the image ● *"Top"* ● *"Centered"* ● *"Bottom"*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *“Element property 'Text variables'” on page 1897*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

Element property 'Text properties'

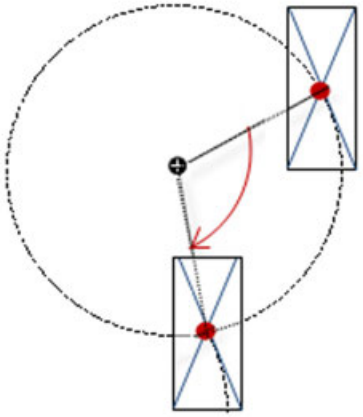
The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 1820*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 1894*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

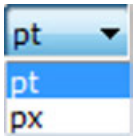
Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

“Text list”	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
“Text index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
“Tooltip index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

Element property 'Font variables' The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1895*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Color"	Color variable for the Frame ● Variable (`DWORD`) for the color Example: `PLC_PRG.dwColor` ● Color literal Example of gray and opaque: `16#FF888888` Requirement: *"Show Frame"* property is activated. Please note that the normal state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color variable for the Frame in alarm state ● Variable (`DWORD`) for the alarm color Example: `PLC_PRG.dwAlarmColor` ● Color literal Example of red and opaque: `16#FFFF0000` Please note that the alarm state is in effect if the expression in the *"Colorvariables ➔ Toggle color"* property has the value `TRUE`.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Button state variable'

“Digital variable”	At runtime, the property controls whether the Button is displayed as pressed or not. Values: ● `FALSE`: The Button is displayed as not pressed. ● `TRUE`: The Button is displayed as pressed. Argument passed to the property: ● Placeholder for the user input variable to couple the representation of the Button with the input variable. – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the Button. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `prgA.xButtonState` Note: Implement a value assignment in the code for the variable specified here.

Element property 'Image ID variable'

"Image ID"	Variable (`STRING`). Contains the image ID. The contents of the string corresponds to the description of the *"Static ID"* property. Example: `PLC_PRG.stImageID := 'ImagePool_A.Image3';`

See also

- *Chapter 1.4.5.19.5.5 "Visualization Element 'Image'" on page 1842*
- *Chapter 1.4.1.20.2.13 "Object 'Image Pool'" on page 873*

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Group Box'

Symbol:

Category: *“Common Controls”*

The element provides a visual grouping of visualization elements. The group box can have multiple levels of nesting.

> *You can also use drag&drop to add elements to a “Group Box”. To do this, drag the element to the window area of the “Group Box”. The appearance of the cursor changes (a small plus sign is displayed). When you click the [Shift] key at the same time, the element is not added.*
>
> *You can remove elements from the “Group Box” by dragging them out of the window area.*

Element properties

“Element name”	Example: `Parameter axis 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Group Box”*
“Clipping”	☑: Elements that protrude beyond the size of the group box are clipped.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

> *You can also change the values by dragging the box symbols (⊡) to other positions in the editor.*

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

The properties contains character strings for labeling the element.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Example: `Axis 1`.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Parameters of Axis 1`.

See also

- ↳ *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

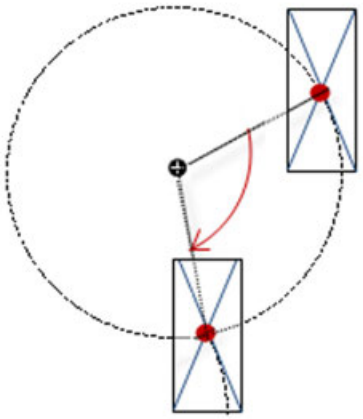
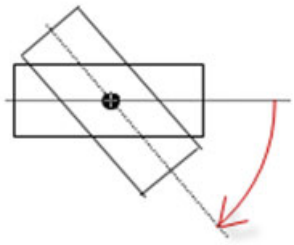
Element property 'Text properties'

The properties contain fixed values for the text properties.

“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed. 

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Table'

Symbol:

Category: *“Common Controls”*

The element displays data that can be represented as an array in a table. Therefore, the data type of the visualizing variable can be 1) a one-dimensional array, 2) a maximum two-dimensional array, 3) an array of an array, 4) an array of structures, or 5) an array of a function block.

Element properties

“Element name”	Example: `Data set component 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	Table
“Data array”	Array whose data is visualized as a table Variable (`ARRAY`) whose data type determines the number of columns and rows in the table Array types ● One-dimensional array: The table has one column. ● Two-dimensional array: The second dimension determines the number of columns. ● Array of an array: The number of array elements of the back array determines the number of columns. ● Array of a structure: The number of structure members determines the number of columns. ● Array of a function block: The number of local variables determines the number of columns. Example: `PLC_PRG.aiTable` Declaration: `aiTable : ARRAY[0..3, 0..4] OF INT := [4(1, 2, 3, 4, 5)];` Hint: If the declaration of the array changes, then the table can be refreshed by placing the cursor in the data array value field and pressing the *[Enter]* key.
“Max. array index”	Top index limit for the displayed table. Limits the number of displayed rows. The index begins at 0. ● Variable (integer data type) Example: `PLC_PRG.iUpperIndexBoundToDisplay` ● Integer literal Example: `4` is displayed as 5 in the row of the table.

See also

- *Data Type 'ARRAY'*

Element property 'Columns'

The *“Table”* element shows the values of a variable in a tabular view. The array elements of structure members are shown in a column or in a row. Two-dimensional arrays or arrays of a structure are shown in multiple columns. The visualized variable is defined in the *“Data array”* property. When a variable is assigned there, you can specify the display of the Table columns where the array elements are shown. An individual configuration is possible for each column that is assigned to an index [<n>].

“Show row header”	☑: The row header is visible. Example: For an array, the index of the array element is displayed in the header.
“Show column header”	☑: The column label is visible.
“Row height”	Height of the rows (in pixels)
“Row header width”	Width of the row label
“Scroll bar size ”	Size of the scroll bar (in pixels)

Table 360: “Element property 'Columns: Column [<n>]'”

“Column header”	By default, the name of the array or structure is applied as the heading with the index or structure member for the column. If an array of a function block has been selected for *“Data array”*, then the name of the array is applied to the column header with the local variables of the function block that belong to the column. The column label can be changed here by specifying a new title.
“Width”	Column width (in pixels)
“Image column”	☑: Images can be displayed in the column. Images are used from the global image pool or custom image pools. The image IDs are shown in the cells of the Table as they are defined in the image pool.
“Image configuration”	
“Fill mode”	• `Fill cell`: The image resizes to the dimensions of the cell without fixing the height/width ratio. • `Centered`: The image is centered in the cell and retains its proportions (height/width ratio).
“Transparency”	☑: The color which is specified in *“Transparent color”* is displayed as transparent.
“Transparent color”	This color is displayed as transparent. Requirement: The *“Transparency”* property is activated.
“Text alignment of header”	Alignment of the column header: • `Left` • `Centered` • `Right`
“Use template”	☑: Another visualization element (type *“Rectangle”*, *“Rounded Rectangle”*, or *“Ellipse”*) is inserted into each line of this Table column. The properties list is extended automatically with the properties of this element in *“Template”*.
“Text alignment of the headline from the template”	Requirement: The *“Use template”* property is activated. ☑: When activated, the settings for font (size) and alignment in the inserted template are also applied to the column header.
“Template”	Requirement: The *“Use template”* property is activated. The properties of all elements assigned to the column are listed in *“Template”*. They can be modified there as described in *“Rectangle”*, *“Rounded Rectangle”*, and *“Ellipse”*.

See also

- *Chapter 1.4.5.19.5.1 “Visualization Element 'Rectangle', 'Rounded Rectangle', 'Ellipse'” on page 1792*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.

"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables enable dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog box.
"Size"	Variable (integer data type). Contains the font size (in pixels). Example: `PLC_PRG.iFontHeight := 16;.` The selection of font sizes corresponds to the default *"Font"* dialog box.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Charset”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog box.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

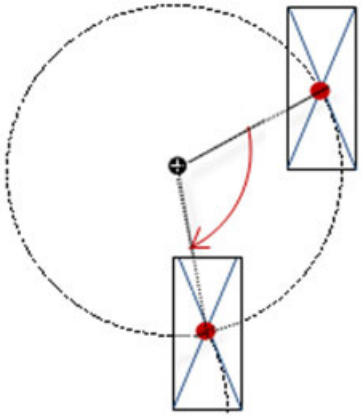
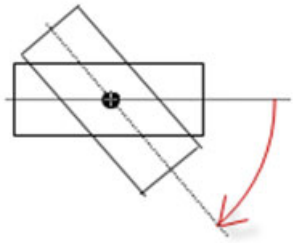
See also

- *“Element property 'Text properties'” on page 1919*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X.` Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y.` Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Selection'

“Background color on selection”	Fill color of the selected row.
“Selection font color”	Font color of the selected row.
“Selection type”	Selection when clicking the table row. ● `No selection`: No selection ● `Cell selection`: The clicked cell only. ● `Row selection`: Row of the clicked cell. ● `Column selection`: Column of the clicked cell. ● `Row and column selection`: Row and column of the clicked cell.
“Frame around selected cells”	☑: A frame is drawn around the selected cells.
“Variable for selected column”	Variable (INT). Contains the array index of the *“Column”* of the selected cell. If the data array points to a structure, then the structure components are indexed, starting at 0. Warning: This index represents the correct position in the array only if no columns have been removed from the table in the display.
“Variable for selected row”	Variable (INT). Contains the array index of the *“Row”* of the selected cell.
“Variable for valid column selection”	Variable (BOOL). TRUE: The *“Variable for selected column”* variable contains a valid value.
“Variable for valid row selection”	Variable (BOOL). TRUE: The *“Variable for selected row”* variable contains a valid value.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.21.5 "Displaying Array Variables in Tables" on page 2140*
- *Data Type 'ARRAY'*

Visualization Element 'Text Field'

Symbol:

Category: *"Common Controls"*

The element is used for the following purposes:

- Static output of text. The contents of a variable can be part of the text.
- Showing a tooltip. The text is managed as static text and can also be defined so that the contents of a variable are also displayed.
- Dynamic output of text. Texts of a text list are displayed dynamically.
- Input of text. For example, a user can input a number or a text literal.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element properties

“Element name”	Optional Example: `FileName_A` Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Text Field”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Colors'

“Normal state”	The normal state is in effect if the variable in *“Color variables ➔ Toggle color”* is not defined or it has the value `FALSE`.
“Frame color”	Frame and fill color for the corresponding state of the variable.

“Fill color”	
“Transparency”	Transparency value (0 to 255) for defining the transparency of the selected color. Example: 255: The color is opaque. 0: The color is completely transparent.
“Alarm state”	The alarm state is in effect if the variable in *“Color variables ➔ Toggle color”* has the value `TRUE`.

See also

- *Chapter 1.4.5.19.3.5 “Dialog 'Gradient Editor'” on page 1748*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

“Line width”	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Fill attributes”	The way in which the element is filled. ● *“Filled”*:The element is filled with the color from property *“Colors ➔ Fill color”*. ● *“Invisible”*: The fill color is invisible.
“Line style”	Type of line representation ● *“Solid”* ● *“Dashes”* ● *“Dots”* ● *“Dash Dot”* ● *“Dash Dot Dot”* ● *“not visible”*

You can assign variables in the “Appearance variables” property for controlling the appearance dynamically. The fixed values here are overwritten.

See also

- *“ Element property 'Appearance variables'” on page 1867*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl]* + *[Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1919*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1918*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

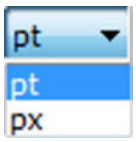
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▼: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1919*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

“Toggle color”	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *“Color”* property. ● `TRUE`: The element is displayed with the color specified in the *“Alarm color”* property. Assignment options: ● Placeholder for the user input variable – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
“Normal state” *“Alarm state”*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *“Color variables”*, *“Toggle color”* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *“Colorvariables”*, *“Toggle color”* has the value `TRUE`.
“Frame color”	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
“Filling color”	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the “Activate semi-transparent drawing” option of the visualization manager is selected.

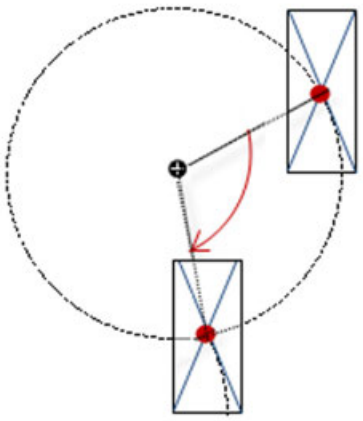
Select the “Advanced” option in the toolbar of the properties view. Then all element properties are visible.

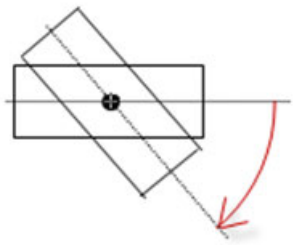
See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Selection and caret configuration'

The variables allow for controlling the caret position and the selection of the text.

"Caret position"	Variable (integer data type). Contains the position of the cursor.
"Selection start"	Variable (integer data type). Contains the position of the first selected character. Example: `PLC_PRG.iSelStart`
"Selection end"	Variable (integer data type). Contains the position of the last selected character. Example: `PLC_PRG.iSelEnd`
"All selected"	Variable (BOOL). Toggles the selection of the entered text. `TRUE`: The text in the text field is selected. `FALSE`: The selection starts with the value in *"Selection start"* and ends with *"Selection end"*.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.

"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.
"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Scroll Bar'

Symbol:

Category: *"Common Controls"*

The element sets the value of a variable, depending on the position of the scroll bar.

Element properties

"Element name"	Example: `Speed Conveyor Belt 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Scroll Bar"*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Value"	Variable as type integer that includes the position of the scroll bar.
"Minimum value"	Smallest value of the scroll bar (fixed value or variable).
"Maximum value"	Largest value of the scroll bar (fixed value or variable).

"Page size"	Page size ● As a fixed value, for example `10` ● As a variable of data type integer Requirement: Visible when the *"Move to click"* property is **not** selected.
"Move to click"	Behavior of the scroll bar at visualization runtime when it is clicked: ☑: The scrollbar moves to the clicked position. ☐: The scrollbar moves to one *"Page size"* in the direction of the click.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊡) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Bar'

The property defines the representation of scaling and direction of travel.

"Orientation"	Alignment of the slider; defined by the ratio of width to height. ● *"Horizontal"* ● *"Vertical"* You can modify the alignment in the visualization editor by using the pointing device to adjust the width and height of the Scroll Bar.
"Running direction"	The drop-down list varies depending on the alignment of the slider. Horizontal ● *"Left to right"*: Scale starts at the left. ● *"Right to left"*: Scale starts at the right. Vertical ● *"Bottom to top"*: Scale starts at the bottom. ● *"Top to bottom"*: Scale starts at the top.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Color"	Color for the element in its normal state. Please note that the normal state is in effect if the expression in the *"Color variables ➔ Toggle color"* property is not defined or it has the value `FALSE`.
"Alarm color"	Color for the element in alarm state. Please note that the alarm state is in effect if the expression in the *"Color variables ➔ Toggle color"* property has the value `TRUE`.
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *"GlobalTextList"* text list. Therefore, these texts can be localized.

"Text"	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Text"*.
"Tooltip"	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *"Text variable ➔ Tooltip"*.

See also

- *"Element property 'Text variables'" on page 1931*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*
- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

"Text variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *"Texts ➔ Text"*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
"Tooltip variable"	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *"Texts ➔ Tooltip"*.

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*
- *"Element property 'Texts'" on page 1930*
- *Chapter 1.4.1.19.5.17 "Enumerations" on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

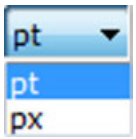
"Text list"	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
"Text index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
"Tooltip index"	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 "Object 'Text List'" on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.

“Flags”	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
“Character set”	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *“Script”* setting of the standard *“Font”* dialog.
“Color”	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
“Flags for text alignment”	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 1919*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assignment options: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
"Normal state" *"Alarm state"*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *"Color variables"*, *"Toggle color"* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *"Colorvariables"*, *"Toggle color"* has the value `TRUE`.
"Frame color"	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
"Filling color"	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

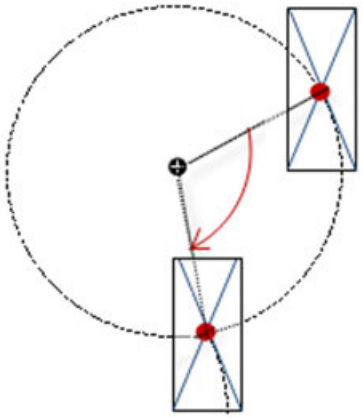
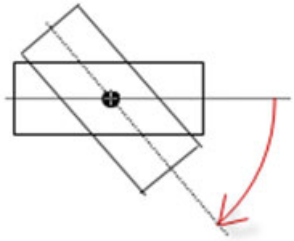
Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Slider'

Symbol:

Category: *"Common Controls"*

The element changes the value of a variable, depending on the position of the slider within the slider bar. You define the value range of the slider bar by means of the scale start and scale end.

Element properties

"Element name"	Example: `Speed controller conveyor belt 1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Slider"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	Variable (numeric data type) Example: `PLC_PRG.rSlider` When executed, the variable assigns a value that corresponds to the position of the slider in the bar.
"Page size"	Page size ● As a fixed value, for example `10` ● As an IEC variable of data type integer Requirement: The *"Move to click"* element property is **not** selected.
"Move to click"	Behavior of the slider at visualization runtime when it is clicked: ☑: The slider moves to the clicked position. ☐: The slider moves to the value (defined in the *"Page size"* element property) in the direction of the click.

Element property 'Scale'

"Show scale"	☑: The element has a visible scale. Note: This option is available for the *"Slider"* only.
"Scale start"	Least value of the scale and the lower limit of the value range for the element. Example: `0` : The property *"Variable"* is shown below.

“Variable”	Variable (integer data type). Contains the scale start.
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element. Example: `100` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale end.
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two tick marks on the rough scale. Example: `10` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the distance.
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Subscale”	Distance between two dashes on the fine scale. You can hide the fine scale by setting the value to `0`. Example: `2` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the distance.
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 5;`
	`END_VAR`

“Scale format (C Syntax)”	Formatting of the scale label (example: `%d %s`) Note: This property is available for the Slider only.
“Scale proportion”	Size of the scale (in %) of the total size

Element property 'Bar'

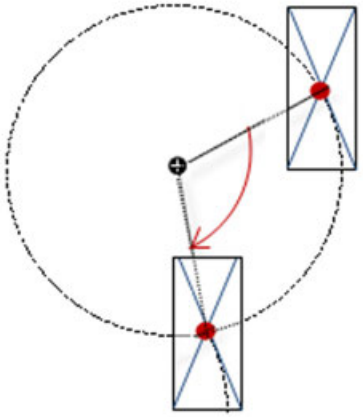
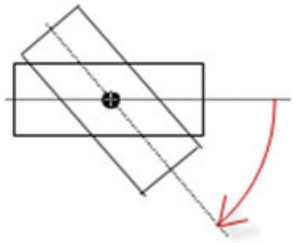
The property defines the representation of scaling and direction of travel.

"Diagram type"	The drop-down list varies depending on the alignment of the diagram. Horizontal ● *"Top"*: Scale is above the slider. ● *"Bottom"*: Scale is below the slider. ● *"Top and bottom"*: Two scales frame the slider above and below. Vertical ● `Left`: Scale is left of the slider. ● `Right`: Scale is right of the slider. ● `Left and right`: Two scales frame the slider on the left and the right.
"Orientation"	Alignment of the slider; defined by the ratio of width to height. ● *"Horizontal"* ● *"Vertical"* You can modify the alignment in the visualization editor by using the pointing device to adjust the width and height of the scrollbar.
"Running direction"	The drop-down list varies depending on the alignment of the slider. Horizontal ● *"Left to right"*: Scale starts at the left. ● *"Right to left"*: Scale starts at the right. Vertical ● *"Bottom to top"*: Scale starts at the bottom. ● *"Top to bottom"*: Scale starts at the top.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Spin Box'

Symbol:

Category: *“Common Controls”*

The element increments or decrements the value of a variable in defined intervals.

Element properties

“Element name”	Example: `Speed controller conveyor belt` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Spin Box”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

"Variable"	Variable (numeric data type) Example: `PLC_PRG.iTemp`
"Number format"	Format of the value in printf syntax Example: %d, %5.2f
"Interval"	Interval used for modification of the value

Element property 'Value range'

"Minimum value"	Lower limit of the output value ● fixed value ● Variable (INT)
"Maximum value"	Upper limit of the output value ● fixed value ● Variable (INT)

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Usage of"	● *"Default style values"*: The values of the visualization style are used. ● *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

"Toggle color"	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *"Color"* property. ● `TRUE`: The element is displayed with the color specified in the *"Alarm color"* property. Assigning the property: ● Placeholder for the user input variable – *"<toggle/tap variable>"* – *"<NOT toggle/tap variable>"* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *"Tap"* or *"Toggle"* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *"Toggle"* and *"Tap"*, then the variable specified in *"Tap"* is used. Hint: Click the symbol to insert the placeholder *"<toggle/tap variable>"*. When you activate the *"Inputconfiguration"*, *"Tap FALSE"* property, then the *"<NOT toggle/tap variable>"* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.

The transparency part of the color value is evaluated only if the "Activate semi-transparent drawing" option of the visualization manager is selected.

Select the "Advanced" option in the toolbar of the properties view. Then all element properties are visible.

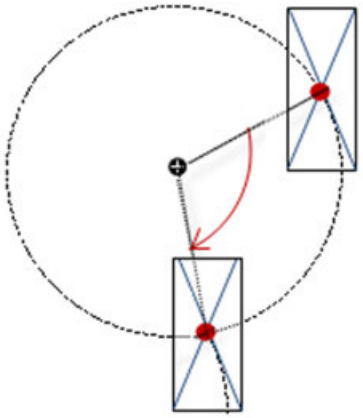
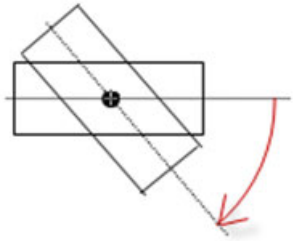
See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The "X", "Y", "Rotation", and "Interior rotation" properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. User input is a user event from the perspective of the element.

The input configuration refers to the text area of the element only, not the two buttons.

The *“Configure”* button opens the *“Input configuration”* dialog box for creating or modifying a user input configuration. A configuration contains one or more input actions for the respective input event. Existing input actions are displayed below it. Example: *“Execute ST code”*: `PLC_PRG.i_x := 0;`	
“OnDialogClosed”	Input event: The user closes the dialog box.
“OnMouseClick”	Input event: The user clicks the element completely. The mouse button is clicked and released.
“OnMouseDown”	Input event: The user clicks down on the element only.
“OnMouseEnter”	Input event: The user drags the mouse pointer to the element.
“OnMouseLeave”	Input event: The user drags the mouse pointer away from the element.
“OnMouseMove”	Input event: The user moves the mouse pointer over the element area.
“OnMouseUp”	Input event: The user releases the mouse button over the element area.

See also

- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

“Tap”	When a mouse click event occurs, the variable defined in *“Variable”* is described in the application. The coding depends on the options *“Tap FALSE”* and *“Tap on enter if captured”*.
“Variable”	Variable (`BOOL`). Contains the information whether a mouse click event exists. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts while the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *“Tap FALSE”* option is not activated.
“Tap FALSE”	☑: The mouse click event leads to a complementary value in *“Variable”*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
“Tap on enter if captured”	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

“Shift”	When a mouse click event occurs, the variable here is described in the application. When the mouse click event ends, its value is toggled with the *“Toggle on up if captured”* option.
“Variable”	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Tip: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
“Toggle on up if captured”	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

“Hotkeys”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Event(s)”* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
“Key”	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
“Event(s)”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed in the “Keyboard configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Invisible Input'

Symbol:

Category: *"Common Controls"*

This element is displayed in the editor with a dashed line which is not visible in online mode. You define the behavior of the el in the input configuration.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Unsichtbare_Eingabe_1`
"Type of element"	*"Invisible Input"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

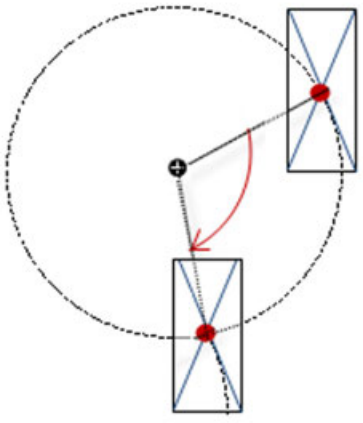
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

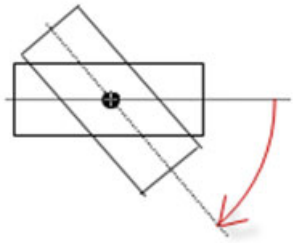
“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

“Hotkey”	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *“Events”* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
“Key”	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
“Events”	● *“None”* ● *“Mouse down”*: Pressing the key triggers the input actions that are configured in the *“OnMouseDown”* property. ● *“Mouse up”*: Releasing the key triggers the input actions that are configured in the *“OnMouseUp”* property. ● *“Mouse down/up”*: Pressing and releasing the key triggers the input actions that are configured in the *“OnMouseDown”* property and the *“OnMouseUp”* property.
“Shift”	☑: Combination with the Shift key Example: *[Shift]+[T]*.
“Control”	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
“Alt”	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the “Keyboard Configuration” tab.

See also

- *Chapter 1.4.5.19.2.2 “Command 'Keyboard Configuration'” on page 1720*
- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Check Box'

Symbol:

Category: *“Common Controls”*

The element is used for setting and resetting a Boolean variable. The set state is identified by a check mark.

Element properties

“Element name”	Example: `signal_tone_for_parts_deficit` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Check Box”*
“Text ID”	ID for the text in the *“GlobalTextList”* Example: `22` The text ID cannot be changed. As soon as you specify and save a text in *“Texts”* - *“Text”*, CODESYS automatically creates an entry in the *“GlobalTextList”* and displays the ID here.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

“Variable”	Variable of type `BOOL` Example: *“PLC_PRG.xIsTrue”*
“Frame size”	Distance of the element to the edge Example: *“From style”*

Element property 'Texts'

The properties contains character strings for labeling the element.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Example: `Axis 1`.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Parameters of Axis 1`.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element property 'Text properties'

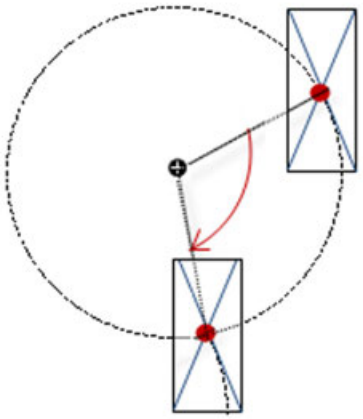
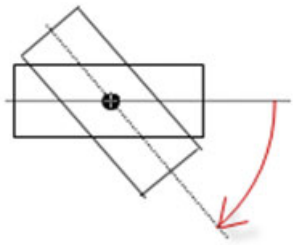
The properties contain fixed values for the text properties.

“Usage of”	● *“Default style values”*: The values of the visualization style are used. ● *“Individual settings”*: The "Individual text properties" property group is shown The values can be customized here.
“Individual text properties” Requirement: The *“Individual settings”* text property is defined.	
“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.

"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Progress Bar'

Symbol:

Category: *"Common Controls"*

The element displays the value of a variable as a progress bar.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Progress_Data_Transfer`
"Type of element"	*"Progress Bar"*
"Text ID"	ID of the global text list Requirement: Text is configured in the property *"Texts ➔ Text"*.
"Variable"	Variable (numeric data type). Represents the length of the progress bar.
"Minimum value"	Value range of the variable
"Maximum value"	
"Style"	● *"Blocks"* ● *"Bar"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

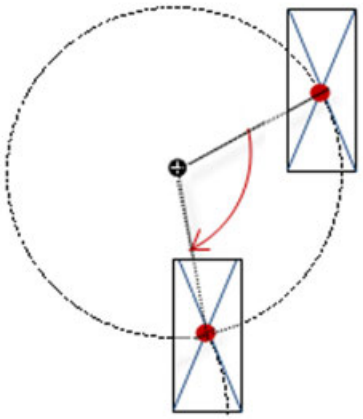
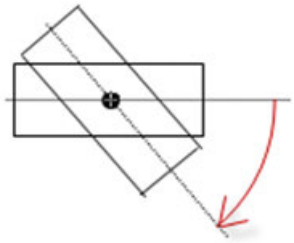
> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

Element property 'Texts'

"Text"	String label for the element. Example: `Zoom`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

● *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Radio Buttons'

Symbol:

Category: *"Common Controls"*

The element provides a series of radios buttons with an unlimited number of options.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Morning Shift`
“Type of element”	*“Radio Buttons”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Variable”	Variable (integer data type) that gives the index of the radio button that the visualization user has activated Example: `PLC_PRG.iNrOfActivatedRadioButton`
“Number of columns”	Definition of the number of list boxes displayed in a row Example: `2`
“Radio button order”	*“Left to right”*: The radio buttons are aligned by rows until the number of columns is reached. *“Top to bottom”*: The radio buttons are aligned row by columns until the number of columns is reached.
“Frame size”	Defines the distance from the list boxes to the edge (in pixels).
“Row height”	Height of the row (in pixels) Modifying the height of the row also changes the size of the list box.

Element property 'Text properties'

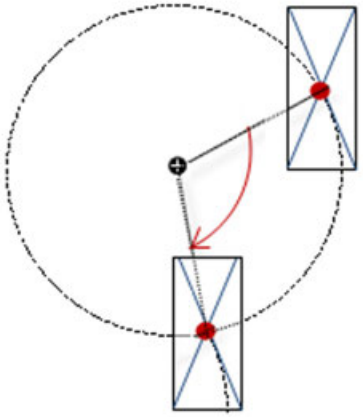
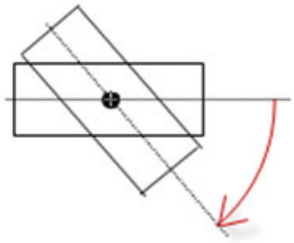
The properties contain fixed values for the text properties.

"Usage of"	● *"Default style values"*: The values of the visualization style are used. ● *"Individual settings"*: The "Individual text properties" property group is shown The values can be customized here.
"Individual text properties" Requirement: The *"Individual settings"* text property is defined.	
"Horizontal alignment"	Horizontal alignment of the text within the element.
"Vertical alignment"	Vertical alignment of the text within the element.
"Text format"	Definition for displaying texts that are too long ● *"Default"*: The long text is truncated. ● *"Line break"*: The text is split into parts. ● *"Ellipsis"*: The visible text ends with "..." indicating that it is not complete.
"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Radio button settings'

“Radio button” ● *“Areas”* – *“[<n>]”*	*“Create new”*: Clicking this button creates a new selection button in the editor and lists an additional area in the properties editor. For each radio button, an area is visible that records the settings. ● [<n>] – *“[<n>]”*: This number indicates the area. Clicking *“Delete”* will delete the associated radio button with its settings *“Text”*, *“Tooltip”*, and *“Line spacing (in pixels)”*.
Areas: [<n>]	
“Text”	The button name is specified here. Default value: *“Radio_button”*
“Tooltip”	Text is specified here that is displayed in a tooltip.
“Line spacing (in pixels)”	The distance (in pixels) to the upper button can be specified here.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Alarm Table'

Symbol:

Category: *“Alarm Manager”*

The element displays alarms in a list. In the element properties, you specify which information is shown. You define the appearance of the element and the variables that control the element behavior.

In online mode, you can sort an alarm table by a specific column – even in the classic view. Click into the column header. A small triangle indicates the current sort order (ascending, descending). Clicking the symbol reverses the order.

Sorting inside the column depends on the type of the contained information. The "Priority" column is sorted numerically, and the "Message" and "Class" columns alphabetically. The "Value" and "Latch" columns may contain different value types. In this case, sorting is first by type (blank, Boolean, numeric value, character string) and then either numerically or alphabetically depending on the type.

If an alarm history has been created, then you can programmatically delete it at runtime. The recording starts again from the time of deletion. See the help page for "Visualizing Alarm Management".

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Alarm Table"*
"Data source"	Selection of the device and the application where the data to be visualized and the alarms are generated ● Remote data source which accesses a remote device, accesses a remote application, and then transfers the data to the alarm configuration Example: `DataSource_A` Below the (now visible) *"Application"* property, the remote application is displayed as configured in the data source. Example: `App_A` Note: If the data source is accessed symbolically by means of a symbol file (CODESYS symbolic), then the required symbol file and the corresponding project have to be saved in the same folder. ● Local application below which the alarm configuration is located Example: *"<local application>"*

See also

- *Object 'Data Source'*

Element property 'Alarm configuration'

"Alarm groups"	Opens the *"Select Alarm Group"* dialog where you define the alarm groups that you want to display.
"Priority from"	Least priority for alarm display. (0 to 255).
"Priority to"	Greatest priority for alarm display. (0 to 255).
"Alarm classes"	Opens the *"Select Class Group"* dialog where you define the alarm classes that you want to display.
"Filter criterion"	For the *"Alarm Banner"* element only ● *"Most important"*: The alarm with the highest priority (lowest value) is displayed. ● *"Newest"*: The most recent alarm is displayed.

"Filter by latch 1"	The generated alarms (previous and current) can be filtered by the contents of *"Latch variable 1"*, which is specified in the configuration of the alarm group. In *"Filter type"*, you define whether or not the filtering is performed by a string value or a numerical value. ● *"Filter variable"*: Indicates what the alarms are filtered by. Possible entries: Application variable of data type `STRING` or `WSTRING`, or a literal value directly. Examples:`PLC_PRG.strFilterVariable`, `'STRING'`. ● *"Filter type"*: Integer value that determines by which criteria the latch variable value is used for filtering. Possible entries: Numerical variable from the application (example: `PLC_PRG.diFilterType`, or a value directly (example: `2`). Possible values: – `0`: No filtering – `1`: Filter by alarms whose latch variable 1 contains the string specified in *"Filter variable"*. Example: The filter variable contains `'Error 1'` which is the latch variable 1 of different alarms of type `STRING` and has the value `'Error 1'` ->. Only these alarms are displayed. – `2`: Filter by alarms whose latch variable 1 contains the typed literal specified in *"Filter variable"* according to IEC 61131-3. Examples: `T#1h2s`, `DINT#15`, `REAL#1.5`, `FALSE` – `3`: Filter by alarms whose latch variable 1 contains the LINT literal value specified in *"Filter variable"*. Therefore, the value of the latch variables has to be in the range of 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. – All other values: The behavior is not defined and can change in the future.
"Filter by time range"	The generated alarms (remote, historical, local) can be displayed for a specified time range. You use the *"Filter type"* to define whether filtering by time range is enabled or disabled. ● *"Filter variable, from"*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeFrom`) for the start time that the alarms are displayed. ● *"Filter variable, to"*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeTo`) for the end time that the alarms are displayed. ● *"Filter type"*: Variable of integer data type that determines whether *"Filter by time range"* is enabled or disabled. Possible values: – `1`: Filtering is enabled – `0`: Filtering is disabled

See also

- *Chapter 1.4.5.19.3.17 "Dialog 'Selected Alarm Group'" on page 1769*
- *Chapter 1.4.5.19.3.16 "Dialog 'Selected Alarm Class'" on page 1768*

Element property 'General table configuration'

"Show row header"	☑: Display of the row number at the beginning of the row.
"Show column header"	☑: Display of the column heading as defined in *"Column heading"*.
"Row height"	Height of the table rows (in pixels).
"Row header width"	Width of the line header (in pixels).

"Scrollbar size"	Width of the scrollbar when it runs vertically. Width of the scrollbar when it runs horizontally. Specified in pixels
"Automatic line break for alarm message"	☑: The message text is truncated at the end of the line. ☐: The message text is truncated at the end of the column, if the text is too long.

Element property 'Columns: Column [<n>]'

By default, columns [0] and [1] are configured: *"Time stamp"* and *"Message text"*. You can create more columns by clicking the *"Create new"*, and remove columns by clicking *"Delete"*.

Animations (dynamic text, font variables), text, and tooltip are not supported.

"Column header"	The standard header is set and changed here by specifying a new text.
"Use text alignment in title"	☑: The text in the column header is aligned according to the current definition in *"Text alignment"*. ☐: The text in the column header is centered.
"Width"	Width of the column (in pixels).
"Data type"	Notice about time stamps: For use in a TargetVisu or WebVisu, you can control the date and time format with the help of the global string variables from the library `Alarmmanager.library`: `AlarmGlobals.g_sDateFormat` (example: `AlarmGlobals.g_sDateFormat := 'MM.yyyy'`) and `AlarmGlobals.g_sTimeFormat` (example: `AlarmGlobals.g_sTimeFormat := 'HH:mm'`). Define the information to be displayed in the column. ● *"Symbol"* ● *"Time stamp"*: Date and time of the last status change of the alarm. ● *"Time stamp active"*: Date and time of the last activation of the alarm. ● *"Time stamp inactive"*: Date and time of the last deactivation of the alarm. ● *"Time stamp acknowledge"*: Date and time of the last acknowledgment. ● *"Value"*: Current value of the printout ● *"Message text"*: Output of the message text ● *"Priority"*: Alarm priority ● *"Class"*: Alarm class ● *"State"*: Alarm state ● *"Latch Variable <n>"*: Value of the selected latch variables
"Text alignment"	Alignment of the text in this column ● *"Left"* ● *"Centered"* ● *"Right"*
"Color settings"	● *"Activate color settings"*: Boolean variable for activating and deactivating the color settings defined here. Example: `PLC_PRG.bColorSettings` ● *"Cell fill color"*: – *"Color variable"*: Variable for the cell fill color, example: `dwCellColor` (hexadecimal color definition: 16#TTRRGGBB) – *"Use color also for column header"*: ☑: The color defined via *"Color variable"* is used in the column header as well. ● *"Text color"*: – *"Color variable"*: Variable for the definition of the text color in the column, example: `dwTextColor` (hexadecimal color definition: 16#TTRRGGBB) – *"Use color also for column header"*: ☐: The color defined via *"Color variable"* is used in the column header as well.

See also

- *Chapter 1.4.5.8.3 "Animating a color display" on page 1295*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Selection'

"Selection color"	Fill color of the selected row
"Selection font color"	Font color of the selected row
"Frame around selected cells"	☑: A frame is drawn around the selected cells at runtime.
"Variable for selected alarm group"	Name of the affected alarm group; type: `STRING`, `WSTRING`
"Variable for selected alarm ID"	Alarm ID of the affected alarm group; type: `STRING`, `WSTRING`
"Variable for selected line"	Index of the selected alarm line (0-based). The index can be read and written; integer data type
"Variable for valid selection"	TRUE: An alarm line is selected. FALSE: The selection is invalid. For example, for an empty alarm table or when an alarm is not selected yet.
"Variable for selected alarm information"	Information about the selected alarm. Type `AlarmSelectionInfo` For easy usage, the function block `AlarmSelectionInfoDefault` is provided. This FB fills the structure with the memory for 10 messages and 10 latch variables. Example: `myAlarmSelectionInfoDefault.AlarmSelectionInfo` The following information is available: • `sAlarmgroup` • `uialarmID` • `timeStampActive` • `timeStampInactive` • `timeStampAcknowledge` • `timeStampLast` • `paLatchVariables` • `iLatchVariablesCount` • `papwsAlarmMessages` • `dwAlarmMessageTextBufferSize` • `iAlarmMessagesCount` • `iSelectionChangedCounter`

Element property 'Control variables'

Boolean variables are defined here for executing specific actions in the table can be executed at runtime.

“Acknowledge selected”	Variable (`BOOL`) Example: `PLC_PRG.bAckSelectedAlarms` If the assigned variable is `TRUE`, then the selected alarm is acknowledged.
“Acknowledge all visible”	Variable (`BOOL`) Example: `PLC_PRG.bAckVisibleAlarms` If the assigned variable is `TRUE`, then all alarms are acknowledged that are visible in the alarm table.
“History”	Variable (`BOOL`) Example: `PLC_PRG.bShowHistory` If the assigned variable is `TRUE`, then the history alarms are displayed in addition to the active alarms. In the classic view, the same sort options apply as in normal mode. Note: Acknowledgment is not possible in this view.
“Freeze scroll position”	Variable (`BOOL`) Example: `PLC_PRG.bFreezeScrollPosition` If the assigned variable is `TRUE`, then the scroll position set in the *“History”* view is retained, even if a new alarm is active. If not, then the scroll position jumps to the first table row (the newest alarm).
“Count alarms”	Variable (integer data type) Example: `PLC_PRG.iNumberOfAlarms.` Number of alarms that are currently displayed in the alarm table. Defined by the alarm table.
“Count visible rows”	Variable (integer data type) Example: `PLC_PRG.iNumberVisibleLines` Number of alarms that can be displayed on one page of the alarm table. Defined by the alarm table.
“Current scroll index”	Variable (integer data type) Example: `PLC_PRG.iScrollIndex` The index of the first visible row if the alarm table (0-based). The variable can be read and written.
“Current column sorting”	Variable (integer data type) Example: `PLC_PRG.iColSort` The variable contains a value of the enumeration "VisuElemsAlarm.VisuEnumAlarmDataType". This value determines the column that sorts the alarm table.
“Variable for sorting direction”	Variable (BOOL) Example: `PLC_PRG.xSortAscending` The variable determines the sort order for the entries in the alarm table (`TRUE`: ascending; `FALSE`: descending).

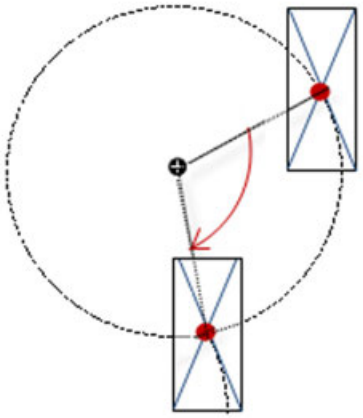
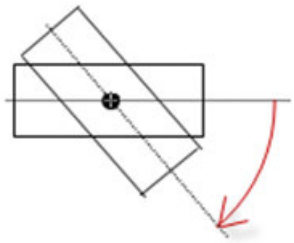
You can also use the “Insert Elements for Acknowledging Alarms” command to define buttons with predefined control variables.

See also

- *Chapter 1.4.5.19.2.23 “Command 'Add Elements for Alarm Acknowledgement'” on page 1744*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Alarm Banner'

Symbol:

Category: *"Alarm Manager"*

The element is a simplified version of the alarm table. It visualizes a single alarm only. In the element properties, you specify which information is shown. You define the appearance of the element and the variables that control the element behavior.

The alarm banner displays active alarms only. If the alarm is acknowledged, then it disappears from the alarm banner.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Alarm Banner"*
"Data source"	If you intend to use a remote alarm configuration, then you have to specify the name of the remote application here. If you do not specify anything, the alarm configuration will be located locally.

Element property 'Alarm configuration'

"Alarm groups"	Opens the *"Select Alarm Group"* dialog where you define the alarm groups that you want to display.
"Priority from"	Least priority for alarm display. (0 to 255).
"Priority to"	Greatest priority for alarm display. (0 to 255).
"Alarm classes"	Opens the *"Select Class Group"* dialog where you define the alarm classes that you want to display.
"Filter criterion"	For the *"Alarm Banner"* element only ● *"Most important"*: The alarm with the highest priority (lowest value) is displayed. ● *"Newest"*: The most recent alarm is displayed.

“Filter by latch 1”	The generated alarms (previous and current) can be filtered by the contents of *“Latch variable 1”*, which is specified in the configuration of the alarm group. In *“Filter type”*, you define whether or not the filtering is performed by a string value or a numerical value. ● *“Filter variable”*: Indicates what the alarms are filtered by. Possible entries: Application variable of data type `STRING` or `WSTRING`, or a literal value directly. Examples:`PLC_PRG.strFilterVariable`, `'STRING'`. ● *“Filter type”*: Integer value that determines by which criteria the latch variable value is used for filtering. Possible entries: Numerical variable from the application (example: `PLC_PRG.diFilterType`, or a value directly (example: `2`). Possible values: – `0`: No filtering – `1`: Filter by alarms whose latch variable 1 contains the string specified in *“Filter variable”*. Example: The filter variable contains `'Error 1'` which is the latch variable 1 of different alarms of type `STRING` and has the value `'Error 1'` ->. Only these alarms are displayed. – `2`: Filter by alarms whose latch variable 1 contains the typed literal specified in *“Filter variable”* according to IEC 61131-3. Examples: `T#1h2s`, `DINT#15`, `REAL#1.5`, `FALSE` – `3`: Filter by alarms whose latch variable 1 contains the LINT literal value specified in *“Filter variable”*. Therefore, the value of the latch variables has to be in the range of 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. – All other values: The behavior is not defined and can change in the future.
“Filter by time range”	The generated alarms (remote, historical, local) can be displayed for a specified time range. You use the *“Filter type”* to define whether filtering by time range is enabled or disabled. ● *“Filter variable, from”*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeFrom`) for the start time that the alarms are displayed. ● *“Filter variable, to”*: Variable of data type `DT` or `DATE_AND_TIME` (example: `PLC_PRG.filterTimeTo`) for the end time that the alarms are displayed. ● *“Filter type”*: Variable of integer data type that determines whether *“Filter by time range”* is enabled or disabled. Possible values: – `1`: Filtering is enabled – `0`: Filtering is disabled

See also

- *Chapter 1.4.5.19.3.17 “Dialog 'Selected Alarm Group'” on page 1769*
- *Chapter 1.4.5.19.3.16 “Dialog 'Selected Alarm Class'” on page 1768*

Element property 'Columns: Column [<n>]'

By default, columns [0] and [1] are preconfigured: *“Time stamp”* and *“Message text”*. You create more columns by clicking *“Create new”*. You remove columns by clicking *“Delete”*.

Animations (dynamic text, font variables), texts, and tooltips are not supported.

"Width"	Width of the column (in pixels)
"Type of data"	About time stamps: When used in a TargetVisu or WebVisu, you can control the date and time format by means of the global string variables from the library `Alarmmanager.library`: `AlarmGlobals.g_sDateFormat` (example: `AlarmGlobals.g_sDateFormat := 'MM.yyyy'`) and `AlarmGlobals.g_sTimeFormat` (example: `AlarmGlobals.g_sTimeFormat := 'HH:mm'`). Here you define the information to be displayed in the column. ● *"Bitmap"* ● *"Time stamp"*: Date and time of the last status change of the alarm ● *"Time stamp active"*: Date and time of the last activation of the alarm ● *"Time stamp inactive"*: Date and time of the last deactivation of the alarm ● *"Time stamp acknowledge"*: Date and time of the last acknowledgement ● *"Value"*: Actual value of the expression ● *"Message"*: Output of the message text ● *"Priority"*: Alarm priority ● *"Class"*: Alarm class ● *"State"*: Alarm state ● *"Latch Variable <n>"*: Value of the selected latch variables
"Text alignment"	Alignment of the contents in the column ● *"Left"* ● *"Centered"* ● *"Right"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (■) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Text properties'

The properties contain fixed values for the text properties.

"Font"	Example: *"Default"* ...: The *"Font"* dialog box opens. ▼: Drop-down list with style fonts.
"Font color"	Example: *"Black"* ...: The *"Color"* dialog box opens. ▼: Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

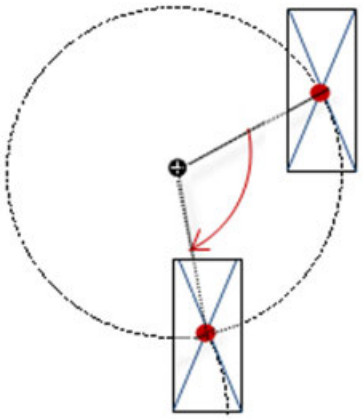
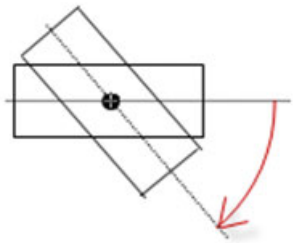
"Acknowledge variable"	A rising edge of this variable acknowledges the alarm.

Handling of multiple active alarms

"Automatic switch"	☑: The display in the alarm banner is switched automatically according to the time to the next alarm as configured in *"Every N second"*.
"Every N second"	Time period until the next switching. Available only if *"Automatic switch"* is selected.
"Next alarm"	Variable for switching to the next alarm. Available only if *"Automatic switch"* is not selected.
"Previous alarm"	Variable for switching to the previous alarm. Available only if *"Automatic switch"* is not selected.
"Multiple alarms active"	Variable that has the value `TRUE` if multiple alarms are active.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Bar Display'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable.

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Element properties

“Element name”	Example: `GenElemInst_2`
“Type of element”	*“Bar Display”*
“Value”	Variable (numeric data type) The value of the variable is displayed as a bar length.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊡) to other positions in the editor.

See also

- ↳ *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Background color"	Drop-down list with background colors Note: This property depends on the style. For example, there are no heterochromatic background images for *"FlatStyle"* and *"Whitestyle"*.
"Own image"	● *"image"*: Image ID of the background image. You select the background image from an image pool by clicking the ... symbol. Info: If you specify the *"<default>"* value or select the image from the *"Default"* category in the input assistant, then the original element background image is used. ● *"Transparent color"*: Color of pixels that are displayed as transparent. Selection from drop-down list or input assistant.
"Optimized drawing"	☑: The background image is drawn one time. If there is a change in the foreground, then only the affected part of the image is redrawn. ☐: The background image is redrawn in cycles. Note: Deactivating this option is sensible only in certain exceptional cases.

Element property 'Bar'

"Diagram type"	Position of the scale ● *"Scale besides bar"* ● *"Scale in bar"* ● *"Bar in scale"* ● *"No scale"*
"Orientation"	Orientation depending on the ratio of width to height of the Bar Display: ● *"Horizontal"* ● *"Vertical"*
"Running direction"	Direction the values are increased. Drop-down list for *"Orientation Horizontal"*: ● *"Left to right"* ● *"Right to left"* Drop-down list for *"Orientation Vertical"*: ● *"Bottom to top"* ● *"Top to bottom"*
"Optimum size for bar"	☑: The bar width requires the majority of the element surface. Note: This property depends on the style. It is not provided for *"FlatStyle"* or *"WhiteStyle"*.

Element property 'Scale'

“Scale start”	Least value of the scale and the lower limit of the value range for the element. Example: `0` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale start.
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element. Example: `100` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the scale end.
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between 2 values on the rough scale. Example: `10` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the distance.
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Subscale”	Distance between 2 values on the fine scale. You can hide the fine scale by setting the value to `0`. Example: `2` : The property *“Variable”* is shown below.

“Variable”	Variable (integer data type). Contains the spacing.
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

“Scale line width”	Specified in pixels. Example: `3`
“Scale color”	Color of scale lines • ...: The *“Color”* dialog box opens. • ▼: A drop-down list with color names opens.
“Scale in 3D”	☑: Tick marks are displayed with slight 3D shadowing. Note: This property depends on the style. Not available for *“FlatStyle”*.
“Element frame”	☑: A frame is drawn around the element.

Element property 'Label'

“Unit”	Text that is displayed in the element. Example: Units displayed in m/s.
“Font”	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the “” button. ...
“ Scale format (C Syntax)”	Values scaled in "printf" syntax Examples: `%d, %5.2f`
“Max. text width of labels”	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Text height of labels”	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Font color”	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

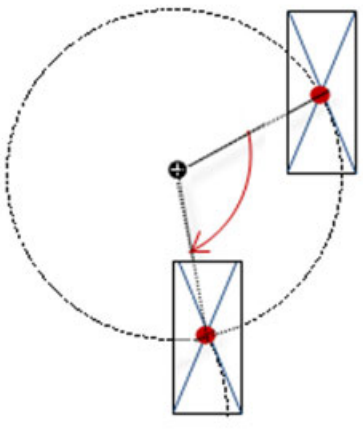
“Horizontal offset”	Distance from the scale (bar) to the horizontal element frame Specified in pixels. Used for achieving the exact position relative to the background image.
“Vertical offset”	Distance from the scale (bar) to the vertical element frame Specified in pixels. Used for achieving the exact position relative to the background image.
“ Horizontal scaling”	Horizontal division of the scale Specified in pixels. Used for achieving the exact positioning relative to the background image.
“Vertical scaling”	Vertical division of the scale Specified in pixels. Used for achieving the exact positioning relative to the background image.

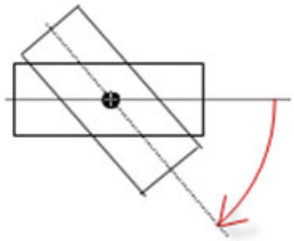
Element property 'Colors'

"Graph color"	Color of the bar
"Bar background"	☑: The background of the bar is black. ☐: The background of the bar is white.
"Frame color"	Color that the frames are drawn.
"Switch whole color"	☑: The total color of the bar is switched to the color of the color area of the current value.
"Use gradient color for bar"	☑: Bar is displayed with a gradient.
"Color range markers"	The color areas can be separated from each other inside the bar with a vertical mark. ● *"No markers"*: No display. ● *"Marker forwards"*: The color of the vertical mark corresponds to the color of the previous color area. ● *"Marker backwards"*: The color of the vertical mark corresponds to the color of the next color area.
"Color areas"	
"Create new"	A new color area is added.
" Delete"	The color area is removed from the list.
"Begin of area"	Start value of the color area
"End of area"	End value of the color area
"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Meter 90°'

Symbol:

Category: *"Measurement Controls"*

The element displays the value of a variable. The needle is positioned according to the value of the assigned variable. A meter is used to represent a tachometer, for example.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Meter 90°"*
"Value"	Variable (numeric data type) The variable value determines the pointer direction of the element.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • : Drop-down list with color names
"Angle range"	Drop-down list for the alignment of the element
"Additional arrow"	: An additional arrow is shown inside the scale.

Element property 'Scale'

“Sub scale position”	● *“Outside”*: The subscale is displayed on the outer scale ring. (*“Frame outside”*) ● *“Inside”*: The subscale is displayed on the inner scale ring. (*“Frame inside”*)
“Scale type”	Type of scale ● *“Lines”* ● *“Dots”* ● *“Squares”*
“Scale start”	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *“Variable”* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

"Variable"	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

"Scale line width"	Specified in pixels Example: `3`
"Scale color"	Color of scale lines ● ...: The *"Color"* dialog opens. ● ▼: A list box with style colors opens.
"Scale in 3D"	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *"FlatStyle"*.
"Show scale"	☑: The scale is displayed.
"Frame inside"	☑: A frame is drawn at the inner end of the scale.
"Frame outside"	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

"Label"	Selection list ● *"Outside"*: Scale values are placed outside of the scale. ● *"Inside"*: Scale values are placed inside of the scale.
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *""* button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

"Usage of"	• *"Preset style values"*: Values from the current style • *"User-defined settings"*: The subnode *"Positioning"* appears.
"Positioning" Requirement: *"User-defined settings"* is selected as *"Usage of"*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
"Needle movement"	Length of the needle (in pixels)
"Scale movement"	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *"Background"*.
"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` ◆: The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

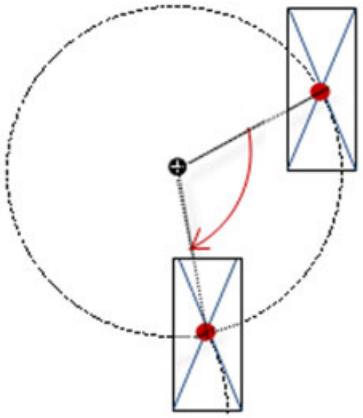
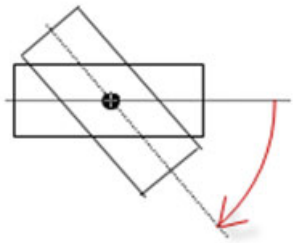
"End of area"	End value of the color area Example: `120` ◆: The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254“”*

Visualization Element 'Meter 180°'

Symbol:

Category: *“Measurement Controls”*

The element displays the value of a variable. The needle is positioned according to the value of the assigned variable on a scale. A meter is used to represent a tachometer, for example.

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Meter 180°”*
“Value”	Variable (numeric data type) The variable value determines the pointer direction of the element.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.

"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • ▼: Drop-down list with color names
"Angle range"	Drop-down list for the alignment of the element
"Additional arrow"	☑: An additional arrow is shown inside the scale.

Element property 'Scale'

"Sub scale position"	• *"Outside"*: The subscale is displayed on the outer scale ring. (*"Frame outside"*) • *"Inside"*: The subscale is displayed on the inner scale ring. (*"Frame inside"*)
"Scale type"	Type of scale • *"Lines"* • *"Dots"* • *"Squares"*
"Scale start"	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *"Variable"* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

“Scale line width”	Specified in pixels Example: `3`

"Scale color"	Color of scale lines ● ...: The *"Color"* dialog opens. ● ▼: A list box with style colors opens.
"Scale in 3D"	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *"FlatStyle"*.
"Show scale"	☑: The scale is displayed.
"Frame inside"	☑: A frame is drawn at the inner end of the scale.
"Frame outside"	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

"Label"	Selection list ● *"Outside"*: Scale values are placed outside of the scale. ● *"Inside"*: Scale values are placed inside of the scale.
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the "" button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

"Usage of"	● *"Preset style values"*: Values from the current style ● *"User-defined settings"*: The subnode *"Positioning"* appears.
"Positioning" Requirement: *"User-defined settings"* is selected as *"Usage of"*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
"Needle movement"	Length of the needle (in pixels)
"Scale movement"	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *"Background"*.

"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

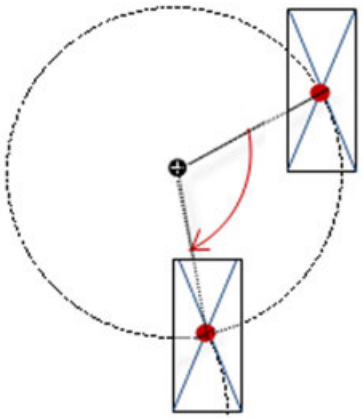
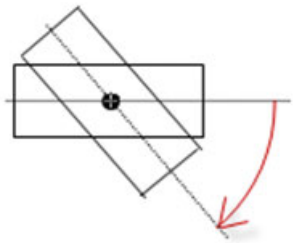
"End of area"	End value of the color area Example: `120` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The "Invisible" property is supported by the "Client Animation" functionality.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Meter'

Symbol:

Category: *"Measurement Controls"*

The element displays the value of a variable. The needle is positioned according to the value of the assigned variable. A meter is used to represent a tachometer, for example.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Meter"*
"Value"	Variable (numeric data type). The variable value determines the pointer direction of the element.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.

"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • ▼: Drop-down list with color names
"Arrow start"	Angle (in degrees) between the scale start and the horizontal axis
"Arrow end"	Angle (in degrees) between the right edge of the pointer instrument and the horizontal axis
"Additional arrow"	☑: An additional arrow is shown inside the scale.

Element property 'Scale'

"Sub scale position"	• *"Outside"*: The subscale is displayed on the outer scale ring. (*"Frame outside"*) • *"Inside"*: The subscale is displayed on the inner scale ring. (*"Frame inside"*)
"Scale type"	Type of scale • *"Lines"* • *"Dots"* • *"Squares"*
"Scale start"	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *"Variable"* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

“Scale line width”	Specified in pixels Example: `3`

“Scale color”	Color of scale lines ● ...: The *“Color”* dialog opens. ● ▼: A list box with style colors opens.
“Scale in 3D”	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *“FlatStyle”*.
“Show scale”	☑: The scale is displayed.
“Frame inside”	☑: A frame is drawn at the inner end of the scale.
“Frame outside”	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

“Label”	Selection list ● *“Outside”*: Scale values are placed outside of the scale. ● *“Inside”*: Scale values are placed inside of the scale.
“Unit”	Text that is displayed in the element. Example: Units displayed in m/s.
“Font”	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *“”* button. ...
“ Scale format (C Syntax)”	Values scaled in "printf" syntax Examples: `%d, %5.2f`
“Max. text width of labels”	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Text height of labels”	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
“Font color”	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

“Usage of”	● *“Preset style values”*: Values from the current style ● *“User-defined settings”*: The subnode *“Positioning”* appears.
“Positioning” Requirement: *“User-defined settings”* is selected as *“Usage of”*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
“Needle movement”	Length of the needle (in pixels)
“Scale movement”	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *“Background”*.

"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

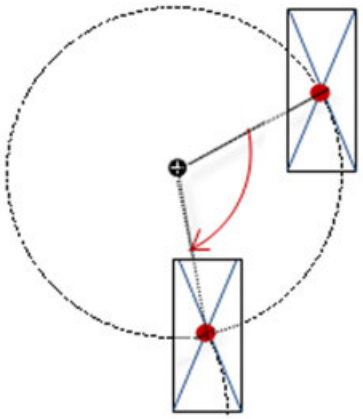
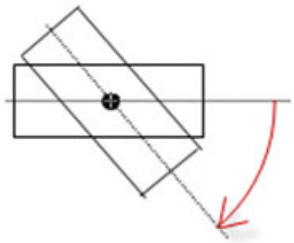
"End of area"	End value of the color area Example: `120` : The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

> *The "Invisible" property is supported by the "Client Animation" functionality.*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Potentiometer'

Symbol:

Category: *"Measurement Controls"*

The element displays the value of a variable as a setting on the potentiometer. A visualization user can modify the value by dragging the pointer to another position.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Potentiometer"*
"Variable"	Variable (numeric data type). Contains the position of the pointer for the potentiometer. A visualization user can modify the value by dragging the pointer to another position.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Background'

"Image color"	List box containing background colors
"Own image"	• *"Image"*: ID of the background image. You select the background image from an image pool by clicking ... Info: If you specify the value *"<default>"* or select the image from the *"Default"* category in the Input Assistant, then the original element background image is used. • *"Transparency color"*: Selection from list box or Input Assistant.

Element property 'Arrow'

"Hand style"	Drop-down list with different arrow types
"Color"	• ...: The *"Color"* dialog box opens. • ▼: Drop-down list with color names
"Arrow start"	Angle (in degrees) between the left edge of the element and the horizontal axis
"Arrow end"	Angle (in degrees) between the right edge of the element and the horizontal axis

Element property 'Scale'

“Sub scale position”	● *“Outside”*: The subscale is displayed on the outer scale ring. (*“Frame outside”*) ● *“Inside”*: The subscale is displayed on the inner scale ring. (*“Frame inside”*)
“Scale type”	Type of scale ● *“Lines”* ● *“Dots”* ● *“Squares”*
“Scale start”	Least value of the scale and the lower limit of the value range for the element Example: `0` : The *“Variable”* property is displayed in the line below this.

“Variable”	Variable (integer data type). Contains the scale start
	Example: `PLC_PRG.iScaleStart`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleStart : INT := 0;`
	`END_VAR`

“Scale end”	Greatest value of the scale and the upper limit of the value range for the element Example: `100` : The *“Variable”* property is shown below this.

“Variable”	Variable (integer data type). Contains the scale end
	Example: `PLC_PRG.iScaleEnd`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iScaleEnd : INT := 120;`
	`END_VAR`

“Main scale”	Distance between two values on the main scale Example: `10` : The *“Variable”* property is shown below.

“Variable”	Variable (integer data type) Contains the distance between two values on the main scale
	Example: `PLC_PRG.iMainScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iMainScale : INT := 20;`
	`END_VAR`

“Sub scale”	Distance between two values on the fine scale You can hide the fine scale by setting the value to `0`. Example: `2` : The *“Variable”* property is shown below this.

"Variable"	Variable (integer data type) Contains the distance between two values on the fine scale
	Example: `PLC_PRG.iSubScale`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iSubScale : INT := 5;`
	`END_VAR`

"Scale line width"	Specified in pixels Example: `3`
"Scale color"	Color of scale lines ● ...: The *"Color"* dialog opens. ● ▼: A list box with style colors opens.
"Scale in 3D"	☑: Scale lines are displayed with soft 3D shadowing. Note: This property is not displayed in *"FlatStyle"*.
"Show scale"	☑: The scale is displayed.
"Frame inside"	☑: A frame is drawn at the inner end of the scale.
"Frame outside"	☑: A frame is drawn at the outer end of the scale.

Element property 'Label'

"Label"	Selection list ● *"Outside"*: Scale values are placed outside of the scale. ● *"Inside"*: Scale values are placed inside of the scale.
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *""* button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	(optional) Value that redefines the maximum width of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	(optional) Value that redefines the maximum height of the scale label. The correct value is normally set automatically. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

Element property 'Positioning'

"Usage of"	• *"Preset style values"*: Values from the current style • *"User-defined settings"*: The subnode *"Positioning"* appears.
"Positioning" Requirement: *"User-defined settings"* is selected as *"Usage of"*. The displayed positioning settings depend on the type of needle instrument and Potentiometer, and partially on whether a custom background image is selected. The following settings are used for achieving the exact position relative to the background image.	
"Needle movement"	Length of the needle (in pixels)
"Scale movement"	Distance from the tick marks to the center (in pixels) Requirement: A customer image is selected as *"Background"*.
"Scale length"	Length of the tick marks (in pixels) Requirement: A customer image is selected as *"Background"*.
"Label offset":	Distance from the labels to the tick marks (in pixels)
"Unit offset":	Distance of the unit text *"Label ➔ Unit"* from the upper scale edge (in pixels)
"Origin offset"	Offset of the element (in pixels) Requirement: For the elements *"Meter 180°"* and *"Meter 90°"*, this property is displayed only if a custom image is selected as *"Background"*.

Element property 'Colors'

"Color areas"	
"Durable color areas"	☐: All color areas are visible, regardless of the current value. ☑: Only the color area is visible that includes the current value.
"Use colors for scale"	☑: Colors in the color area are used only for the scale and frame.
"Color areas"	
"Create new"	A new color area is added to the *"Elements"* view.
" Delete"	The color area is removed from the list and the list is refreshed.
"Begin of area"	Start value of the color area Example: `20` ◆: The property *"Variable"* is shown below.

"Variable"	Variable (integer data type). Contains the start value.
	Example: `PLC_PRG.iColorAreaStart0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaStart0 : INT := 80;`
	`END_VAR`

"End of area"	End value of the color area Example: `120` ◆: The property *"Variable"* is shown below.

“Variable”	Variable (integer data type). Contains the end value.
	Example: `iColorAreaEnd0`
	Declaration:
	`PROGRAM PLC_PRG`
	`VAR`
	`  iColorAreaEnd0 : INT := 100;`
	`END_VAR`

“Color”	Color that is used for displaying the area.

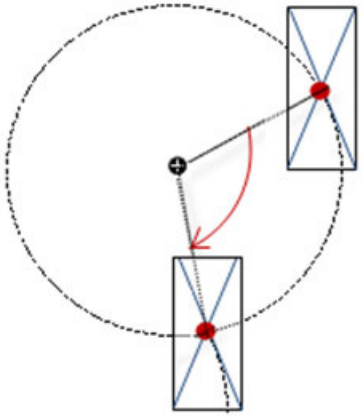
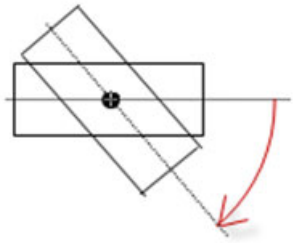
Element property 'State variables' The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Absolute movement' The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Histogram'

Symbol:

Category: *“Measurement Controls”*

The element displays the data of a one-dimensional array as a histogram. You can define specific colors for certain value ranges.

See also

- *Chapter 1.4.5.21.4 “Displaying Array Data in a Histogram” on page 2138*

Element properties

“Element name”	Example: `GenElemInst_35`
“Type of element”	*“Histogram”*
“Data array”	One-dimensional array with data displayed in this histogram. Example: `PLC_PRG.arr1`

Element property 'Subrange of array'

“Use subrange”	☑: Only part of the array is displayed in the histogram.
“Start index”	First array index with a displayed value. Requirement: *“Use subrange”* is activated.
“End index”	Last array index with a displayed value. Requirement: *“Use subrange”* is activated.

“Display type”	● *“Bars”*: Data is displayed as bars. ● *“Lines”*: Data is displayed as lines. ● *“Curve”*: Interpolation of data into a curve.
“Line width”	Specified in pixels Requirement: *“Curve”* is selected as the *“Display type”*.
“Show horizontal lines”	☑: Horizontal lines are drawn on the main scale. Note: Not all visualization styles have this property. This element property is not available for visualization styles that have striped backgrounds (example: *“Flat style”*).
“Relative bar width”	Integer value between `1` and `100` ● `1`: The bars are drawn as lines. ● `100`: The entire width of the histogram is filled with the bars.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Scale'

"Scale start"	Least value of the scale and the lower limit of the value range for the element. Example: `0` ◆: The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the scale start. Example: `PLC_PRG.iScaleStart`
"Scale end"	Greatest value of the scale and the upper limit of the value range for the element. Example: `100` ◆: The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the scale end. Example: `PLC_PRG.iScaleEnd`

"Main scale"	Distance between 2 values on the rough scale. Example: `10` : The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the distance. Example: `PLC_PRG.iMainScale`
"Subscale"	Distance between 2 values on the fine scale. You can hide the fine scale by setting the value to `0`. Example: `2` : The property *"Variable"* is shown below.
"Variable"	Variable (integer data type). Contains the spacing. Example: `PLC_PRG.iSubScale`
"Scale color"	Color of scale lines • ...: The *"Color"* dialog box opens. • : A drop-down list with color names opens.
"Base line"	Value of the main scale where the horizontal base line of the Histogram is located. The drawing of the bar starts at the base line.

Example A valid declaration is required for the variables used as an example in the table above.

```
PROGRAM PLC_PRG
VAR
    iScaleStart : INT := 0;
    iScaleEnd : INT := 120;
    iMainScale : INT := 20;
    iSubScale : INT := 5;
END_VAR
```

Element property 'Label'

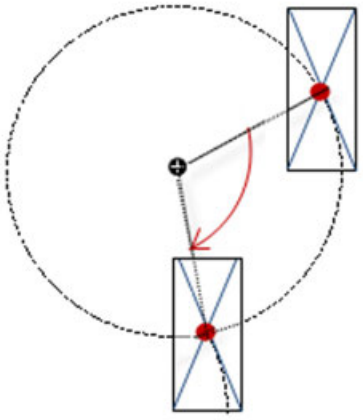
"Unit"	Text that is displayed in the element. Example: Units displayed in m/s.
"Font"	Font for labels (example: scale numbering). Selection from the drop-down list or by clicking the *""* button. ...
" Scale format (C Syntax)"	Values scaled in "printf" syntax Examples: `%d, %5.2f`
"Max. text width of labels"	Optional value that defines the maximum width of the scale label. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Text height of labels"	Optional value that defines the maximum height of the scale label. Note: Change this value only if the automatic adjustment does not yield the expected result.
"Font color"	Selection from the drop-down list or by clicking the ... button.

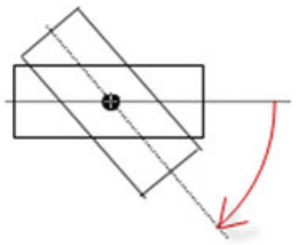
Element property 'Colors'

"Graph color"	Color of the bar in normal state. Note: The normal state is in effect when the current value of the array component does not fulfill the alarm condition.
"Alarm value"	Threshold for the alarm
"Alarm condition"	If the current value of the array component fulfills the alarm condition, then the alarm condition is set. ● *"Less"*: The current value is less than the *"Alarm value"* ● *"More"*: The current value is greater than the *"Alarm value"*
"Alarm color"	Color of the bar in alarm state.
"Use color areas"	☑: The color areas defined in this element are used.
"Color areas"	
"Create new"	A new color area is added.
" Delete"	The color area is removed from the list.
"Begin of area"	The start value on the *"Scale"* of the Histogram where the color area begins.
"End of area"	The end value on the *"Scale"* of the Histogram where the color area ends.
"Color"	Color that is used for displaying the area.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Image Switcher'

Symbol:

Category: *"Lamps/Switches/Bitmaps"*

The element displays one of three referenced images. Mouse actions change the displayed image. The images are defined in the *"Image settings"* element properties. The effects of mouse clicks are defined in the *"Element behavior"* property.

Element properties

“Element name”	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `ImageSwitcher_1`
“Type of element”	*“Image Switcher”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Variable”	Variable (`BOOL`). The value of the variable changes according to user input and it is independent of the *“Element behavior”* element property.

Image settings

“Image "on"”	Image ID from an image pool. The image can be selected using the input assistant. The image is used if the variable of the *“Variable”* property has the value `TRUE`.
“Image "off"”	Image ID from an image pool. The image can be selected using the input assistant. The image is used if the variable of the *“Variable”* property has the value `FALSE`.
“Image "clicked"”	Image ID from an image pool. The image is selected using the input assistant. In runtime mode, the visualization displays the referenced image when the element is clicked (and the mouse button is held down). Requirement: The *“Element behavior”* is *“Image toggler”*.

"Transparency"	☑: The *"Transparent color"* is selected.
"Transparent color"	The image pixels that have the transparent color are displayed as transparent. Requirement: *"Transparency"* is activated. ● ...The *"Color"* dialog box opens. ● ▼: A drop-down list with color names opens.
"Scaling type"	Defines how an image fits in the element frame. ● *"Fixed"*: The original size of the image is retained, regardless of the dimensions of the element. ● *"Isotropic"*: The entire image is shown in the element frame, either larger or smaller. As a result, the proportion of height and width are retained. ● *"Anisotropic"*: The image resizes automatically to the dimensions of the element frame, filling the entire element frame. As a result, the proportions are not retained.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

"Element behavior"	● *"Image toggler"*: Every mouse click switches the image. ● *"Image tapper"*: While a visualization user holds down the mouse button, the image of the *"Image on"* property is displayed. At the same time, the value `TRUE` is assigned to the *"Variable"* property.
"Tap FALSE"	☑: While the mouse button is pressed, the image of the *"Image"* property is displayed and the *"Variable"* property gets the value `FALSE` instead of the value `TRUE`, and back. Requirement: *"Image tapper"* is selected in the *"Element behavior"* property.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

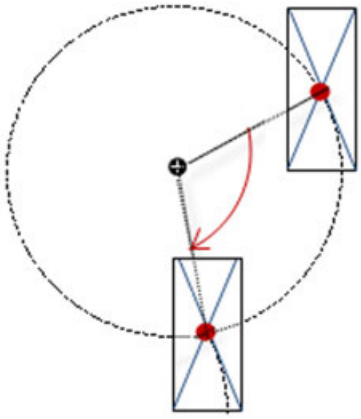
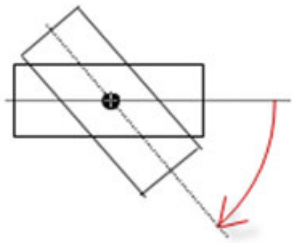
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access`.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Lamp'

Symbol:

Category: *“Lamps/Switches/Bitmaps”*

The element shows the value of a variable, and the element is displayed as illuminated or not.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Lamp_green`
"Type of element"	*"Lamp"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Variable"	Variable (`BOOL`). The variable value is displayed as a lamp that goes on (`TRUE`) or off (`FALSE`).

Image settings

"Transparency"	: The *"Transparent color"* property is selected.
"Transparent color"	Pixels in this color are displayed as transparent. Requirement: *"Transparency"* is activated. • The *"Color"* dialog box opens. • : A drop-down list with style colors opens.

"Scaling type"	Reaction of the element when the dimension of the *"Frame"* element is changed: ● *"Isotropic"*: The height and width of the image are resized proportionally to the *"Frame"*. Please note: To retain the alignment of elements also within a scaled *"Frame"* element, define the *"Horizontal alignment"* or *"Vertical alignment"* explicitly with *"Centered"*. ● *"Anisotropic"*: The image fills the entire *"Frame"* regardless of its proportions.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

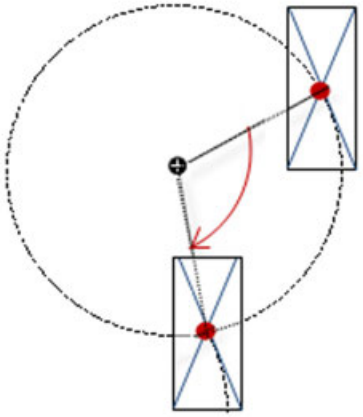
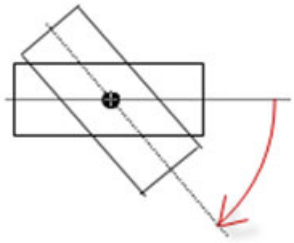
Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access`.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Background'

“Image”	Drop-down list with background colors Depends on the visualization style

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

● *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Dip Switch', 'Power Switch', 'Push Switch', 'Push Switch LED', 'Rocker Switch'

Symbols:

Category: *"Lamps/Switches/Bitmaps"*

The element assigns a value to a Boolean variable. The switch position "on" the value `TRUE` to the variable, and the switch position "off" assigns the value `FALSE`. Use the mouse to change the switch position.

Element properties

"Element name"	Example: `Operating_Switch` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	Depending on the element: *"Dip Switch"*, *"Power Switch"*, *"Push Switch"*, *"Push Switch LED"*, or *"Rocker Switch"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Variable"	Variable (`BOOL`) The value of the variables `TRUE` and `FALSE` indicates the switch position on/off.

Image settings

"Transparency"	☑: The *"Transparent color"* property is selected.
"Transparent color"	Pixels in this color are displayed as transparent. Requirement: *"Transparency"* is activated. ● ...: The *"Color"* dialog box opens. ● ▼: A drop-down list with style colors opens.
"Scaling type"	Reaction of the element when the dimension of the *"Frame"* element is changed: ● *"Isotropic"*: The height and width of the image are resized proportionally to the *"Frame"*. Please note: To retain the alignment of elements also within a scaled *"Frame"* element, define the *"Horizontal alignment"* or *"Vertical alignment"* explicitly with *"Centered"*. ● *"Anisotropic"*: The image fills the entire *"Frame"* regardless of its proportions.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

"Element behavior"	● *"Image toggler"*: Every mouse click changes the switch and the *" Variable"* value. ● *"Image tapper"*: The switch is "on" and the *"Variable"* value is `TRUE` while the mouse button is pressed.
"Tap FALSE"	☑: The value `TRUE` is assigned to the *"Variable"* property instead of the value `FALSE`, and back. Requirement: *"Image tapper"* is selected in the *"Element behavior"* property.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

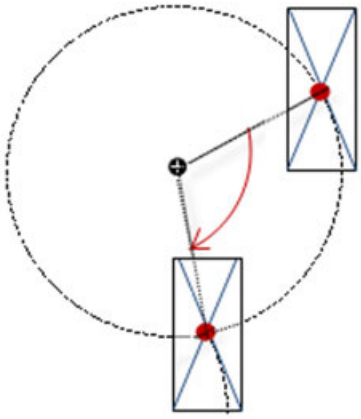
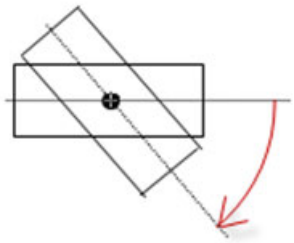
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access.`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Background'

“Image”	Drop-down list with background colors Depends on the visualization style

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Rotary Switch'

Symbol:

Category: *“Lamps/Switches/Bitmaps”*

The element assigns a value to a Boolean variable. The switch position "on" the value `TRUE` to the variable, and the switch position "off" assigns the value `FALSE`. Use the mouse to change the switch position.

Element properties

“Element name”	Example: `Operating_Switch` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Rotary Switch”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (■) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

“Variable”	Variable (`BOOL`). The value of the variables `TRUE` and `FALSE` indicates the switch position on/off.

Image settings

"Transparency"	☑: The *"Transparent color"* property is selected.
"Transparent color"	Pixels in this color are displayed as transparent. Requirement: *"Transparency"* is activated. ● ...The *"Color"* dialog box opens. ● ▼: A drop-down list with style colors opens.
"Scaling type"	Reaction of the element when the dimension of the *"Frame"* element is changed: ● *"Isotropic"*: The height and width of the image are resized proportionally to the *"Frame"*. Please note: To retain the alignment of elements also within a scaled *"Frame"* element, define the *"Horizontal alignment"* or *"Vertical alignment"* explicitly with *"Centered"*. ● *"Anisotropic"*: The image fills the entire *"Frame"* regardless of its proportions.
"Horizontal alignment"	Horizontal alignment of the image within the element frame or element ● `Left` ● `Centered` ● `Right` Requirement: *"Scaling type"* is *"Isotropic"*.
"Vertical alignment"	Vertical alignment of the image within the element frame or element ● `Top` ● `Centered` ● `Bottom` Requirement: *"Scaling type"* is *"Isotropic"*.

"Element behavior"	● *"Image toggler"*: Every mouse click changes the switch and the *" Variable"* value. ● *"Image tapper"*: The switch is "on" and the *"Variable"* value is `TRUE` while the mouse button is pressed.
"Orientation"	● *"At top"*: The rotary switch turns from the top right to the top left. ● *"At side"*: The rotary switch turns from the top right to the bottom right.
"Color change"	☑: The element changes in color when *" Variable"* is `TRUE`.

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

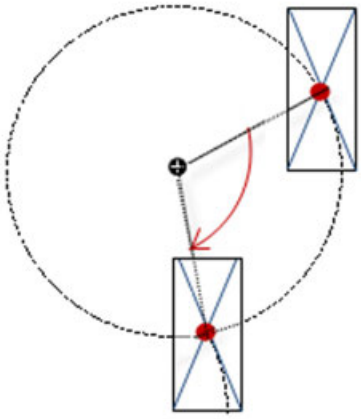
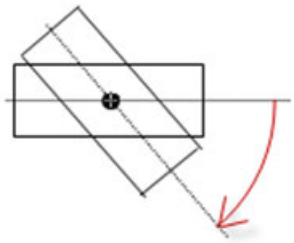
You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Texts'

"Tooltip"	String display as tooltip for the element Example: `Valid access`.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Background'

"Image"	Drop-down list with background colors Depends on the visualization style

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Trace'

Symbol:

Category: *“Special Controls”*

The element displays the graphical curve of variable values. In addition, variables can be configured to control the view.

See also

- *Chapter 1.4.5.10 “Displaying data curve with trace” on page 1306*
- *“Dialog box 'Trace Configuration'” on page 1734*

Element properties

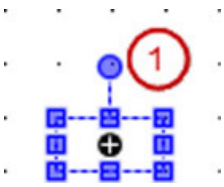
“Element name”	Example: `Velocity`
“Data Source”	Location where the trace data is buffered. : ● *“<local application>”*: The trace record is listed below the local application. The visualization that contains the trace is located below this application. When the application is downloaded, the trace configuration is downloaded to the local device. During execution, the data is stored locally in the trace buffer. ● *“ <data source name>”*: Data source that identifies the remote device where the trace record is created. When the local application is downloaded with the visualization, the trace configuration is downloaded to the **remote** device. During execution, the trace buffer is filled, and the trace data is transferred and then displayed in the local visualization as HMI. Example: `DataSoure_PLC_A` Note: The trace buffer is filled only if the remote application is being executed. The data recording is started when the local visualization is started.
“Application”	Application where data was recorded. : Lists all applications that are present below the data source. Requirement: A remote data source (not *“<local application>”*) is referenced in the *“Data source”* property.
“Type of element”	*“Trace”*
“Trace”	*“ <name of trace configuration>”*: Opens the *“Trace Configuration”* dialog where you can modify the trace configuration.

See also

- *“Dialog box 'Trace Configuration'” on page 1734*
- *Data Source Manager*

Element property 'Position'

The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Y"	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Width"	Specified in pixels Example: `150`
"Height"	Specified in pixels Example: `30`
	Tip: You can change the values in *"X"*, *"Y"*, *"Width"*, and *"Height"* by dragging the corresponding symbols to another position in the editor.
"Angle"	Static angle of rotation (in degrees) Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Show cursor"	☑: A cursor (vertical line) is displayed at the mouse position. The trigger and variable values where the cursor points are displayed as a tooltip.
"Overwrite existing trace on PLC"	☑: If a trace with the same name is on the PLC, then it is overwritten at download with the configuration that is defined here.
"Number format"	Number format of values in the tooltip in printf syntax (example: %d, %5.2f).

Element property 'Control variables'

The control variables are assigned automatically when you click *"Insert elements for controlling Trace"*.

“Reset Trigger”	Variable (`BOOL`). Standard control variable: `bResetTrigger` `TRUE`: Resets the triggering. After the action is executed, the variable is set automatically to `FALSE`.
“Start Trace”	Variable (`BOOL`). Standard control variable: `bStart` `TRUE`: Starts the Trace. After the action is executed, the variable is set automatically to `FALSE`.
“Stop Trace”	Variable (`BOOL`). Standard control variable: `bStop` `TRUE`: Stops the Trace. After the action is executed, the variable is set automatically to `FALSE`.
“Save Trace to a file”	
“Save Trace”	Variable (`BOOL`). Standard control variable: `bStore` `TRUE`: Saves the current trace configuration and the data that is stored in the development system to a file. When the action is ended, the variable is set automatically to `FALSE`.
“File name”	Variable (`STRING`) that contains the file name of the file to be saved. Standard control variable: `sStoreFilename`
“Load trace from file”	
“Load Trace”	Variable (`BOOL`). Standard control variable: `bRestore` `TRUE`: Reads the file specified below and loads its contents into the trace editor. The file contains a trace configuration and possibly also trace data. To do this, the stored trace configuration must match the application where the trace configuration is located. When the action is ended, the variable is set automatically to `FALSE`. Note: A trace configuration can be loaded from a file only under special circumstances. The file must have been created with exactly the same (running) application with which it will then be loaded. The consequence of changing the running application (for example by downloading again) is that a file which was previously created from the application cannot no longer be read into the application. Even external manual changes to the file can cause this. You should edit only those configuration settings that have an effect on displaying the variables. If you change variable definitions directly in the file (for example by replacing variable x with v y), then the file cannot be loaded.
“File name”	Variable (`STRING`) that contains the file name of the file to be read. Standard variable: `sRestoreFilename`

See also

- *Chapter 1.4.5.19.2.15 “Command 'Insert Elements for Controlling Trace'” on page 1737*

Element property 'Center'

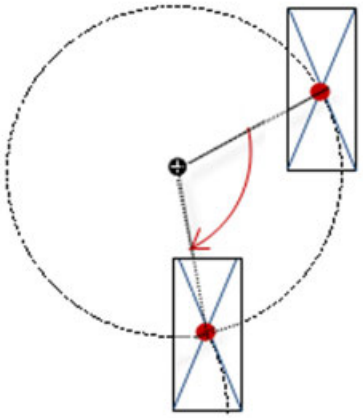
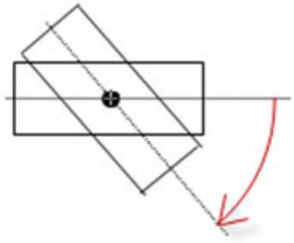
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Trend'

Symbol:

Category: *"Special Controls"*

The element displays the curve of variable values as a trend diagram. The trend diagram is suitable for representing a long-term data curve because the data is read from a trend recording and hence from a database. Moreover, you can run the *"Trend"* element together with the *"Date Range Picker"*, *"Legend"*, and *"Time Range Picker"* operating elements so that the user can navigate conveniently in the diagram.

You can programmatically delete the recorded trend curve at runtime. The recording starts again from the time of deletion. See the help page for "Programming a Trend Visualization".

Element properties

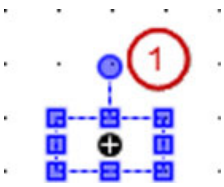
"Element name"	Example: `Velocity`
"Data source"	Data source for the connection via the device and the application to the *"Trend Recording"* object where the trend data that you want to show was saved. If the *"Trend Recording"* object is on the local device, then it is sufficient when you specify the respective application. If the trend recording is on a remote device, then you need to specify the data source connection to this device. ● *" <local application>"* The *"Trend Recording"* object is located on the local device in the local application. ● `<device name> . <application name>` Example: `Device_A.App_A` The *"Trend Recording"* object is located on the local device `Device_A` below the application `App_A`. ● `<data source name>` Example: `DataSource_B` The *"Trend Recording"* object is located on a remote device that is connected via the data source `DataSource_B`. Below the (now visible) *"Application"* property, the remote application is displayed as configured in the data source. Example: `App_B` Note: If the data source is accessed symbolically by means of a symbol file (CODESYS symbolic), then the required symbol file and the corresponding project have to be saved in the same folder.
"Type of element"	*"Trend"*
"Trend recording"	: Trend recording whose data is displayed as a diagram The trend recording is located on the device specified in the *"Data source"* property.
"Display Mode"	: Opens the *"Display Settings"* dialog.

See also

- *Chapter 1.4.5.11 "Displaying data curve with trend" on page 1309*
- *Chapter 1.4.5.19.2.16 "Command 'Configure Display Settings of Trend'" on page 1738*
- *Object 'Data Source'*

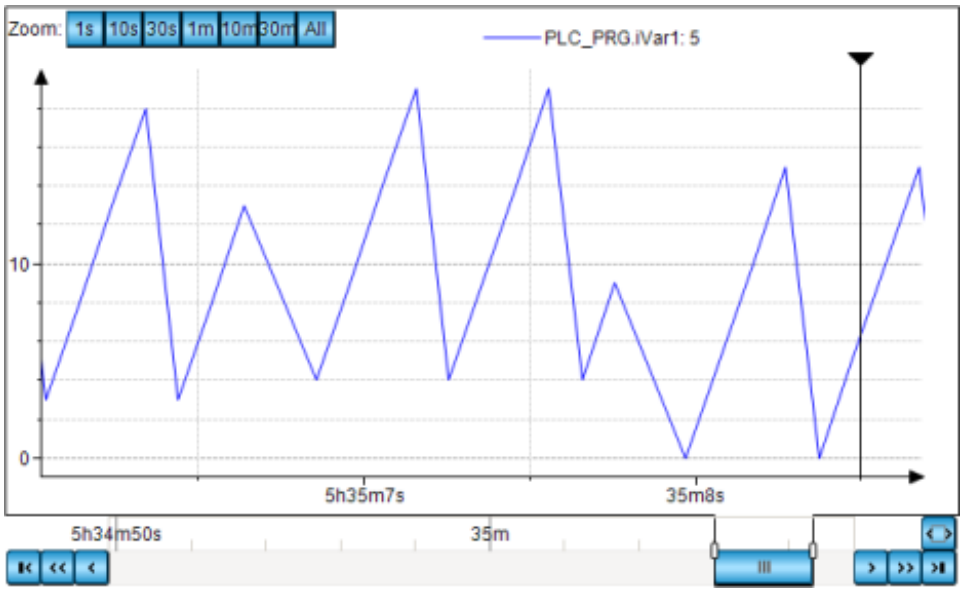
Element property 'Position'

The position defines the location and size of the element in the visualization window. This is based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	The x-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Y"	The y-coordinate of the upper left corner of the element Specified in pixels Example: `10`
"Width"	Specified in pixels Example: `150`
"Height"	Specified in pixels Example: `30`
	Tip: You can change the values in *"X"*, *"Y"*, *"Width"*, and *"Height"* by dragging the corresponding symbols to another position in the editor.
"Angle"	Static angle of rotation (in degrees) Example: `35` The element is displayed rotated in the editor. The point of rotation is the center of the element. A positive value rotates clockwise. Tip: You can change the value in the editor by focusing the element to the handle. When the cursor is displayed as a rotating arrow, you can rotate the element about its center as a handle.  (1): Handle Note: If a dynamic angle of rotation is also configured in the property *"Absolute movement ➔ Internal rotation"*, then the static and dynamic angles of rotation are added in runtime mode. The static angle of rotation acts as an offset.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

“Show cursor”	☑: A cursor (black triangle with vertical line) is shown in the trend diagram. Behavior at runtime: As soon as the graph is drawn, the user can move the cursor along the time axis in order to mark a specific time. Then the variable value belonging to the cursor position is displayed in the legend above the graph. 
“Show tooltip”	Requirement: *“Show cursor”* is activated. ☑: A tooltip opens at the cursor. Behavior at runtime: The variable value belonging to the cursor position is displayed as a tooltip.
“Show frame”	☑: The trend diagram is drawn with a frame.
“Number format”	Format specification in printf syntax, which determines how the values are displayed in the tooltip and in the legend Example: `%d` (integer variable) or `%5.2f` (floating-point number)

Element property 'Tick mark labels'

The time stored in the trend recording are in the UTC time zone. If the time is displayed in the trend of the visualization element, then the time stamps are converted to the local time zone of the operating system of the PLC.

Change the time zone in the operating system if the times in the trend diagram are not in the zone that you need.

“Time stamps”	X-value of the trend diagram ● *“Absolute time stamps”* The absolute time with date and time is displayed at each tick mark on the time axis. Example: `03/18/2016 12h30m50s` ● *“Relative time stamps”* The time period from the start of the recording (=0) is displayed at each tick mark. Example: `5m30s`
“Draw labels on two lines”	☑: The time stamps are displayed on two lines (for example, the date is displayed on the first line and the time on the second line). ☐: The time stamp is displayed on one line. Example: `2019-11-01-12:30:50.`
“ Omit irrelevant information in timestamps”	☑: The time stamps are displayed in a truncated form (without insignificant information). For example, the date is displayed at the first tick mark, and only the time is displayed at the following tick marks. The *“Internationalization (format strings)”* property is not visible and is ignored. ☐: The time stamps are displayed with all information. This takes into consideration the *“Internationalization (format strings)”* property which contains the format specification for the date and time display.
“Internationalization (format strings)”	Format specification for the date and time display of the time stamp (when it is displayed in full) Note: The property is visible only if the *“Omit irrelevant information in timestamps”* option is **not** selected.

"Date"	Format string that returns the date display according to the defined format. The operating system locale is used as the default setting.
	Defined format strings for the date:
	● Year: `yyyy`, `yy`, `y`
	● Month: `MM`, `M`
	● Day: `dd`, `d`
	● Recommended separator: `- . /`
	Example:
	`yyyy-MM-d` displays `2019-10-25`
	`yyyy-MM-dd` displays `2019-10-25`
	`dd.MM.yyyy` displays `25.10.2019`
	`dd/MM/yyyy` displays `25/10/2019`

"Time"	Format string that returns the time (or time of day) display according to the defined format. The operating system locale is used as the default setting.
	Defined format strings for the time:
	● 24-hour time definition: `HH`, `H`
	● 12-hour time definition: `hh`, `h`
	● AM/PM for 12-hour time definition: `tt`
	● Minutes: `mm`, `m`
	● Seconds: `ss`, `s`
	● Milliseconds: `ms`
	● Microseconds: `us`
	● Recommended separator: `:` or space character
	Example:
	`HH:mm:ss:ms` displays `15:30:59:123`
	`h:mm:ss tt` displays `3:30:59 PM`

See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Assigned control elements'

These elements are created automatically when the control elements are added with the command *"Insert elements for controlling Trend"*.

"Date Range Picker"	Control element for changing the date and time of the displayed data sets. With ..., all elements are provided that have implemented the interface `IDateRangeSelector`. By default, instances of the *"Date Range Picker"* visualization element are available.
"Time Range Picker"	Control element for changing the time of the displayed data sets. With ..., all elements are provided that have implemented the interface `ITimeSelector`. By default, instances of the *"Time Range Picker"* visualization element are available.
"Legend"	Control element for displaying a legend for the graphs. With ..., all elements are provided that have implemented the interface `ILegendDisplayer`.

See also

- *Chapter 1.4.5.19.2.18 "Command 'Insert Elements for Controlling the Trend'" on page 1739*

Element property 'Center'

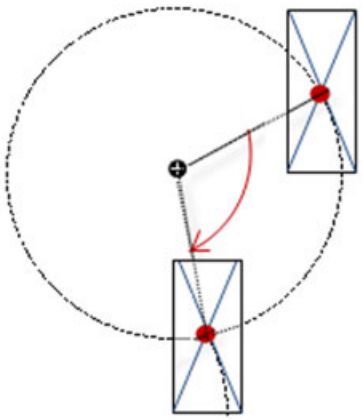
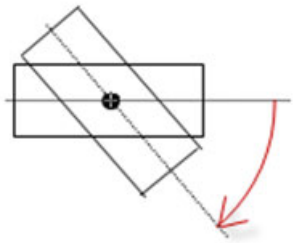
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.11 "Displaying data curve with trend" on page 1309*
- *Chapter 1.4.5.11.1 "Getting Started with Trend Visualization" on page 1309*
- *Chapter 1.4.5.11.2 "Programming a Trend Visualization" on page 1312*
- *Object 'Trend Recording'*

Visualization Element 'Legend'

Symbol:

Category: *"Special Controls"*

The element is used as a legend for another element (for example, a trend). The legend is assigned in the properties of the other element.

See also

- *Chapter 1.4.5.11 “Displaying data curve with trend” on page 1309*

Element properties

“Element name”	Example: `LegendOfTrendA` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Legend”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Orientation"	Orientation of the element. The value is configured in the assigned element. ● *"Horizontal"* ● *"Vertical"*
"Attached element instance"	Example: `Element_A`
"Show frame"	☑: The element is displayed with frames.
"Number format"	The format of the value in printf syntax (example: %d, %5.2f)

Element Property 'Layout'

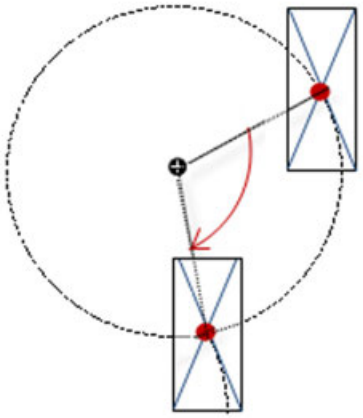
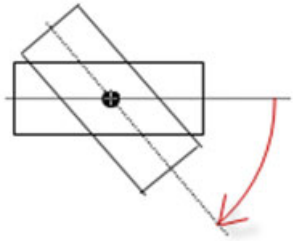
Defines how many variables can be displayed at a maximum and is calculated from the row and column number.	
"Max. number of rows"	Example: `3`
"Max. number of columns"	Example: `2`

Element Property 'Text properties' The property affects the text configured in the associated element.

"Text format"	*"Default"*: The text will be cut and displayed in only the part that fits into the visualization element. *"Linebreak"*: The text will be wrapped in rows. *"Ellipsis"*: The text is cut and ellipsis `...` are added to indicate that something is missing.
"Font"	Font of the text. The entries of the selection list are defined in the visualization style.
"Font color"	Text color, for example `Grey`. The entries of the selection list are defined in the visualization style.
"Transparency"	Transparency value (255 to 0), which defines the transparency of the corresponding color. Example: 255: The color is opaque. 0: The color is fully transparent.

Element property 'Absolute movement' The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.

"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The "X", "Y", "Rotation", and "Interior rotation" properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'ActiveX'

Symbol:

Category: *"Special Controls"*

The element is used to link an existing ActiveX control in the visualization. You can configure the method calls and their parameters in the element properties of the *"ActiveX"* element.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"ActiveX"*
"Element"	Installed ActiveX component that is linked to the visualization. Hint: To avoid typing errors, select the required ActiveX component by means of the Input Assistant.

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

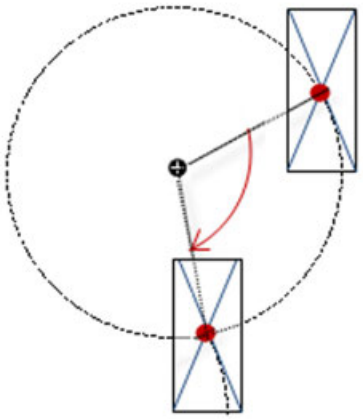
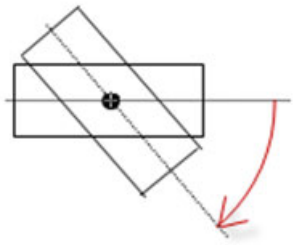
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”	
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right. 
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed. 

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Initial calls'

These method calls are executed during initialization. They are executed in the first cycle only.

“Method calls ”	Button '*“Create new”*' Creates a subnode below *“Methods”* with parameters for the method call.
“Methods”	*“[<number>]”* ● *“Method”*: Name of the method ● *“Parameter”*: Parameter passed at the method call ● *“Result parameter”*: Optional variable for the return value of the method

Element property 'Cyclic calls'

These method calls are executed in every cycle. They are executed in the refresh rate of the visualization.

“Method calls ”	Button '*“Create new”*' Creates a subnode below *“Methods”* for a method call and its parameters.
“Methods”	*“[<number>]”* ● *“Method”*: Name of the method ● *“Parameter”*: Parameter passed at the method call ● *“Result parameter”*: Optional variable for the return value of the method

Element property 'Conditional calls'

These method calls are executed in the refresh rate of the visualization. You define the call condition in the property *“Methods ➔ [<number>] ➔ Call condition”*.

“Method calls ”	Button '*“Create new”*' Creates a subnode below *“Methods”* with a call condition and parameters for the method call.
“Methods”	*“[<number>]”* ● *“Method”*: Name of the method ● *“Call condition”*: Variable (`BOOL`). A rising edge of this variable triggers the call of this method. ● *“Parameter”*: Parameter passed at the method call ● *“Result parameter”*: Optional variable for the return value of the method

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Web Browser'

Symbol:

Category: *“Special Controls”*

The element shows a website, PDF file, or video that has a URL.

NOTICE!
The display options of the *“Web Browser”* element depend on the operating system and the display variant of the visualization.

Requirement: The software components of the web browser are available in the runtime and configured accordingly (example: videos to be shown on Linux).

See also

- *Chapter 1.4.5.21.6 “Displaying Web Contents” on page 2141*

Element properties

“Element name”	Example: `GenElemInst_59`
“Type of element”	*“Web Browser”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

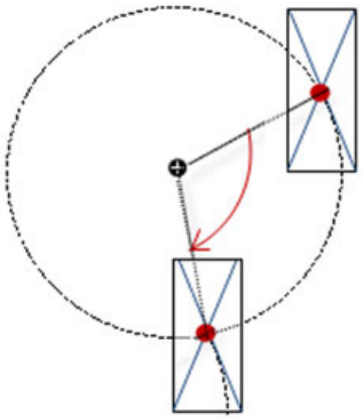
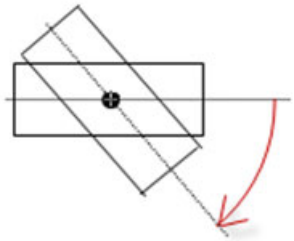
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

> *You can also change the values by dragging the symbols (⊕) to other positions in the editor.*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

Element property 'Control variables'

“URL”	URL of the web page that is displayed in the visualization. ● Variable (`STRING` or `WSTRING`) Example: `PLC_PRG.stURL` ● Literal in single straight quotation marks Example: `'http://de.wikipedia.org'`
“Show”	Variable (`BOOL`). Example: `PLC_PRG.bSetURL` Controls the display of the *“Web browser”* element. If the variable contains a rising edge, then the visualization calls the web page given in *“URL”* and displays its contents in the 'Web browser' visualization element.
“Back”	Variable (`BOOL`). Example: `PLC_PRG.bGoBack` Controls the back navigation in the *“Web browser”*. If the variable has a rising edge, then the visualization displays the contents of the previously displayed page.
“Forward”	Variable (`BOOL`). Example: `PLC_PRG.bGoForward` Controls the forward navigation in the *“Web browser”*. If the variable has a rising edge, then the visualization displays the contents of the previously displayed page.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Busy Symbol, Cube'

Symbol:

Category: *“Special Controls”*

At runtime, this element indicates automatically that the runtime is busy or waiting for data.

Element properties

"Element name"	Example: `Data_Transfer` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Busy Symbol, Cube"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

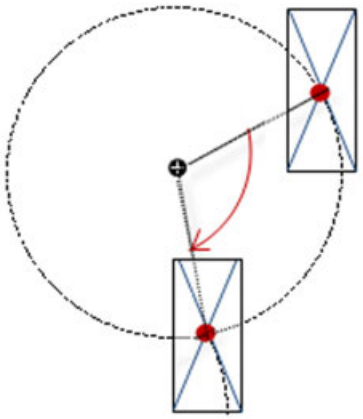
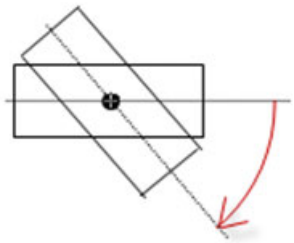
The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- ↳ *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Busy Symbol, Flower'

Symbol:

Category: *“Special Controls”*

The element indicates that the system is busy or waiting for data.

Element properties

“Element name”	Example: `Data_Transfer` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
“Type of element”	*“Busy Symbol, Flower”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols (⊞) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

"Frame color"	
"Fill color"	
"Transparency"	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.

See also

- *Chapter 1.4.5.19.3.5 "Dialog 'Gradient Editor'" on page 1748*
- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

"Line width"	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of 1 pixel. If no line should be displayed, then the *"Line style"* property must be set to the option *"Invisible"*.
"Fill attributes"	The way in which the element is filled. ● *"Filled"*:The element is filled with the color from property *"Colors ➔ Fill color"*. ● *"Invisible"*: The fill color is invisible.
"Line style"	Type of line representation ● *"Solid"* ● *"Dashes"* ● *"Dots"* ● *"Dash Dot"* ● *"Dash Dot Dot"* ● *"not visible"*

You can assign variables in the "Appearance variables" property for controlling the appearance dynamically. The fixed values here are overwritten.

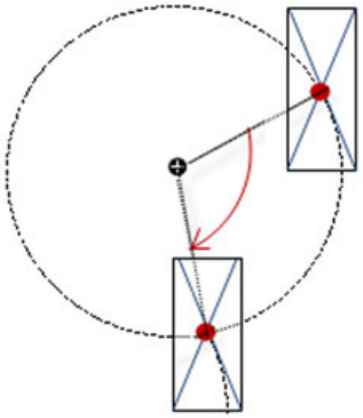
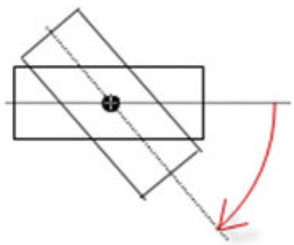
See also

- *" Element property 'Appearance variables'" on page 2095*

“Symbol color”	Selection of a color for the flower symbol.
“Line”	Stroke width of the lines (in pixels).

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Text Editor'

Symbol:

Category: *"Special Controls"*

The element shows the contents of text files that are saved on the controller. Files can be encoded in ASCII or Unicode formats.

A visualization user can also edit the text.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Text Editor"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Font'

“Font name”	Non-proportional font used by the visualization to display the contents of the file Example: *“Courier New”*
“Size”	Font size Example: `12`

Element property 'Control variables'

Table 361: Element property “Control variables --> File”

“Variable”	Variable (`STRING`). Contains the file names and optionally the location of the file. It is located in the file system of the controller. Example: `PLC_PRG.strFile: STRING := '/Documentation/Info.txt';`
“Open”	Variable (`BOOL`). Controls opening the file which is defined in the *“Variable”* property Example: `bOpen: BOOL;` `TRUE`: The file is opened.
“Close”	Variable (`BOOL`). Controls closing the file which is defined in the *“Variable”* property Example: `bClose: BOOL;` `TRUE`: The file is closed.
“Save”	Variable (`BOOL`). Controls saving the file which is defined in the *“Variable”* property Example: `bStore: BOOL;` `TRUE`: The file is saved.
“New”	Variable (`BOOL`). Controls creating a new file. The name is defined in the *“Variable”* property. Example: `bCreate: BOOL;` `TRUE`: A file is created and opened.

Table 362: Element property “Control variables --> Edit ”

“Variable”	Variable (`STRING`). Contains the string to search for in the file Example: `strFind: STRING := 'abc';`
“Find”	Variable (`BOOL`). Controls executing the search for the string in the *“Variable”* property Example: `bFind: BOOL;` `TRUE`: The search is performed. The variable is automatically reset to `FALSE`.
“Find next”	Variable (`BOOL`). Controls the location to begin the search in the file Example: `bFindNext: BOOL;` `TRUE`: The search begins at the last search result location. `FALSE`: The search begins at the beginning of the file.

Table 363: Element property “Control variables --> Cursor position”

“Line”	Variable (integer data type). Contains the line of the cursor Example: `iRowCursor: INT;`
“Column”	Variable (integer data type). Contains the column of the cursor Example: `iColumnCursor: INT;`
“Position”	Output variable (integer data type). Shows the **absolute** cursor position in the text. Example: `iPosCursor: INT;`
“Set cursor”	Variable (`BOOL`). Controls the setting of the cursor at a specific location Example: `iSetCursor: INT;` `TRUE`: The cursor is moved. The new position is defined in the *“Line”* and *“Column”* properties. `FALSE`: The *“Line”*, *“Column”*, and *“Position”* properties contain the actual values. Note: The variable is used as the control variable for an input event triggered by a visualization user.

Table 364: Element property “Control variables --> Selection”

“Start position”	Output variable (integer data type). Shows the **absolute** position for starting the text selection Example: `iPosSelection: INT;`
“End position”	Output variable (integer data type). Shows the **absolute** position for ending the text selection. Example: `iPosEndSelection: INT;`
“Start line number”	Output variable (integer data type). Shows the line where the text selection begins Example: `iRowSelection: INT;`
“Start column index”	Output variable (integer data type). Shows the column where the text selection begins Example: `iColumnSelection: INT;`
“End line number”	Output variable (integer data type). Shows the line where the text selection ends Example: `iRowEndSelection: INT;`

"End column index"	Output variable (integer data type). Shows the column where the text selection ends Example: `iColumnEndSelection: INT;`
"Line to select"	Variable (integer data type). Contains the line number that is selected Note: The selection is controlled by the variables in the *"Trigger selection"* property.
"Set selection"	Variable (`BOOL`). Controls the selection of a line. Example: `bSetSelection: BOOL;` `TRUE`: The line from the *"Line to select"* property is selected and highlighted in the Text Editor. if the line is not in the current text segment of the Text Editor, then the text segment is moved to this line. Note: The variable is used as the control variable for an input event triggered by a visualization user. The control variable is not reset automatically. You are responsible for this to occur in the visualization.

Table 365: Element property "Control variables --> Error handling"

"Variable for error code"	Variable (integer data type). Contains the error code when an error occurs Example: `iError: INT;` The error codes are declared in `GVL_ErrorCodes` in the `VisuElemTextEditor` library. To display the error text, the `VisuFctTextEditorGetErrorText()` function of the library must be called.

"Variable for content changed"	Variable (`BOOL`). Shows whether the contents have changed Example: `bIsContentEdited: BOOL;` `TRUE`: The contents of the Text Editor have changed.
"Variable for access mode"	Variable (`BOOL`). Controls the access privileges to the file Example: `bIsReadOnly: BOOL;` `TRUE`: A visualization user has read-only permission. At runtime, the file contents are highlighted in gray in the Text Editor. `FALSE`: A visualization user has read/write permission. Note: The variable overwrites the setting in the *"Editor mode"* property.

"Maximum line length"	Maximum number of characters per line
"Editor mode"	● *"Read-only"*: A visualization user has read-only permissions to the file. At runtime, the file contents are highlighted in gray in the text editor. ● *"Read/write"*: A visualization user has read-write permissions.

Element property 'New files'

"Encoding"	Character encoding of the new file: ● *"ASCII"* ● *"Unicode (Little endian)"* ● *"Unicode (Big endian)"*
"New line character sequence"	End of line character of the new file: ● *"CR/LF"*: Normal for Windows systems ● *"LF"*: Normal for UNIX systems Please note: When a visualization user opens an existing file, the end-of-line character of the file is detected and used automatically.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

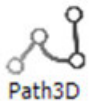
- ↳ *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*

Visualization Element 'Path3D'

Symbol:



Category: *“Special Controls”*

The *“Path3D”* visualization element graphically displays the curves of two independent records as a 3D path. It is specially designed for use with Motion Solution CNC in order to display the trajectory of a machine tool or a robot. The programmed path (path) and the path actually traveled (track) is displayed.

Although the visualization element is designed for use with Motion Solution CNC, it can also be used to display any other record. In this case the application has to provide the path data. The sample application `3D Path Generator`, which is available in CODESYS Forge, shows how these data can be generated.

If the element is used together with SoftMotion CNC, then function blocks from the library `SM3_CNC_Visu` help to generate the data from the path and track. These function blocks are used by the sample project `CNC_File_3DPath`, which is stored in the installation directory of CODESYS.

- `SMC_PathCopier`
- `SMC_PathCopierCompleteQueue`
- `SMC_PathCopierFile`
- `SMC_PositionTracker`

A description of the function blocks can be found in the Library Manager in the library `SM3_CNC_Visu`.

The element does not work with the CODESYS HMI display variant.

See also

- *CNC Example 6: Using Path3D with SoftMotion CNC*
- *Sample project in CODESYS Forge*

Element properties

“Element name”	Example: `GenElemInst_1`
“Type of element”	*“Path3D”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols () to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Path description

"Path data (VisuStruct3DTrack)"	Variable of the type `VisuStruct3DTrack`, which is declared in the IEC code. Example: `PLC_PRG.pc.vs3dt`. A description of the structure can be found in the library manager in the library `VisuElem3DPath.library`. The data structure describes a path or track through a certain number of points. The points are determined and buffered by the application. The track typically displays the last n positions, so that only a certain part of them is ever displayed at any one time. `VisuStruct3DTrack.pProjection` is a variable that is set by the visualization element and contains information about the path/track projection. It can be read (only) by the application. In addition, the methods `Projection.Apply` or `.ApplyV` can be used in order to see whether the transformed position lies inside or outside the visualization display area, which is defined by `Projection.ElementRect`.
"Path color"	Color of the path drawn
"Path line width"	Path line width in pixels, e.g.: "2"
"Style of boundary points"	Display of the points between two successive objects in the path ● End points are not displayed ● End points are marked with a circle ● End points are marked with a cross ● End points are marked with a plus

Track description The track data are structured in exactly the same way as the path data: VisuStruct3DTrack

"Track data (VisuStruct3DTrack)"	Variable of the type `VisuStruct3DTrack`, which is declared in the IEC code. Example: `PLC_PRG.pc.vs3dt`. A description of the structure can be found in the library manager in the library `VisuElem3DPath.library`.
"Track color"	Color of the track drawn
"Track line width"	Track line width in pixels, e.g.: "2"

Camera control The camera position for the 3D mode is controlled with a reference to the external data structure. This structure allows the following operations:

- Shifting to the left/to the right/upwards/downwards
- Rotation around the X/Y/Z axis
- Resetting of the view at X/Y, Y/Z or Z/X level, so that the path and the track are completely visible.

"Control data (VisuStruct3DControl)"	Variable of the type `VisuStruct3DControl`, which is declared in the IEC code. Example: `PLC_PRG.pc.vs3dc`. A description of the structure can be found in the library manager in the library `VisuElem3DPath`. The values can be set via the application itself or via the visualization element *"ControlPanel"*. The library `VisuElem3DPath` contains ready-to-use visualization frames that provide a possible user interface for these data.

Additional aspects

"Coordinate system"	☑: The coordinate system is displayed
"Grid"	☑: Grid lines are displayed
"Grid color"	Color of the grid lines

Highlighting Individual parts of the path can be visually highlighted. This is typically used to mark the already processed part of a track with a different color. Each point in the path is given a unique ID, which in the case of a CNC editor is linked with the object ID on which the point lies. This ID ("highlight ID") can be specified via the application so that dynamic elements/parts of the track can be highlighted.

Highlight mode	Select one of the following highlight modes: ● Only the element whose ID corresponds to the value of the variable is highlighted. ● All elements whose ID (linked with the object ID in the case of a CNC editor) is smaller than or equal to the value in Variable are highlighted.
Variable	Project variable that specifies the ID of an element. Example: `PLC_PRG.iVarElementID`. This "highlight ID" is taken into account for the setting of the highlight mode. The variable must be set in the IEC application.
Highlight color	

Element look

"Frame line width"	Width of the frame around the element, in pixels, for example: "1"
"Frame line style"	Select one of these style types for the frame line: ● `Solid` ● `Dashes` ● `Dots` ● `Dash Dot` ● `Dash Dot Dot` ● `Hollow`

"Transparent background"	☑: The background of the element is displayed transparently. ☐: The background of the element is displayed in the defined background color.
"Background color"	

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Control Panel'

Symbol:

Category: *“Special Controls”*

This visualization element is used in connection with the *“Path3D”* visualization element. It is used for changing the position and orientation to the CNC path shown with *“Path3D”*.

See also

- *Chapter 1.4.5.19.5.42 “Visualization Element 'Path3D'” on page 2082*

Element properties

“Element name”	Optional. Hint: Assign individual names for elements so that they are found faster in the element list. Example: `Camera_Path_1`
“Type of element”	*“Frame”*
“Clipping”	☑: If you have set the *“Scaling type”* to *“Fixed”*, then only that part of the visualization is displayed that fits in the frame.
“Show frame”	Displays the frame ● *“No frame”*: The displayed area of the frame does not have borders. ● *“Frame”*: The displayed area of the frame has borders. ● *“No frame with offset”*: The displayed area of the frame does not have a border and the displayed area of the referenced visualization is reduced inwards by one pixel as compared to the frame area. The resulting gap prevents the referenced visualization from touching any adjacent elements.
“Scaling type”	Describes how the frame reacts when the visualization is resized: ● *“Isotropic”*: The frame retains its proportions. This allows the ratio of height to width to be preserved, even if the height and width of the visualization have been changed separately. ● *“Anisotropic”*: The frame depends on the size of the visualization, so that height and width of the referenced visualization can be changed separately. ● *“Fixed”*: The original size of the frame is retained, regardless of the visualization size. If you have also selected the *“Clipping”* option, then only the fitting part is displayed. ● *“Fixed and scrollable”*: The referenced visualization is displayed without scaling. If the value is greater than the window area of the frame, then scrollbars are added to the frame. To set the position of the scroll bar with a variable, use the *“Scroll position variable horizontal”* or *“Scroll position variable vertical”* property.
“Deactivation of the background drawing”	☐: To optimize the performance of the visualization, the non-animated elements of the frame element are drawn as a background bitmap. This could result in the elements not being displayed in the expected order. ☑: Deactivation of the background drawing. This can prevent the behavior described above.

Element property 'References'

Contains the currently configured visualization references as a subnode

"References"	Clicking *"Configure"* opens the *"Frame Configuration"* dialog. This is used to manage the referenced visualizations. Caution: Visualizations can be nested at any depth by means of Frame elements. In order to use the *"Switch to any visualization"* Frame selection type without any problems, a Frame must not contain more than 21 referenced visualizations. For more information, see also the description for the *"Input configuration"* of an element: Action *"Switch Frame visualization"*.
List of the currently referenced visualizations	Visualizations that have a button also have this displayed as a subnode. Each interface variable is listed with the currently assigned transfer parameters. Example: `vis_FormA` ● `iDataToDisplay_1 : PLC_PRG.iVar1` ● `iDataToDisplay_2 : PLC_PRG.iVar2` Hint: You can change the assignment of the variables to an interface variable here and edit the value field. Or click the *"Configure"* button instead.

See also

- *Chapter 1.4.5.19.2.1 "Command 'Interface Editor'" on page 1719*
- *Chapter 1.4.5.15 "Creating a structured user interface" on page 1321*
- *"Input action 'Switch Frame Visualization'" on page 1756*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Colors'

The properties contain fixed values for setting colors.

“Color”	Color for the element in its normal state. Please note that the normal state is in effect if the expression in the *“Color variables ➔ Toggle color”* property is not defined or it has the value `FALSE`.
“Alarm color”	Color for the element in alarm state. Please note that the alarm state is in effect if the expression in the *“Color variables ➔ Toggle color”* property has the value `TRUE`.
“Transparency”	Value (0 to 255) for defining the transparency of the selected color. Example `255`: The color is opaque. `0`: The color is completely transparent.
“Use gradient color”	☑: The element is displayed with a color gradient.
“Gradient setting”	The *“Color gradient editor”* dialog box opens.
“Frame color”	Example: *“Black”*
“Fill color”	Example: *“Light gray”*

See also

- *Chapter 1.4.5.19.3.5 “Dialog 'Gradient Editor'” on page 1748*
-

Element property 'Appearance'

The properties contain fixed values for setting the look of the element.

“Line width”	Value in pixels Example: `2` Note: The values `0` and `1` both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Type of line representation ● *“Solid”* ● *“Dashes”* ● *“Dots”* ● *“Dash Dot”* ● *“Dash Dot Dot”* ● *“not visible”*

You can assign variables in the “Appearance variables” property for controlling the appearance dynamically. The fixed values are defined here.

See also

- “ *Element property 'Appearance variables'” on page 2095*

Element property 'Texts'

The properties contains character strings for labeling the element. The character string can also contain a placeholder with a format definition. In runtime mode, the placeholder is replaced by the current value in the specified format.

CODESYS accepts the specified texts automatically into the *“GlobalTextList”* text list. Therefore, these texts can be localized.

“Text”	Character string (without single straight quotation marks) for the labeling the element. Add a line break by pressing the keyboard shortcut *[Ctrl] + [Enter]*. Example: `Accesses: %i` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Text”*.
“Tooltip”	Character string (without single straight quotation marks) that is displayed as the tooltip of an element. Example: `Number of valid accesses.` The variable that contains the current value for the placeholder is specified in the property *“Text variable ➔ Tooltip”*.

See also

- *“Element property 'Text variables'” on page 2091*
- *Chapter 1.4.5.3 “Designing a visualization with elements” on page 1254*
- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*

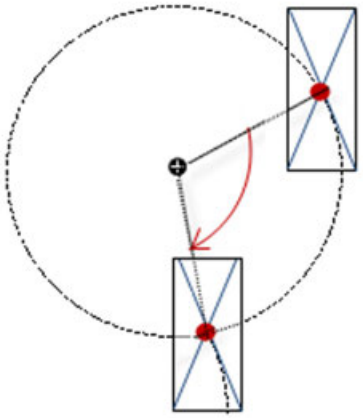
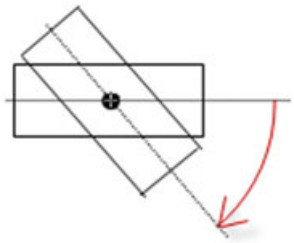
Element property 'Text properties'

The properties contain fixed values for the text properties.

“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Text format”	Definition for displaying texts that are too long ● *“Default”*: The long text is truncated. ● *“Line break”*: The text is split into parts. ● *“Ellipsis”*: The visible text ends with "..." indicating that it is not complete.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'Relative movement'

The properties contains variables for moving the element. The reference point is the position of the element (*“Position”* property). The shape of the element can change.

“Movement top-left”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **left** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaX`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **top** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaY`
“Movement bottom-right”	
“X”	Variable (integer data type). It contains the number (in pixels) that the **right** edge is moved horizontally. Incrementing the value moves the element to the right. Example: `PLC_PRG.iDeltaWidth`
“Y”	Variable (integer data type). It contains the number (in pixels) that the **bottom** edge is moved vertically. Incrementing the value moves the element to the down. Example: `PLC_PRG.iDeltaHeight`

See also

- *“Element property 'Absolute movement'” on page 2117*

Element property 'Text variables'

These properties are variables with contents that replace a format definition.

“Text variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccesses` Note: The format definition is part of the text in the property *“Texts ➔ Text”*. Note: If you specify a variable of type enumeration with text list support, then the name of the enumeration data type is added automatically in angle brackets after the variable name. Example: `PLC_PRG.enVar <enumeration name>`. Then the symbolic value of the enumeration component is printed instead of the numeric value when text is printed. Refer to the help page for the enumerations.
“Tooltip variable”	Variable (data type compliant with the format definition). It contains what is printed instead of the format definition. Example: `PLC_PRG.iAccessesInTooltip` Note: The format definition is part of the text in the property *“Texts ➔ Tooltip”*.

See also

- *Chapter 1.4.5.18.2 “Placeholders with Format Definition in the Output Text” on page 1708*
- *“Element property 'Texts'” on page 2089*
- *Chapter 1.4.1.19.5.17 “Enumerations” on page 676*

Element property 'Dynamic texts'

Dynamic texts are variably indexed texts of a text list. At runtime, the text is displayed that is currently indexed in the variable.

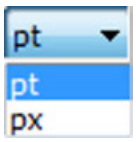
“Text list”	Variable (string) or name of the text list as a fixed string in single straight quotation marks. Example: `'Errorlist'` ▾: Drop-down list with the dialogs available in the text lists.
“Text index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'1'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strTextID` Sample assignment: `PLC_PRG.strTextID := '1';`
“Tooltip index”	Text list ID. This refers to the desired output text. ● As fixed string with the ID in single straight quotation marks. Example: `'2'` ● As a variable (`STRING`) for dynamically controlling the text output. Example: `strToolTipID` Sample assignment: `PLC_PRG.strToolTipID := '2';`

See also

- *Chapter 1.4.1.20.2.24 “Object 'Text List'” on page 927*

Element property 'Font variables'

The variables allow for dynamic control of the text display.

"Font name"	Variable (`STRING`). Includes the font of the text. Example: `PLC_PRG.stFontVar := 'Arial';` The selection of fonts corresponds to the default *"Font"* dialog.
"Size"	Variable (numeric data type). Contains the font size (in pixels or points). The applied unit is specified in brackets after the variable name. ● <pt>: Points (default) Example: `PLC_PRG.iFontHeight <pt>` Code: `iFontHeight : INT := 12;` ● <px> : Pixels Example: `PLC_PRG.iFontHeight <px>` Code: `iFontHeight : INT := 19;`  If you click in the value field, a drop-down list opens on the right for setting the unit. Hint: The font size is specified in points (example: Arial 12). Use points when the variable font size should match a font, for example if a font is set in the property *"Text property ➔ Font"*.
"Flags"	Variable (`DWORD`). Contains the flags for displaying fonts. Flags: ● `1`: Italics ● `2`: Bold ● `4`: Underline ● `8`: Strikethrough Note: You can combine the font displays by adding the coding of the flags. For example, a bold and underlined text: `PLC_PRG.dwFontType := 6;`
"Character set"	Variable (`DWORD`). Contains a character set number for the font. The selection of character set numbers corresponds to the *"Script"* setting of the standard *"Font"* dialog.
"Color"	Variable (`DWORD`). Includes the color of the text. Example: `PLC_PRG.dwColorFont:= 16#FF000000;`
"Flags for text alignment"	Variable (integer data type). Contains the coding for text alignment. Example: `PLC_PRG.dwTextAlignment.` Coding: ● `0`: Top left ● `1`: Horizontal center ● `2`: Right ● `4`: Vertical center ● `8`: Bottom Note: You can combine the text alignments by adding the coding of the flags. For example, a vertical and horizontal centered text: `PLC_PRG.dwFontType := 5;`

Fixed values for displaying texts are set in “Text properties”.

See also

- *“Element property 'Text properties'” on page 2089*

Element property 'Color variables'

The Element property is used as an interface for project variables to dynamically control colors at runtime.

“Toggle color”	The property controls the toggled color at runtime. Value assignment: ● `FALSE`: The element is displayed with the color specified in the *“Color”* property. ● `TRUE`: The element is displayed with the color specified in the *“Alarm color”* property. Assignment options: ● Placeholder for the user input variable – *“<toggle/tap variable>”* – *“<NOT toggle/tap variable>”* The color change is not controlled by its own variable, but by a user input variable. Note: Specify a variable for the mouse events *“Tap”* or *“Toggle”* in the input configuration of the element. Only then is the placeholder set. If you configure a variable in both *“Toggle”* and *“Tap”*, then the variable specified in *“Tap”* is used. Hint: Click the symbol to insert the placeholder *“<toggle/tap variable>”*. When you activate the *“Inputconfiguration”*, *“Tap FALSE”* property, then the *“<NOT toggle/tap variable>”* placeholder is displayed. ● Instance path of a project variable (`BOOL`) Example: `PLC_PRG.xColorIsToggeled` Note: In the code, declare and implement the variable specified here. Its value assignment determines when the color changes.
“Normal state” *“Alarm state”*	The properties listed below control the color depending on the state. The normal state is in effect if the variable in *“Color variables”*, *“Toggle color”* is not defined or it has the value `FALSE`. The alarm state is in effect if the variable in *“Colorvariables”*, *“Toggle color”* has the value `TRUE`.
“Frame color”	Assignment options: ● Variable (`DWORD`) for the frame color Example: `PLC_PRG.dwBorderColor` ● Color literal Example of green and opaque: `16#FF00FF00`
“Filling color”	Assignment options: ● Variable (`DWORD`) for the fill color Example: `PLC_PRG.dwFillColor` ● Color literal Example of gray and opaque: `16#FF888888`

The transparency part of the color value is evaluated only if the “Activate semi-transparent drawing” option of the visualization manager is selected.

Select the “Advanced” option in the toolbar of the properties view. Then all element properties are visible.

See also

- *Chapter 1.4.5.8.3 “Animating a color display” on page 1295*

Element property 'Appearance variables'

The properties contain variables for controlling the appearance of the element dynamically.

“Line width”	Variable (integer data type). Contains the line weight (in pixels). Note: The values 0 and 1 both result in a line weight of one pixel. If no line should be displayed, then the *“Line style”* property must be set to the option *“Invisible”*.
“Line style”	Variable (DWORD). Controls the line style. Coding: ● `0`: Solid line ● `1`: Dashed line ● `2`: Dotted line ● `3`: Line type "Dash Dot" ● `3`: Line type "Dash Dot Dot" ● `8`: Invisible: The line is not drawn.

Fixed values can be set in the “Appearance” property. These values can be overwritten by dynamic variables at runtime.

See also

- *“Element property 'Appearance'” on page 2088*

Element property 'Switch frame variable'

The variable controls the switching of the referenced visualizations. This variable indexes one of the referenced frame visualizations and this is displayed in the frame. When the value of the variable changes, it switches to the recently indexed visualization.

"Variable"	● Variable (integer data type) that contains the index of the active visualization Example: `PLC_PRG.uiIndexVisu` Hint: The *"Frame Configuration"* dialog includes a list of referenced visualizations. The visualizations are automatically numerically indexed via the order in the list. Note: This variant of switching usually affects all connected display variants. ● Array element (integer data type) for index access via `CURRENTCLIENTID` Example: `PLC_PRG.aIndexVisu[CURRENTCLIENTID]` Note: This variant of switching applies to the current client only, and therefore only on one display variant. That is the display variant where the value change was triggered (for example, by means of user input).

See also

- *Chapter 1.4.5.19.2.9 "Command 'Frame Selection'" on page 1727*

Element property 'State variables'

The variables control the element behavior dynamically.

"Invisible"	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
"Deactivate inputs"	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The "Invisible" property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *"Support client animations and overlay of native elements"* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Input configuration'

The properties contain the configurations for the user input when using the mouse or keyboard. A user input defines an event and one or more actions that are executed when an event occurs.

The *"Configure"* button opens the *"Input Configuration"* dialog. There you can create or edit user inputs. Configured user inputs are listed below the events. They each include the action that is triggered and the setting in short form. Example: *"Execute ST Code"*: `PLC_PRG.i_x := 0;`	
"OnDialogClosed"	Input event: The user closes the dialog.
"OnMouseClick"	Input event: The user clicks the mouse button completely in the element area. The mouse button is clicked and released.
"OnMouseDown"	Input event: The user clicks down on the mouse button.
"OnMouseEnter"	Input event: The user drags the mouse pointer to the element.
"OnMouseLeave"	Input event: The user drags the mouse pointer away from the element.

"OnMouseMove"	Input event: The user moves the mouse pointer over the element area.
"OnMouseUp"	Input events: ● The user releases the mouse button within the element area. It is irrelevant whether the user has previously pressed the mouse button inside or outside the element area. ● The user presses the mouse button within the element area, leaves the element area, and then releases the mouse button. Note: This CODESYS-specific triggering behavior guarantees that actions for key elements are completed. A key element starts an action for *"OnMouseDown"* and ends the action for *"OnMouseUp"*. Example: A visualization user presses the mouse button within the element area of the key element and then moves the cursor position so that it lies outside the element area. The action is ended anyway because *"OnMouseUp"* is triggered.

"Tap"	When a mouse click event occurs, the variable defined in *"Variable"* is described in the application. The coding depends on the *"Tap FALSE"* and *"Tap on enter if captured"* options.
"Variable"	Variable (`BOOL`) that is set on mouse click event. Example: `PLC_PRG.bIsTapped` `TRUE`: A mouse click event exists. It lasts as long as the user presses the mouse button over the element. It ends when the button is released. `FALSE`: A mouse click event does not exist. Requirement: The *"Tap FALSE"* option is not activated.
"Tap FALSE"	☑: The mouse click event leads to a complementary value in *"Variable"*. `TRUE`: A mouse click event does not exist. `FALSE`: While the mouse click event exists.
"Tap on enter if captured"	☑: During user input, it is also taken into consideration whether the mouse pointer is dragged within the element area or not while the mouse button is pressed. `TRUE`: While the mouse click event exists and the mouse pointer is moved over the element area. `FALSE`: A mouse click event does not exist. Or the user moves the mouse pointer outside of the element area while the mouse button is pressed. The value is `TRUE` again as soon as the user moves the pointer back to the element area. The mouse is then captured.

"Toggle"	With the onset of a mouse click event, the variable is set; when the mouse click event is completed, the variable is reset.
"Variable"	Variable (`BOOL`). Its value toggled when the mouse click event is ended. This is when the user releases the mouse button while the mouse pointer is over the element area. If the user releases the mouse button while the mouse pointer is outside of the element area, then the mouse click event is not ended and the value is not toggled. Hint: The user can cancel a started toggle input by dragging the mouse pointer out of the element area.
"Toggle on up if captured"	☑: The value toggles regardless of where the mouse pointer is when the mouse button is released. The mouse is then captured.

"Hotkey"	Keyboard shortcut on the element for triggering specific input actions. When the keyboard shortcut event occurs, the input actions in the *"Events"* property are triggered. In this way, it is not the input action itself that leads to this input action, but the mouse input action.
"Key"	Key pressed for input action. Example: *[T]* Note: The following properties appear when a key is selected.
"Events"	● *"None"* ● *"Mouse down"*: Pressing the key triggers the input actions that are configured in the *"OnMouseDown"* property. ● *"Mouse up"*: Releasing the key triggers the input actions that are configured in the *"OnMouseUp"* property. ● *"Mouse down/up"*: Pressing and releasing the key triggers the input actions that are configured in the *"OnMouseDown"* property and the *"OnMouseUp"* property.
"Shift"	☑: Combination with the Shift key Example: *[Shift]+[T]*.
"Control"	☑: Combination with the Ctrl key Example: *[Ctrl]+[T]*.
"Alt"	☑: Combination with the Alt key Example: *[Alt]+[T]*.

All keyboard shortcuts and their actions that are configured in the visualization are listed on the "Keyboard Configuration" tab.

See also

- *Chapter 1.4.5.19.2.2 "Command 'Keyboard Configuration'" on page 1720*
- *Chapter 1.4.5.19.3.6 "Dialog 'Input Configuration'" on page 1749*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Date Range Picker'

Symbol:

Category: *"Date/Time Controls"*

The element provides the capability of selecting the date and time range of a saved data set. The element is used with the *"Trend"* visualization element.

Element properties

"Element name"	Example: `DateTrend1` Optional Hint: Assign individual names for elements so that they are found faster in the element list.
"Type of element"	*"Date Range Picker"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

“Show frame”	☑: The visualization element is drawn with a frame.
“Resolution”	Resolution saved for the time stamp: *“Millisecond”* or *“Microsecond”*
“Attached element instance”	The element can be assigned to a *“Trend”* visualization element. As a result, the time range of the trend element can be changed. The available visual elements are selected with the help of the Input Assistant (...).

Element property 'Tick mark labels'

“Two-line labelling”	☑: The time stamps are displayed in two lines. The date is displayed in the first line and the time is displayed in the second line. ☐: Time stamp is displayed in one line. The date and time can also be displayed in one line depending on the formatting.
“ Omit irrelevant information in time stamp”	☑: The time stamp has a shorter form. For example, the date is displayed only for the first tick mark, and only the time for the following tick marks. The settings in *“Internationalization (format strings)”* are ignored for this setting. ☐: All information is displayed for all time stamps.
“Internationalization (format strings)”	Only active when the parameter *“Omit irrelevant information in timestamps”* is deactivated.
“Date”	Definition of the date format. The default setting is taken from the Windows control panel.
“Time”	Definition of the time format. The default setting is taken from the Windows control panel.

Element property 'Text properties' The properties contain fixed values for the text properties.

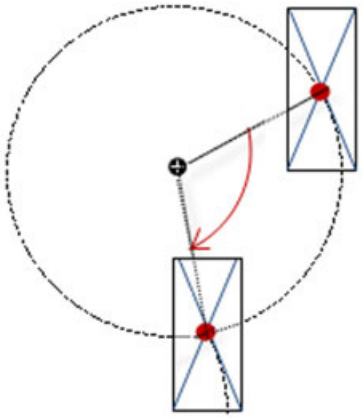
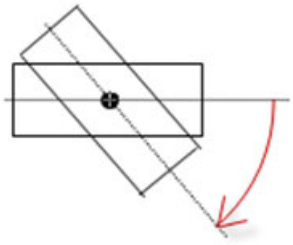
“Horizontal alignment”	Horizontal alignment of the text within the element.
“Vertical alignment”	Vertical alignment of the text within the element.
“Font”	Example: *“Default”* ...: The *“Font”* dialog box opens. ▼: Drop-down list with style fonts.
“Font color”	Example: *“Black”* ...: The *“Color”* dialog box opens. ▼: Drop-down list with style colors.
“Transparency”	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Element property 'Additional buttons'

"Jump to the largest possible time stamp"	☑: An additional button ( ) is displayed for jumping to the last time stamp.
"Jump to the smallest possible time stamp"	☑: An additional button ( ) is displayed for jumping to the first time stamp.
"Zoom out"	☑: An additional button ( ) is displayed for setting the current min./max. range to the maximum range. The selected range is left.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *"Position ➔ Angle"* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

● *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

Visualization Element 'Time Range Picker'

Symbol:

Category: *"Date/Time Controls"*

The element provides configurable buttons for setting the time range of a trend display to a defined time. In the process the end time of the previous display is left unchanged and the start time is adapted.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `TimeRangeTemperature`
"Type of element"	*"Time range picker"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

> *You can also change the values by dragging the box symbols ( ) to other positions in the editor.*

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

"Orientation"	Specifies whether the time picker element is aligned horizontally or vertically in the editor. Hint: Change the width to height ratio of the element in the editor.
"Show frame"	: The visualization element is drawn with a frame.
"Resolution"	Resolution saved for the time stamp: *"Millisecond"* or *"Microsecond"*
"Attached element instance"	Assignment to the element that processes the time picker The element can be assigned for example to a *"Trend"* visualization element. Then the time range of the trend element can be changed. The available visual elements are selected with the help of the input assistance ( ). Example: `GenElemInst_1`

Element property 'Texts'

"Text"	String label for the element. Example: `Zoom`

Element property 'Text properties' The properties contain fixed values for the text properties.

"Font"	Example: *"Default"* : The *"Font"* dialog box opens. : Drop-down list with style fonts.
"Font color"	Example: *"Black"* : The *"Color"* dialog box opens. : Drop-down list with style colors.
"Transparency"	Whole number (value range from `0` to `255`). This determines the transparency of the respective color. Example: `255`: The color is opaque. `0`: The color is completely transparent. Please note: If the color is a style color and already has a transparency value, then this property is write-protected.

Property 'Times' In *"Times"*, the buttons that the element provides at runtime are defined and configured in an array.

"Provide "All" selection"	:Time Range Picker bar extended by "All" button. The diagram represents a time interval that covers all time stamps.
"Times"	: Adds another button to the Time Range Picker bar and increases the array by one entry. An additional index is present in the property *"Times* ➔ *Times* ➔ *Times* ➔ *[<new>]"*. *"Time"* is located under this index. The configuration of the button is to be entered there.
"Times" ● *" [Index]"* with index ∈ {0, 1, 2,...}	Array of all buttons in the time selection bar. Index corresponds to the number of buttons. : The associated button is removed from the Time Range Picker bar. The configuration entry is deleted from the *"Times"* property list.
" [Index]" ● *"Time"*	: Time interval in standardized notation. Example: `3M` for 3 months; `30m` for 30 minutes. If a time interval is indicated in the field, then the button is labelled with it. If a user clicks on the button at runtime, the command is executed to switch the diagram to this time interval. The default is empty.

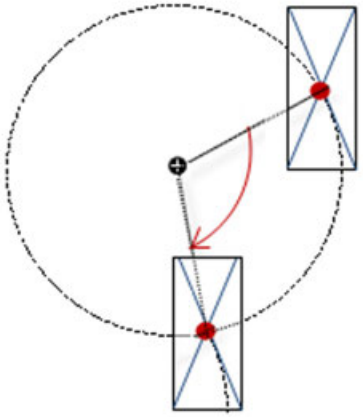
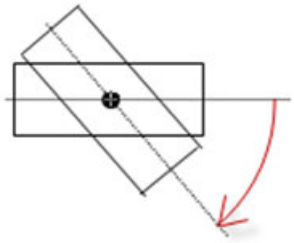
Element property 'Control variables'

"Time"	Displays which time is currently selected. Variable (`STRING`) Example: `PLC_PRG.strSelcetedTime`
""All" selected"	Displays the state of the "All" button Variable (`BOOL`) Example: `PLC_PRG.AllTimesAreSelected`

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

Visualization Element 'Date Picker'

Symbol:

Category: *“Date/Time Controls”*

The element is a calendar that displays the current date. A user can click a day to select a date, which is saved to a variable. In addition, it can customize the time interval that the calendar displays. Clicking the calendar header changes the year. Clicking the arrows in the calendar header changes the month.

Language-dependent texts of the element

The element contains language-dependent texts that are managed in the `System` text list. This deals with the names of the month and the days of the week written out completely or abbreviated. When the date picker is added to a visualization, CODESYS generates the text list automatically below the POU view. The IDs correspond to the standard text and therefore English terms. The text list makes it possible to translate these texts.

Example

`System` text list

ID	Default
`Apr`	`Apr`
`April`	`April`

See also

- *Chapter 1.4.5.6 “Setting Up Multiple Languages” on page 1286*

Element properties

“Element name”	Example: `DueDateCalendar`
“Type of element”	*“Date Picker”*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

“X”	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Y”	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
“Width”	Specified in pixels. Example: `150`
“Height”	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 “Positioning the Element, Adapting Size and Layer” on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

“X”	X-coordinate of the point of rotation
“Y”	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

“Variable”	Input variable (`DATE`). Contains the date that a user selects in the calendar. Example: `PLC_PRG.dtDueDate`
“Design”	● *“From style”*: All settings are preconfigured according to the style. ● *“Explicit”*: The *“Design settings”* property is available. You can customize the calendar here.

Design settings Requirement: This property is visible only if the *“Design”* property is set to *“Explicit”*.

The values of the property can be predefined in the style. Then they are available in the drop-down list.

Table 366: “Header of Date Picker”

Design of the header	
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Arrows”	
“Arrow color”	Style color or user-defined color
“Color of printed arrow”	
“Background”	
“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Fill color”	Style color or user-defined color

Table 367: Design of the main display area

Design of the main display area	
“Today”	Design of today
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The style defines whether and which background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	Style color or user-defined color. Used if *“Yes”* is selected in *“Draw background”*.

“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Selected day”	Design of the selected day
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	Style color or user-defined color
“Show frame”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Current month”	Design of the current month
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Other months”	Design of the previous and subsequent months
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Display other month”	Design of the previous and subsequent months

“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	

“Day of week heading”	Design of the heading with the days of the week
“Font”	Style font or user-defined font
“Font color”	Style color or user-defined color
“Draw background”	*“From style”*: The background is filled with the style color *“From style”*. The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Background color”* property. *“No”*: The background is not filled with a color.
“Background color”	
“Show frame”	*“From style”*: The style defines whether and how a frame is drawn. *“Yes”*: The frame is displayed with the following properties. *“No”*: A frame is not displayed.
“Frame color”	Used if *“Yes”* is selected in *“Show frame”*.
“Rectangle type”	
“Line width”	
“Display separator line”	*“From style”*: The style defines whether and how a separator line is drawn. *“Yes”*: Display with the following properties. *“No”*: A separator line is not displayed.
“Color of the separator line”	Used if *“Yes”* is selected in *“Display separator line”*.
“Width of separator line”	

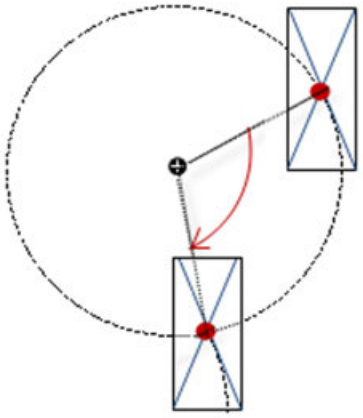
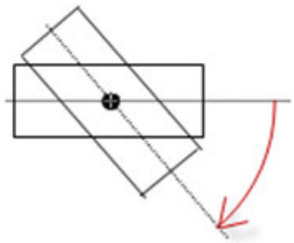
“Background”	Design of the calendar days
“Draw background”	*“From style”*: The style defines whether and how a background is drawn. *“Yes”*: The background is filled with the color in the *“Fill color”* property and framed in the *“Frame color”*. *“No”*: The background is not filled with a color.
“Fill color”	Style color or user-defined color
“Frame color”	

Element property 'Display type'

“Rows”	Number of month calendars per row (preset: `1`)
“Columns”	Number of month calendars per column (preset: `1`)

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

“Movement”		
“X”	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
“Y”	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- ↳ *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

"Access rights"	Opens the *"Access rights"* dialog. There you can edit the access privileges for the element. Status messages: ● *"Not set. Full rights."*: Access rights for all user groups : *"operable"* ● *"Rights are set: Limited rights"*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 "Dialog 'Access Rights'" on page 1745*

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Analog Clock'

Symbol:

Category: *"Date/Time Controls"*

The element is a clock that displays the current time of day. The clock can also display a random time.

Element properties

"Element name"	Example: `GenElemInst_1`
"Type of element"	*"Analog Clock"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

Element property 'Time Display'

"Use system time"	☑: The system time of the PLC is displayed.
"Variable"	Variable (time data type `TOD`, `TIME_OF_DAY`). This receives the time of day that is not the system time. Example: `PLC_PRG.todTimeTokio` Requirement: The *"Use system time"* property is not activated.

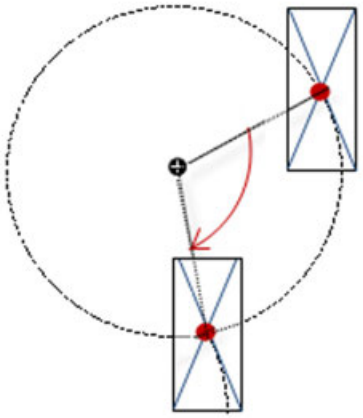
See also

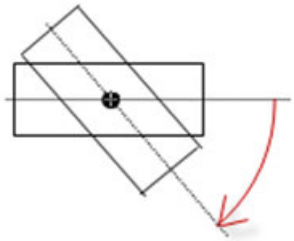
- *Chapter 1.4.1.19.5.5 "Data Type 'TIME'" on page 649*

"Design"	• *"From style"*: All settings are preconfigured according to the style. • *"Explicit"*: The *"Settings"* property is available. Here you can customize the analog clock.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
"Scaling"	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *"Center"* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	

"Interior rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *"Center"* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *"Position ➔ Angle"*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	
"Use REAL values"	Note: Only available if the device supports the use of REAL coordinates. ☑: The properties of the absolute movement are interpreted as REAL values. The values are not rounded. The option allows for the individual fine-tuning of drawing the element, for example for the visualization of a smoother rotation. Hint: If a horizontal or vertical line is drawn blurry on a specific visualization platform, then this can be corrected by an offset of 0.5px in the direction of the line thickness.	

You can link the variables to a unit conversion.

The properties "X", "Y", "Rotation", and "Interior rotation" are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 "Unit conversion" on page 298*

Element property 'Settings'

Requirement: The *"Property"* is *"Explicit"*. Only then is the *"Clock Settings"* category visible.

Table 368: "Background"

"Background color"	Color variants of the default background image ● *"Yellow"* ● *"Red"* ● *"Blue"* ● *"Green"* ● *"Black"*
"Own background"	Background display with the specific *"Image"*. Replaces the default background image.

“Image”	Image from an image pool or library Example: `myImagepool.myImage`
“Transparency color”	The transparent color in the image representation. Example: *“White”*. The white parts of the image are transparent.
“Use background color”	☑: The image background is displayed using the color defined in the *“Background color”* property. Requirement: No image reference is given in the *“Image”* property.
“Background color”	Style color or color Requirement: *“Use background color”* is activated.

Table 369: “Hands”

“Hand style”	Example: *“Thin arrow”*
“Color hour hand”	Style color or color for the hands
“Color minute hand”	
“Color second hand”	

Table 370: “Lines”

“Lines style”	Clock face graduation ● *“None”* ● *“Line”*: Graduation lines by hour ● *“Hours and minutes”*: Graduation lines by hours and minutes ● *“Dots”*: Graduation dots by hour
“Color”	Color of the clock face graduation
“Line width”	Line weight of the clock face graduation
“Scale in 3D”	☑: Representation of the clock face with 3D effect

Table 371: “Numerics”

“Style of numerics”	Digits on the clock face ● *“None”* ● *“Quarter”* ● *“All”*
“Font”	Font for displaying the digits
“Font color”	Font for displaying the digits

Table 372: “Center point”

“Color”	Color of the center of the clock

Table 373: “Positioning”

“Usage of”	● *“Default style values”*: Presetting of the style values ● *“Individual settings”*: User-defined settings in the subordinate *“Positioning”* property.
“Positioning”	Requirement: Visible when the *“Usage or”* property is set to *“Individual settings”*.

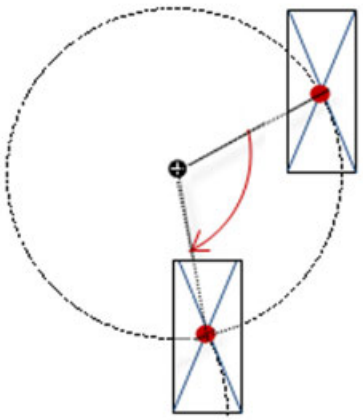
"Numerics movement"	Value (in pixels) for shifting the digits. Example: `80`
"Line movement"	Value (in pixels) for shifting the hour lines. Example: `100`
"Hands scaling"	Factor for scaling the length of the hour hand. You can customize the exact position of the hour hand relative to the background image. Example: `100`

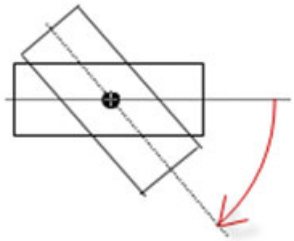
"Scaling type"	Defines the scaling of the height and width of the element. ● *"Anisotropic"*: The background image is scaled to the size of the element The height and width are scaled independently of each other. ● *"Isotropic"*: The background image is scaled to the size of the element, retaining its proportion. The proportion of height and width is fixed.

"Optimized drawing"	☑: The background image is drawn one time. When the hour hand moves, only the affected part of the image is redrawn. ☐: The background image is redrawn in cycles. Hint: Disable this option only for extreme exceptions.

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"		
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.	
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.	
"Rotation"	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *"Center"* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	

“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the *“Position ➔ Angle”* property, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The “X”, “Y”, “Rotation”, and “Interior rotation” properties are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

"Animation duration"	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *"Absolute movement"*, *"Movement"*, *"X"*, *"Y"* ● *"Absolute movement"*, *"Rotation"* ● *"Absolute movement"*, *"Interior rotation"* ● *"Absolute movement"*, *"Exterior rotation"* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
"Move to foreground"	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

See also

- *Chapter 1.4.5.3 "Designing a visualization with elements" on page 1254*

Visualization Element 'Date/Time Picker'

Symbol:

Category: *"Date/Time Controls"*

The element provides the capability of selecting the date and time. The value can be changed by means of the arrow keys on the keyboard. The date can be selected from a calendar.

Element properties

"Element name"	Optional Hint: Assign individual names for elements so that they are found faster in the element list. Example: `StartDateAndTime`
"Type of element"	*"Date/Time Picker"*

Element property 'Position'

The position defines the location and size of the element in the visualization window. These are based on the Cartesian coordinate system. The origin is located at the upper left corner of the window. The positive horizontal x-axis runs to the right. The positive vertical y-axis runs downwards.

"X"	X coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Y"	Y coordinate of the upper left corner of the element Specified in pixels. Example: `10`.
"Width"	Specified in pixels. Example: `150`
"Height"	Specified in pixels. Example: `30`

You can also change the values by dragging the box symbols ( ) to other positions in the editor.

See also

- *Chapter 1.4.5.3.2 "Positioning the Element, Adapting Size and Layer" on page 1256*

Element property 'Center'

The properties contain fixed values for the coordinates of the point of rotation. This point of rotation is shown as the ⊕ symbol. The point is used as the center for rotating and scaling.

"X"	X-coordinate of the point of rotation
"Y"	Y-coordinate of the point of rotation

You can also change the values by dragging the symbols (⊕) to other positions in the editor.

"Variable"	Variable (`DATE, DT, TIME, LTIME, TOD`) The value of the value of the variable is displayed and modified by means of the element. The data type automatically determines the displayed value units: ● `TIME`: Day, hour, minute, and second (by default, milliseconds are not displayed) ● `DATE`: Year, month, and day ● `DT`: Year, month, day, hour, minute, and second ● `TOD`: Hour, minute, and second (by default, milliseconds are not displayed) ● `LTIME`: Day, hour, minute, and second (by default, milliseconds, microseconds, and nanoseconds are not displayed)
"Format string"	The format can restrict the output to individual values. Example for `LTIME`: Format: `HH:mm:ss.ms.us.ns` --> displayed: 08:15:12.780.150.360 `LTIME` restricted: format: `HH:mm` --> displayed: 08:15 Example for `DATE`: Format: `yyyy/MM/dd` --> displayed: 2015/12/17 . Basically, all usual formats available for %t are also supported.
"Design date time picker"	● *"From style"*: All settings are preconfigured according to the style. ● *"Explicit"*: The *"Design settings"* property is available. You can customize the calendar here.
"Design date picker"	● *"From style"*: All settings are preconfigured according to the style. ● *"Explicit"*: The *"Design settings"* property is available. You can customize the calendar here.
"Positioning date picker"	● *"Dynamic"*: The calendar is adapted and positioned automatically. ● *"Manual"*: The *"Position settings"* property is available. You can customize the calendar here.

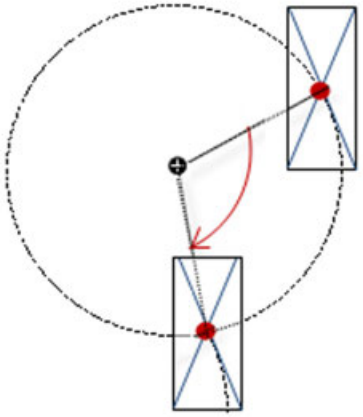
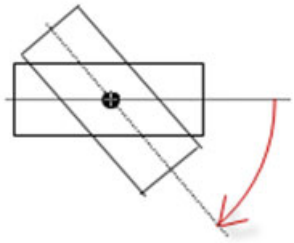
See also

- *Chapter 1.4.5.18.2 "Placeholders with Format Definition in the Output Text" on page 1708*

Element property 'Absolute movement'

The properties contain IEC variables for controlling the position of the element dynamically. The reference point is the upper left corner of the element. In runtime mode, the entire element is moved.

"Movement"	
"X"	Variable (numeric data type). Defines the X position (in pixels). Example: `PLC_PRG.iPos_X`. Increasing this value in runtime mode moves the element to the right.
"Y"	Variable (numeric data type). Defines the Y position (in pixels). Example: `PLC_PRG.iPos_Y`. Increasing this value in runtime mode moves the element downwards.

“Rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle1`. The midpoint of the element rotates at the *“Center”* point. This rotation point is shown as the ⊕ symbol. In runtime mode, the alignment of the element remains the same with respect to the coordinate system of the visualization. Increasing the value rotates the element to the right.	
“Scaling”	Variable (integer data type). Causes centric stretching. Example: `PLC_PRG.iScaling`. The reference point is the *“Center”* property. The value `1` shrinks the element by a factor of `0.001`. The value `1000` returns the element to its original size.	
“Interior rotation”	Variable (numeric data type). Defines the angle of rotation (in degrees). Example: `PLC_PRG.iAngle2`. In runtime mode, the element rotates about the point of rotation specified in *“Center”* according to the value of the variable. In addition, the alignment of the element rotates according to the coordinate system of the visualization. Increasing the value in the code rotates clockwise. The rotation point is shown as the ⊕ symbol. Note: If a static angle of rotation is specified in the property *“Position ➔ Angle”*, then the static angle of rotation is added to the variable angle of rotation (offset) when the visualization is executed.	

You can link the variables to a unit conversion.

The properties “X”, “Y”, “Rotation”, and “Interior rotation” are supported by the "Client Animation" functionality.

See also

- *Chapter 1.4.1.8.18 “Unit conversion” on page 298*

Element property 'State variables'

The variables control the element behavior dynamically.

“Invisible”	Variable (`BOOL`). Toggles the visibility of the element. `TRUE`: The element is not visible at runtime. Example: `bIsVisible` with `VAR bIsVisible : BOOL := FALSE; END_VAR`
“Deactivate inputs”	Variable (`BOOL`). Toggles the operability of the element. `TRUE`: User inputs do not have any effect in runtime more. The element is shown as deactivated.

The “Invisible” property is supported by the "Client Animation" functionality.

These properties are available only when you have selected the *“Support client animations and overlay of native elements”* option in the Visualization Manager.

“Animation duration”	Defines the duration (in milliseconds) in which the element runs an animation ● Variable (`integer value`) Example: `Menu.tContent` with `VAR tContent : INT := 500; END_VAR` ● Integer literal Example: `500` Animatable properties ● *“Absolute movement”*, *“Movement”*, *“X”*, *“Y”* ● *“Absolute movement”*, *“Rotation”* ● *“Absolute movement”*, *“Interior rotation”* ● *“Absolute movement”*, *“Exterior rotation”* The animated movement is executed when at least one value of an animatable property has changed. The movement then executed is not jerky, but is smooth within the specified animation duration. The visualization element travels to the specified position while rotating dynamically. The transitions are smooth.
“Move to foreground”	Moves the visualization element to the foreground Variable (`BOOL`) Example: `bIsInForeground` with `VAR bIsInForeground : BOOL := FALSE; END_VAR` `TRUE`: At runtime, the visualization element is displayed in the foreground. `FALSE`: At runtime, the visualization element is displayed in the layer where it was inserted in the visualization editor.

Element property 'Access rights'

Requirement: User management is set up for the visualization.

“Access rights”	Opens the *“Access rights”* dialog. There you can edit the access privileges for the element. Status messages: ● *“Not set. Full rights.”*: Access rights for all user groups : *“operable”* ● *“Rights are set: Limited rights”*: Access is restricted for at least one group.

See also

- *Chapter 1.4.5.19.3.1 “Dialog 'Access Rights'” on page 1745*

1.4.5.20 Reference, visualization style editor

1.4.5.20.1 Dialog 'Create a New Visualization Style'

Symbol: 

Function: The dialog prompts you to specify data for a new created style file.

Call:

- In CODESYS:
  In the *“Visualization Manager”* object (tab *“Settings”*, group *“Style Settings”*):
  Clicking opens a drop-down list. Click *“Create and Edit Derived Style”*.
- In the visualization style editor:
  Menu bar: *“File ➔ New Style”*

“Name”	Name of the new style. Example: `Style_CI`
“Storage location”	Working directory for style editing
“Base style”	Style to base the new style on The drop-down list includes all styles that are installed in the repository. *“<none>”*: The new style does not derive itself from an existing style.
“Visualization profile”	The profile is intended for informational purposes. For example, you find elements that are not preconfigured with special style entries, and information from the profile. In addition, CODESYS checks in the profile whether a required style is missing. Example: `CODESYS V3.5 SP9`
Click *“OK”*	The new style is created and opened for editing in the visualization style editor. It already includes all required style entries and the localization in German (language column `de`).

See also

- *Chapter 1.4.5.20.3 “Editor 'Visualization Style Editor'” on page 2128*
- *Chapter 1.4.5.19.4.2 “Object 'Visualization manager'” on page 1777*
- *Chapter 1.4.5.17 “Applying Visualization Styles” on page 1360*

1.4.5.20.2 Dialog 'Open Existing Style as a Copy'

Function: This dialog prompts you to specify data for copying a style file.

Call:

- In CODESYS:
  In the editor of the *“Visualization Manager”* object (tab *“Settings”*, group *“Style Settings”*, click for a drop-down list). Click *“Copy and Edit Style”*.
- In the visualization style editor:
  Menu bar: *“File ➔ Open as Copy”*

"Style"	Style to be copied. Example: `Basic Style, 3.5.9.0` Note: You can also select a style from the repository.
"Destination"	Working directory for style editing
"OK"	A copy of the style is created and opened for editing in the visualization style editor.

See also

- *Chapter 1.4.5.17 "Applying Visualization Styles" on page 1360*
- *Chapter 1.4.5.20.3 "Editor 'Visualization Style Editor'" on page 2128*
- *Chapter 1.4.5.19.4.2 "Object 'Visualization manager'" on page 1777*

1.4.5.20.3 Editor 'Visualization Style Editor'

Symbol:

Function: The editor is used for creating, deriving, editing, and localizing visualization styles. In addition, it makes it possible to check and install a style or a hierarchy of styles.

Call:

- In CODESYS:
  In the *"Visualization Manager"* object (tab *"Settings"*, group *"Style Settings"*, click for a drop-down list). Click *"Open Style Editor"*.
- Start menu > CODESYS installation folder > 'CODESYS' > 'Visualization Style Editor'

Menu 'File'

"New style"	The *"Create a New Visualization Style"* dialog box opens.
"Open"	The *"Open Dialog"* dialog box opens. This dialog prompts you to select a style file (format `.visustyle.xml`) to be opened and edited.
"Open as copy"	The *"Open Existing Style as Dialog"* dialog box opens. This dialog prompts you to select a style that is copied, saved to the target location, and opened for editing.
"Close"	Closes the style open in the editor.
"Save"	Saves the changes of the open style.
"Save As"	The *"Select Visualization Style(s)"* dialog box opens. This dialog prompts you to select a file to save the current settings.
"Save and Install"	Saves the open visualization style and installs it to the visualization style repository.
"Recently opened files"	Lists the files for selection that were last opened.
"Abort"	Closes the visualization style editor.

See also

- *Chapter 1.4.5.20.1 "Dialog 'Create a New Visualization Style'" on page 2127*

Menu 'Styles' The commands affect the contents of the *"Style Properties"* tab.

"New Entry (as Child) "	Creates another style entry as a child of the selected style property.
"New Entry (Afterwards)"	Creates a new style entry in the list after the selected style property.
"Move Down"	Moves the selected style entry down. Requirement: Sort order is flat.
"Move Up"	Moves the selected style entry up. Requirement: Sort order is flat.
"Sort Order"	Toggles between three sort orders: ● Flat structure and alphabetical order ● Flat structure and order according to the position of the entry in the XML style file This position also determines the position of the property in CODESYS. The property appears, for example, in the *"Properties"* view below the *"Values"* column in the drop-down list for style properties. ● Hierarchical structure of entries Requirement: The names of the style properties contain at least one dash.
"Check"	The settings of the style properties are checked for consistency errors. This check is also performed when saving the style.

Menu 'Localization' The commands affect the contents of the *"Localization"* tab.

"Add Language"	The dialog box *"Add New Language"* opens. The dialog prompts you to specify data for creating a new language column.
"Remove Selected Language"	Removes the columns of the selected cell.
"Rename Selected Language"	The *"Rename Language"* dialog box opens. The dialog is used for renaming the column that defines the selected cell and removes all previous translations.

Table 374: Dialog box "Add New Language"

"Name"	Name of the new language as a language code according to ISO 639-1. Examples: `de`, `en`, `es`, `it`, `fr`, `ja`
"Copy from existing"	All existing language columns are available for selection. The selected language is copied with all entered translations. *"<do not copy text>"*: The new language receives a blank translation column.

Tab 'General' This tab contains the general metadata of the open style file and allows it to be edited.

Table 375: "Identification"

"Company"	Example: `Xy-z GmbH` Tip: In the installed styles, CODESYS can filter by the company names specified here.
"Name"	Example: `Style_A`
" Version"	User-defined version number Example: `1.1.1.1`

Table 376: “General Settings”

“Base style”	Name and version of the style that the open style is based on. Tip: The derived style properties from the base style are highlighted in yellow in the *“Style Properties”* tab.
“Partial style (usable only as base for other visualization styles)”	☑: The style is identified as incomplete. Therefore, it can be used for other styles as a base style only. Example: Style only with color definitions that derive this to many other styles. Note: CODESYS does not check for consistency errors of an incomplete style for itself. ☐: The style is identified as complete.
“From”	The *“Select Base Style”* dialog box opens. This dialog prompts you to select a style file that is saved to the file system (and not does not have to be installed). The file is used as a base style.

Table 377: “Informational”

“Visualization profile”	The profile is intended for informational purposes. For example, you find elements that are not preconfigured with special style entries, and information from the profile. In addition, CODESYS checks in the profile whether a required style is missing.

Tab 'Style Properties'

This tab lists the names of the style properties with the associated values and makes it possible to edit it, even by means of the commands in the *“Styles”* menu.

The style properties can be defined for colors, fonts, images, and any values.

The style properties defined in a base style are derived and highlighted in yellow.

“Name”	Name of the style property. If the name contains a dash, then the Visualization Style Editor can sort the style properties by the prefixed terms before the dash and display them in a hierarchy. A name can contain more than one dash.
“Value”	Value that is assigned to the style property.
“Type”	Data type of the style property; selected from a drop-down list. Note: This is possible and necessary only for specific style properties with a data type that is not implicitly defined.
“Attribute”	*“hide”*: The associated style property is not listed in the drop-down lists in CODESYS.
“Used by”	Visualization element that can be configured with this style property. Can be edited.
Comment	Example: `Special setting for Bar Display`. Optional.
Double-click a cell.	An input field opens for editing.
[Del]	Removes the selected row.

Tab 'Localization'

This tab makes it possible to translate the names of the style properties into other languages.

“Name”	Lists the name of the style properties as they are defined in the *“Style Properties”* tab.
“<language>”	Identification of the language name (as language code according to ISO 639-1) in which the style property name should be translated.
Double-click a cell.	An input field opens for editing.

1.4.5.21 Tutorial

Here you find instructions specific to different use cases.

This collection of instructions is expanded regularly.

1.4.5.21.1 CODESYS visualization - first steps

If you are not yet familiar with the CODESYS visualization, we recommend the *application example*. The example demonstrates the main features of visualization and provides insights into possible use cases.

1.4.5.21.2 Show instance names

For complex visualizations, it can be helpful if the instance names are displayed within the visualization. How to show instance names is described in the *application example*.

1.4.5.21.3 Visualizing a Refrigerator Controller

This tutorial demonstrates how to add visualizations to the project and link the elements of the visualization to the variables of the control program.

Preparation

This tutorial is based on the sample program `RefigeratorControl`, which was created in the "Your First Program in CODESYS" chapter. The finished program can also be found in the installation directory of CODESYS, in the "Projects" subfolder.

See also

- *Your First CODESYS Program*

Creating the visualizations

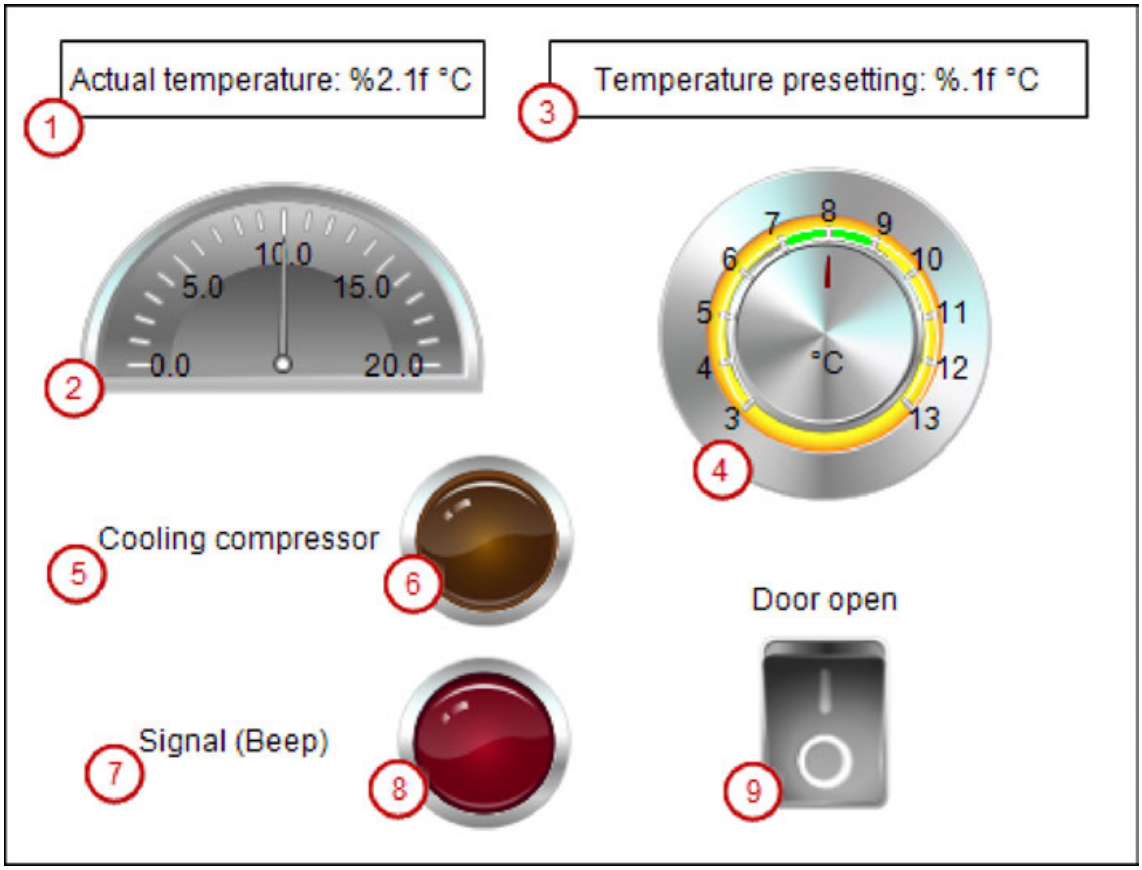
The visualization consists of the following three visualization screens:

- `Visualization`: Control elements and display of the refrigerator
- `Diagnosis`: History of the set and actual temperature, parameter settings
- `Live Visu`: Animation with refrigerator

1. Select the *“Application”* object in the device tree.
2. Click *“Project ➔ Add Object ➔ Visualization”*.
3. Specify `Live_Visu` as the name.
4. Create two more visualizations with the names `Diagnosis` and `Visualization`.

Structure of the visualization
`Visualization`

This screen consists of control and display elements that control the refrigerator.



- (1) Numeric display of the actual temperature
- (2) Pointer to display of the actual temperature
- (3) Numeric display of the set temperature
- (4) Potentiometer for setting the set temperature
- (5) Label for compressor lamp
- (6) Lamp for compressor on
- (7) Label for signal lamp
- (8) Lamp for signal "Close doors"
- (9) Switch for opening and closing the refrigerator door

1. Open the visualization `Visualization` in the editor.
2. Drag a *"Rectangle"* visualization element to the editor.

   Change the following properties

   - *"Texts ➔ Text"*: `Actual temperature: %2.1f °C`
   - *"Text variables ➔ Text variable"*: `Glob_Var.rTempActual`
3. Drag a *"Meter 180°"* visualization element to the editor.

   Change the following properties

   - *"Value"*: `Glob_Var.rTempActual`
   - *"Scale ➔ Scale end"*: 20
   - *"Scale ➔ Main scale"*: 5
   - *"Scale ➔ Subscale"*: 1
4. Drag a *"Rectangle"* visualization element to the editor.

   Change the following properties

   - *"Texts ➔ Text"*: `Temperature presetting: %.1f °C`
   - *"Text variables ➔ Text variable"*: `Glob_Var.rTempSet`

5. Drag a *"Potentiometer"* visualization element to the editor.

   Change the following properties

   - *"Variable"*: `Glob_Var.rTempSet`
   - *"Background ➔ Background color"*: *"yellow"*
   - *"Pointer ➔ Color"*: *"red"*
   - *"Scale ➔ Subscale position"*: *"Outward"*
   - *"Scale ➔ Scale start"*: 3
   - *"Scale ➔ Scale end"*: 13
   - *"Scale ➔ Subscale"*: 1
   - *"Scale ➔ Main scale"*: 1
   - *"Label ➔ Unit"*: °C
   - *"Label ➔ Scale format (C syntax)"*: %.0f
   - *"Label ➔ Max. text width of labels"*: 21
   - *"Label ➔ Height of labels"*:15

6. Drag a *"Label"* visualization element to the editor.

   Change the following properties

   - *"Texts ➔ Text"*: `Cooling compressor`

7. Drag a *"Lamp"* visualization element to the editor. Position it behind the text `Cooling compressor`.

   Change the following properties

   - *"Variable"*: `Glob_Var.bCompressor`

8. Drag a *"Label"* visualization element to the editor.

   Change the following properties

   - *"Texts ➔ Text"*: `Signal (beep)`

9. Drag a *"Lamp"* visualization element to the editor. Position it behind the text "Signal (beep)".

   Change the following properties

   - *"Variable"*: `Glob_Var.bSignal`
   - *"Background ➔ Image"*: `Red`

10. Drag a *"Rectangle"* visualization element to the editor.

    Change the following properties

    - *"Texts ➔ Text"*:`Door open`

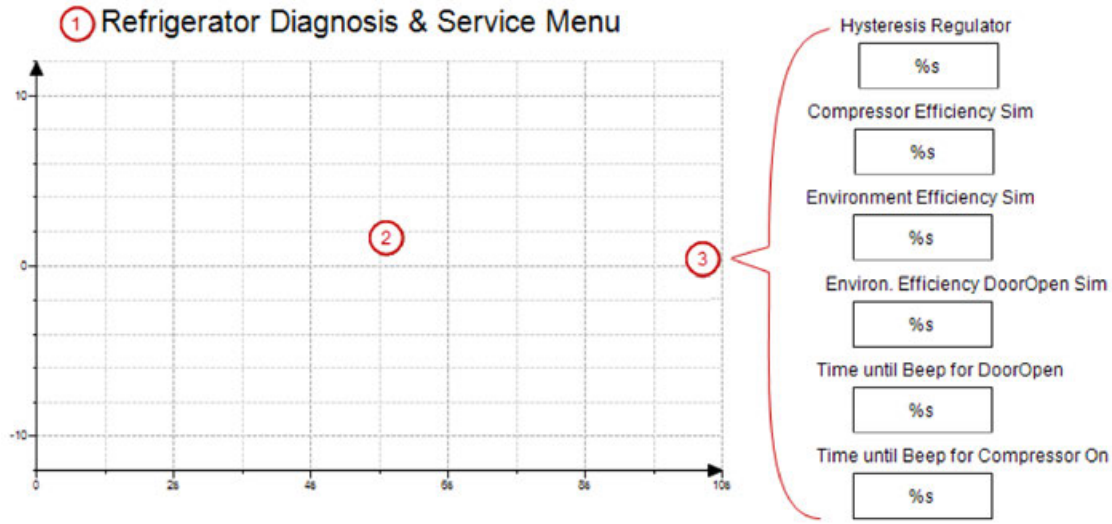
11. Drag a *"Rocker Switch"* visualization element to the editor.

    Change the following properties

    - *"Variable"*: `Glob_Var.rDoorOpen`

Structure of the visualization
`Diagnosis`

In this screen, you can monitor the temperature curve and optimize the parameters.



- (1) *"Label"* elements for the heading
- (2) *"Trace"* element for displaying the temperature curve
- (3) *"Rectangle"* elements for displaying the values

1. Open the visualization `Diagnosis` in the editor.
2. Drag a *"Label"* visualization element to the editor.

   Change the following properties

   - *"Texts ➔ Text"*: Refrigerator Diagnosis & Service Menu
   - *"Text properties ➔ Font"*: Arial, Standard, 18
3. Drag a *"Trace"* visualization element to the editor.
4. Click the `Diagnosis_Trace1` value of the *"Trace"* property.

   ⇨ The *"Trace Configuration"* dialog opens.
5. Select the *"MainTask"* in *"Task"*.
6. Click the *"Add Variable"* link.

   ⇨ A variable is added to the trace. The variable settings are displayed in the dialog.
7. Select `Glob_Var.bCompressor` for the variable.
8. Add the `Glob_Var.rTempSet` and `Glob_Var.rTempActual` variables to the trace. For the other settings, you can use the default values.
9. Click *"OK"* to exit the dialog.
10. Drag a *"Rectangle"* visualization element to the editor. Position it on he right next to the trace element.

    Change the following properties

    - *"Texts ➔ Text"*: %s
    - *"Text variables ➔ Text variable"*: PLC_PRG.rHysteresis
11. Configure the *"OnMouseDown"* input configuration of the element. Click *"Input configuration ➔ OnMouseDown ➔ Configure"*.

    ⇨ The *"Input Configuration"* dialog opens.
12. Assign the *"Write Variable"* command to the action. Accept the default values and click *"OK"*.
13. Drag a *"Label"* visualization element to the editor. Position it over the first rectangle.

    Change the following properties

    - *"Texts ➔ Text"*: Hysteresis Regulator

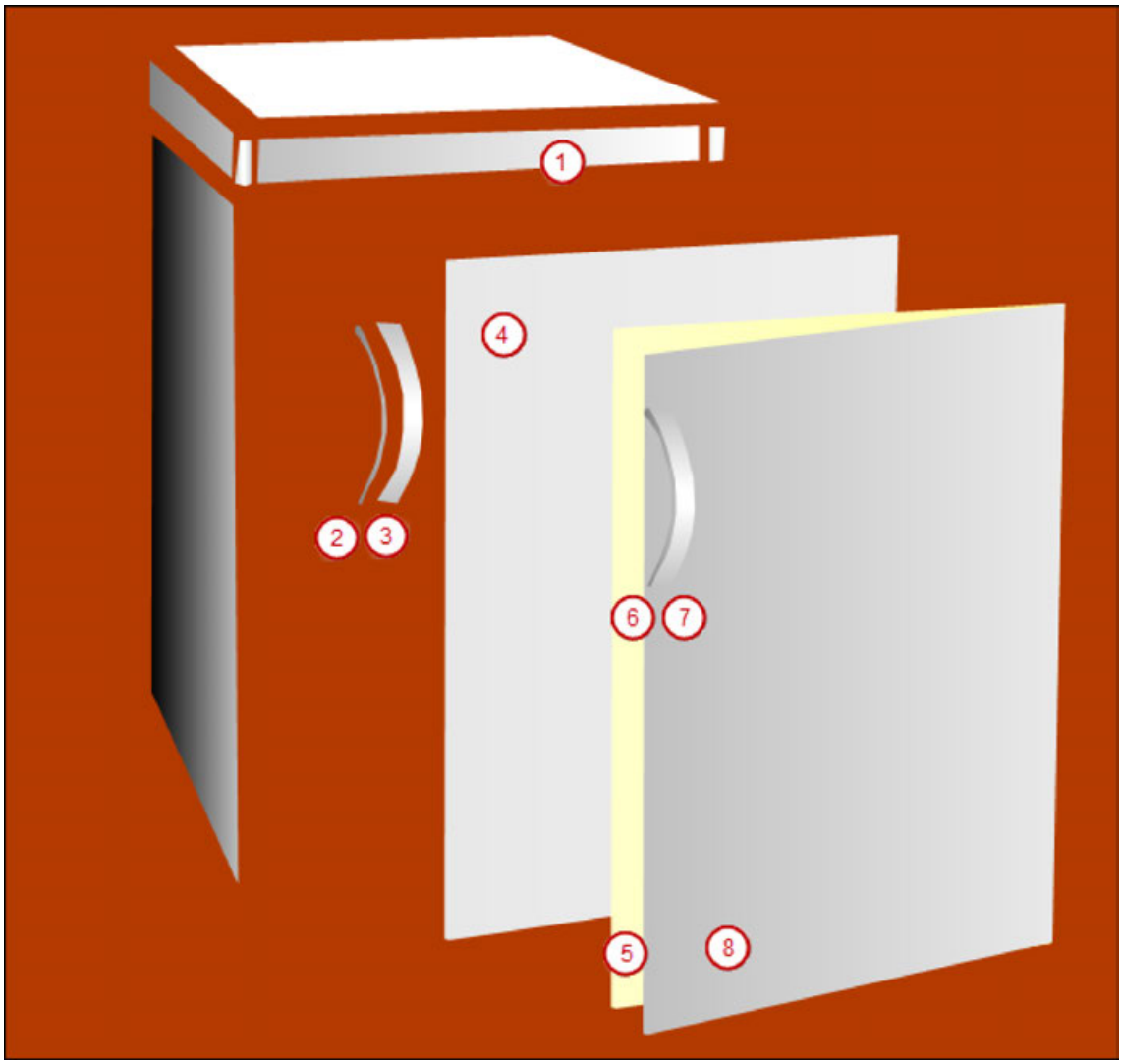
14. Adjust the size and position of both elements.
15. Select both of the *“Rectangle”* and *“Label”* elements and duplicate them by means of copy and paste.
16. Adjust the labels and variables of the copied elements.
    - *“Text”*: `Compressor Efficiency`, *“Text variable”*: `Simulation.P_Cooling`
    - *“Text”*: `Environment Efficiency`, *“Text variable”*: `Simulation.P_Environment`
    - *“Text”*: `Environ. Efficiency DoorOpen Sim`, *“Text variable”*: `Simulation.P_EnvironmentDoorOpen`
    - *“Text”*: `Time until Beep for DoorOpen`, *“Text variable”*: `Glob_Var.timDoorOpenThreshold`
    - *“Text”*: `Time until Beep for Compressor On`, *“Text variable”*: `Glob_Var.timAlarmThreshold`

See also

- *Chapter 1.4.5.19.3.6 “Dialog 'Input Configuration'” on page 1749*

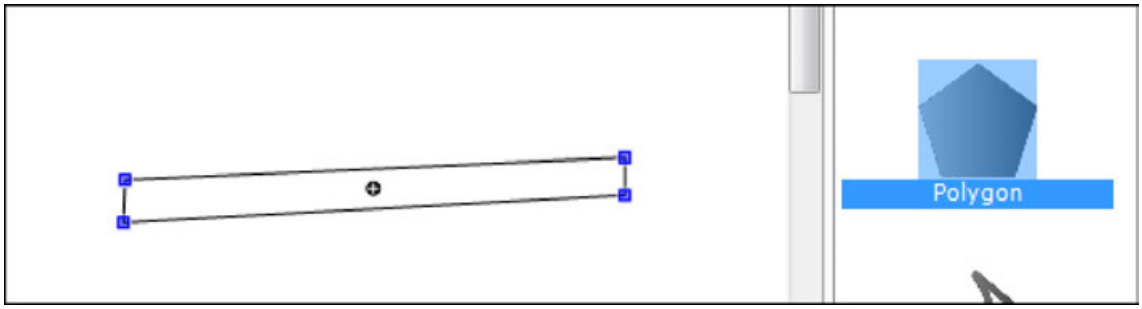
Structure of the visualization 'Live Visu'

This screen includes the representation of a refrigerator. The refrigerator consists of several polygon type visualization elements. The doors of the refrigerator are drawn in both the closed and open states. Both doors consist of a group of single elements.



1. Open the `Live_Visu` visualization in the editor.
2. Select the *“Polygon”* visualization element in the *“Visualization Toolbox”* view.

3. Click several times in the editor to create a surface. Right-click to stop adding corner marks.
4. Move the corner marks to the required position so that the element (1) is formed.



5. Select the element.

   Change the following properties:

   - *"Colors ➔ Use gradient color"*: ☑
   - *"Appearance ➔ Line style"*: *"Invisible"*

6. Click the *"Colors ➔ Use gradient color"* property.
7. Select the color *"Gray"* for *"Color 1"* in the *"Gradient Editor"* dialog.

8. Create all other elements with the *"Polygon"* visualization element.
9. Group the elements of the closed doors (2+3+4) and the open doors (5+6+7+8). To do this, press the *[Shift]* key and click *"Visualization ➔ Group"* to select the elements.
10. Move the elements together so that the completed refrigerator is formed. Position the open doors precisely on the closed doors.
11. Select the "Open doors" group.
12. In the properties, double-click the input field *"State variable ➔ Invisible"*.
13. Press *[F2]* to open the Input Assistant.
14. Select the `rDoorOpen` variable in the *"Variables"* category (below *"Application ➔ Glob_Var"*).
15. Negate the variable with `NOT` (--> NOT Glob_Var.rDoorOpen).

    ⇨ If the `rDoorOpen` variable is FALSE (door is closed), then the element is invisible. Then the underlying doors are visible.

16. Copy the following elements from the `Visualization` screen:

    - Potentiometer for setting the temperature
    - Rectangle for displaying the set temperature
    - `Door open` switch
    - `Cooling compressor` lamp
    - `Signal (beep)` lamp

17. Insert the elements from the clipboard to the `Live_Visu` visualization screen.

18. Reduce the elements and position them on the refrigerator.

Visualization in online mode (simulation)

When the visualization is complete, test it in simulation mode.

1. Click *“Online ➔ Simulation”*.
2. Click *“Online ➔ Login”*.
   ⇨ A dialog opens and prompts you to create and download the application.
3. Click *“Yes”* to confirm the dialog.
4. Click *“Debug ➔ Start”*.
5. Open the visualization `Live_Visu` in the editor.
   ⇨ The refrigerator is in online mode.
6. Open the doors with the switch and monitor the temperature and the alarms. Change the parameters in the screen `Diagnosis` and watch the reaction in the temperature curve.

1.4.5.21.4 Displaying Array Data in a Histogram

Setting element properties for the histogram

Requirements

- A project contains a visualization object and a program.
- A one-dimensional array is declared in the program (example: `histogram : ARRAY[1...10] OF INT;`).
- In the program, `histogram` data is assigned to the array (example: within the range from `0` to `50`).

☑

1. Double-click the *"Visualization"* object in the device tree.
2. In the *"Visualization Toolbox"* view, click *"Measurement Controls"* and drag the *"Histogram"* element to the visualization editor.
3. In the visualization editor, click the inserted *"Histogram"*.

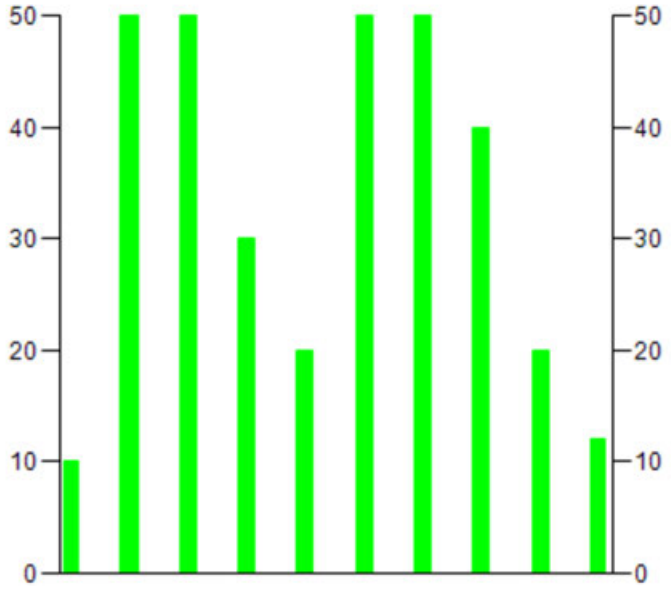
   ⇨ The *"Properties"* view opens.
4. In the *"Properties"* view, double-click the *"Value"* input field in the *"Data array"* element property. Then click ...
5. In the *"Input Assistant"* dialog in the *"Variables"* category of the *"PLC (PRG)"* program, select the array (example here: `histogram : ARRAY[1..10] OF INT;`) and click *"OK"*.
6. To display only part of the array as a histogram, activate the *"Use subrange"* option and specify the index values of the array in *"Start index"* and *"End index"* to define the subrange.
7. Select the *"Display type"* (example: *"Bar"*).
8. Specify a value between `1` and `100` (example: `30`) for the *"Relative bar width"*.
9. Click the histogram in the visualization editor and change the size and position as desired.

   ⇨ The *"Position"* property changes its values accordingly.
10. Specify the values for the *"Scale"* element property. Select the values for *"Scale start"* and *"Scale end"* so that the array is displayed completely. For the example: *"Scale start"* `0`, *"Scale end"* `50`.

    For the distance between values on the main scale, specify the value `10`, for example, in *"Main scale"*.
11. In the *"Label"* element property, specify the *"Unit"* for the display values.
12. Click *"Build ➔ Generate Code"*.

13. If the project has been compiled without errors, then click *“Online ➔ Login”* and click *“Debug ➔ Start”* to start the application.

    ⇨ The histogram is displayed in the visualization as follows:



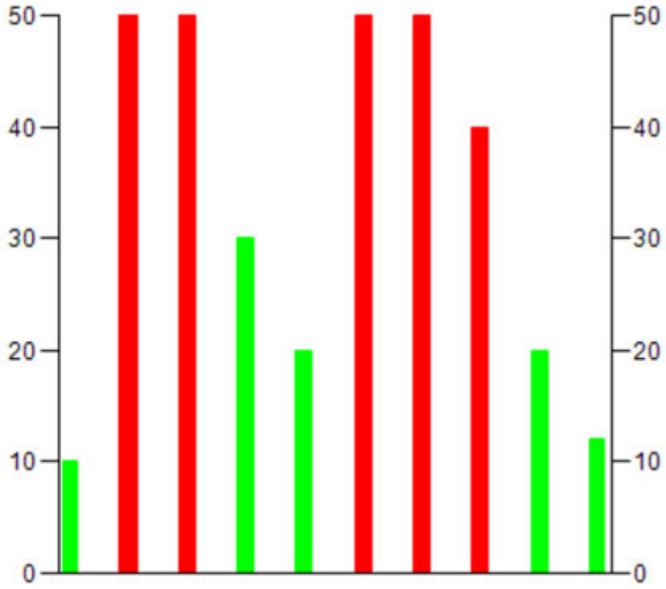
Defining alarm colors for the histogram

☑ The visualization displays a histogram with bars all the same color (example: green). Now you want the bars with values less than `30`, for example, to be displayed in another color (example: red).

1. Click the element property *“Colors ➔ Alarm color”*.
2. Specify the limiting value in *“Alarm value”* above or below which the bars should be displayed in another color.
3. Select *“More”* from the list box in *“Alarm condition”* if all values greater than the *“Alarm value”* should be displayed in another color. Otherwise, select *“Less”*.
4. Select an *“Alarm color”* (example: *“Red”*).
5. Click *“Build ➔ Generate Code”*.

6. If the project has been compiled without errors, then click *“Online ➔ Login”* and click *“Debug ➔ Start”* to start the application.

   ⇨ In the example histogram, all bars with values greater than `30` are displayed in red.



See also

- *Chapter 1.4.5.19.5.29 “Visualization Element 'Histogram'” on page 2019*

1.4.5.21.5 Displaying Array Variables in Tables

A frequently required function of a user interface is the display of data arrays. CODESYS Visualization provides the element *“Table”* for this.

In the configuration of the element *“Table”*, enter an array variable in the property *“Data array”*. The array components are displayed in the rows and columns of the table.

Subsequent instructions describe an example of how an array of a structure is displayed in a table. As a preparation, create the `MYSTRUCT` DUT and the declarations in the `PLC_PRG` program.

```
TYPE MYSTRUCT :
    STRUCT
        iNo : INT;
        bOnStock : BOOL;
        strPartNumber : STRING;
    END_STRUCT
END_TYPE


PROGRAM PLC_PRG
VAR
    arrStruct : ARRAY[0..6] OF MYSTRUCT;
    iSelectedColumn : INT;
END_VAR
```

1. Drag the *"Table"* visualization element to the visualization editor.
2. Assign the array variable `arrStruct` to the *"Data array"* property.
   ⇨ The structure members are displayed as column headings and the array index as row headings.
3. Change the *"Columns ➔ Column ➔ [0] ➔ Column header"* property to an informative heading (example: `Number`).
4. Change the heading of column [1] to `in stock` and column [2] to `Part number`. Adjust the column width.
5. Assign a color to the *"Selection ➔ Selection color"* property.
6. Define the *"Selection ➔ Selection type"* property as `Row selection`.
7. In the *"Selection ➔ Variable for selected row"* property, define the `PLC_PRG.iSelectedColumn` variable.
   ⇨ The following display results in online mode:

	Number	in Stock	Part Number
0	0	FALSE	
1	0	FALSE	
2	0	FALSE	
3	0	FALSE	

See also

- ↳ *Chapter 1.4.5.19.5.13 "Visualization Element 'Table'" on page 1909*

1.4.5.21.6 Displaying Web Contents

Displaying web-sites in a visual-ization

☑ Requirement: A visualization open in a CODESYS project. The *"Visualization Toolbox"* and *"Properties"* views of the visualization are also open.

1. Drag the *"Web Browser"* element from *"Special Controls"* to the visualization editor.
2. Select the element in the editor.
   ⇨ In the *"Properties"* view, the element properties are listed for the *"Web Browser"* element.
3. In the *"Position"* property, specify the size (in pixels) for the *"Width"* and the *"Height"* (example: `600`).
4. In *"Control variables ➔ URL"*, specify the URL for the website (example: `'http://de.wikipedia.org'`). You can also specify a variable here (`STRING` or `WSTRING`) where the URL is assigned in the project.
5. In *"Control variables ➔ Display"*, specify a Boolean variable (example: `bSetURL`).
   ⇨ If the variable `bSetURL` has the value `TRUE`, then the website `'http://en.wikipedia.org'` is displayed at runtime.

Configuring the buttons for forward and back navigation of the website

☑ Requirement: The *"Web Browser"* element of your visualization is configured as described above.

1. In a POU, declare both Boolean variables `bGoBack` and `bGoForward`.
2. In the visualization editor, click the *"Web browser"*.
3. For the property *"Control variables ➔ Back"*, select the variable `bGoBack` from the Input Assistant. For the property *"Control variables ➔ Forward"*, select the variable `bGoForward`.
4. In *"General Controls"*, add the *"Button"* element to your visualization two times.
5. Click a *"Button"* in the visualization editor and drag the *"Button"* to the required position (for example above the *"Web Browser"* element).
6. In the property *"Texts ➔ Text"*, specify the character `>`. In *"Text properties ➔ Font"*, select a font from the Input Assistant (example: `Arial, Bold, 14`).
7. Configure the property *"Input configuration ➔ OnMouseClick"* so that the variable `bGoForward` switches.
8. Configure the second button for back navigation in the same way as described in Steps 5 to 7.

   ⇨ If the variable `bSetURL` has the value `TRUE`, then the website `'http://de.wikipedia.org'` is displayed with the forward and back buttons. When you click the buttons, navigation to the previous and next websites is successful.

See also

- ↳ *Chapter 1.4.5.19.5.38 "Visualization Element 'Web Browser'" on page 2065*
- ↳ *"Input action 'Toggle Variable'" on page 1758*

1.4.5.21.7 Using Client Animation

The example shows a visualization with 3 screens. A menu controls the navigation of the screens. The menu is hidden until it moves into view by means of a hamburger button. During the movement, the transparency of the menu is changed. After the screen is selected, the menu moves back out of view. The animation is computed entirely on the target system. The CODESYS visualization only defines the target values (positions, transparency).

1. Preparation

1. Create a new standard project with the CODESYS Control Win V3 controller.
2. Add a *"Visualization"* object below the *"Application"*. Choose the name `Visu_Main`.
3. Open the Visualization Manager in the editor and select the *"Support client animations and overlay of native elements"* option.

2. Creating the PLC_PRG program

The program checks whether the menu button has been pressed. If the menu bar is not visible (position –300), then the position is moved to the visible area (0). If the menu bar is already visible (position 0), then the position is moved to the hidden area.

1. Open the *“PLC_PRG”* program in the editor.
2. Input the following code into the declaration editor:

```
PROGRAM PLC_PRG
VAR
    iSelection : INT;                   // to switch the
referenced visualization page
    xVisible: BOOL;                     // auxiliary variable to
toggle the menu bar
    iMenuPos : INT := -300;             // position of the menu bar
    xToggle: BOOL;                      // button variable to
toggle the menu bar
END_VAR
```

3. Input the following code into the implementation:

```
IF xToggle THEN
    xToggle := FALSE;
    IF xVisible THEN
        xVisible := FALSE;
        iMenuPos := -300;
    ELSE
        xVisible := TRUE;
        iMenuPos := 0;
    END_IF
END_IF
```

3. Creating the menu bar

The menu bar has 3 menu items. A visualization screen is displayed by clicking the corresponding menu item.

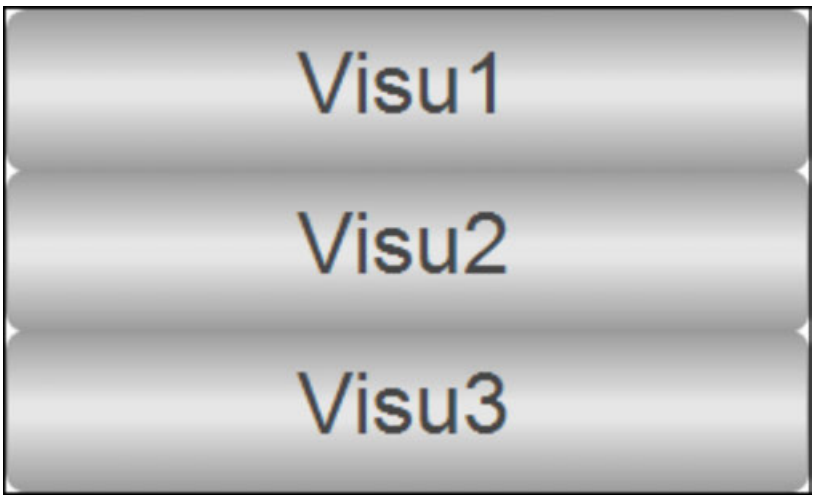
1. Insert a *“Visu_Menu”* visualization below the application.
2. Open the object properties. In the *“Visualization”* tab, set the *“Visualization size”* to a *“Width”* of 300 and a *“Height”* of 180.
3. Open the visualization in the editor.
4. Select the *“Advanced”* option in the *“Properties”* view.
5. In the upper left corner, add a button with a *“Width”* of 300 and a *“Height”* of 60.
6. Label the button as "Visu 1". Set the font size to 24.
7. Open the *“Input configuration ➔ OnMouseClick”* property.
8. Select the *“Execute ST code”* action.
9. Input the following ST code:

```
PLC_PRG.iSelection := 0;
PLC_PRG.xToggle := TRUE;
```

10. Set the *“Button state variable ➔ Digital variable”* property to `PLC_PRG.iSelection=0`
11. Add two more buttons named "Visu 2" and "Visu 3".

12. Edit the button properties of "Visu2" (PLC_PRG.iSelection = 1) and "Visu3" (PLC_PRG.iSelection = 2).

    ⇨ Result:



4. Creating more visualization screens

1. Insert the "Visu1" visualization below the application.
2. Open the object properties. In the *"Visualization"* tab, set the *"Visualization size"* to a *"Width"* of 800 and a *"Height"* of 600.
3. Change the background color of the screen (for example, light gray).
4. Insert a *"Label"* object into the visualization screen and name the element (example: "Visu 1").
5. Insert two more visualizations "Visu2" and "Visu3" below the application. Edit the properties in the same way as for "Visu1".

5. Creating the main visualization screen

On this screen, you can see the menu bar and a button to show or hide the menu bar. The different visualization screens are navigated in a *"Frame"* visualization element.

1. Open the properties of the *"Visu_Main"* visualization. In the *"Visualization"* tab, set the *"Visualization size"* to a *"Width"* of 800 and a *"Height"* of 600.
2. Open the visualization in the editor.
3. Insert a *"Frame"* element into the visualization.

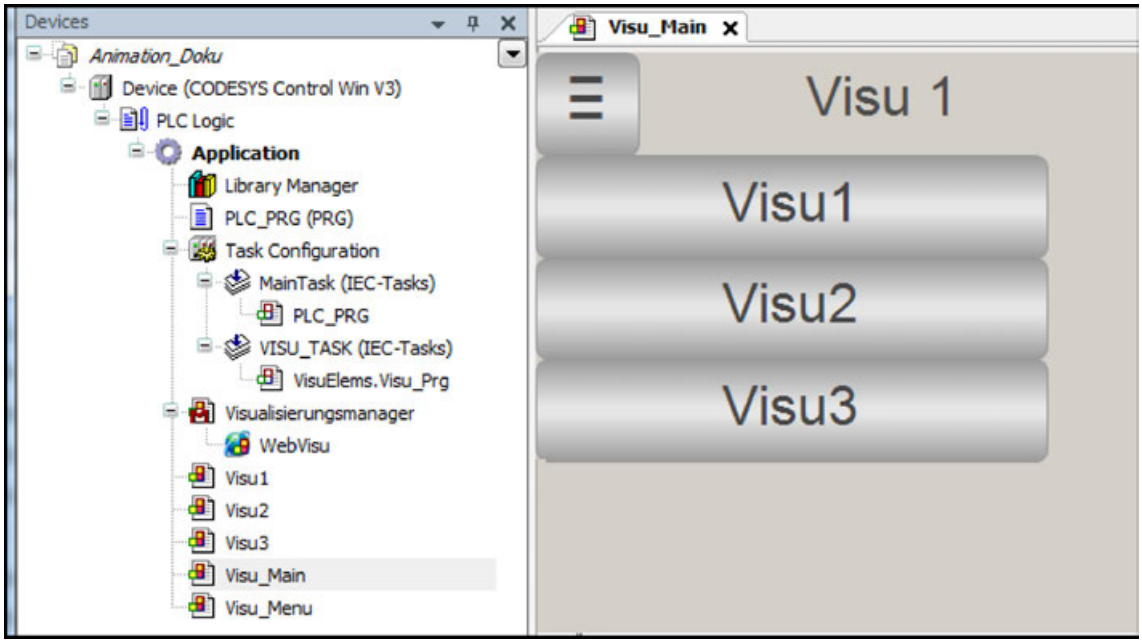
   ⇨ The *"Frame Configuration"* dialog opens.
4. Add the *"Visu1"* (Index 0), *"Visu2"* (Index 1), and *"Visu3"* (Index 2) visualizations.
5. Set the property values of *"Position"* as follows: *"X"* = `0`, *"Y"* = `0`, *"Width"* = `800`, and *"Height"* = `600`.
6. Set the property value of *"Switch frame variable ➔ Variable"* to `PLC_PRG.iSelection`.
7. Insert a *"Button"* element into the visualization.
8. Set the property values of *"Position"* as follows: *"X"* = `0`, *"Y"* = `0`, *"Width"* = `800`, and *"Height"* = `600`.
9. Set the property value of *"Texts ➔ Text"* to `=`.
10. Set the property value of *"Text properties ➔ Font"* to `Arial; 36`.
11. Open the *"Input configuration ➔ OnMouseClick"* property.

12. Select the *“Execute ST code”* action.
13. Input the following ST code:

    ```
    PLC_PRG.xToggle := TRUE;
    ```

14. Set the property value of *“Button state variable ➔ Digital variable”* to `PLC_PRG.xVisible`.
15. Insert the *“Visu_Menu”* visualization element from the *“Current Project”* category into the visualization.
16. Set the property values of *“Position”* as follows: *“X”* = `0`, *“Y”* = `0`, *“Width”* = `300`, and *“Height”* = `180`.
17. Set the property value *“Absolute movement ➔ Movement ➔ X”* to `PLC_PRG.iMenuPos`.
18. Set the property value of *“State variables ➔ Invisible”* to `not(PLC_PRG.xVisible)`.
19. Set the property value of *“Animation duration”* to `2000`.

    ⇨ Result:



See also

- *Chapter 1.4.5.19.5.6 “Visualization Element 'Frame'” on page 1856*

6. Downloading the project to the controller and starting the WebVisu

1. Build the project and download it to the PLC.
2. Start the project.
3. In the browser, connect to the visualization (`http://localhost:8080`).

   ⇨ The WebVisu connects to the controller and the visualization opens.
4. In the visualization, click the menu button.

   ⇨ The menu moves into view.

5. Select a menu item.
   ⇨ The visualization screen is selected and the menu moves back out of view.

1.5 Libraries and solutions

1.5.1 Information on libraries

System libraries

When upgrading Automation Builder or an existing project, new AC500 V2 system libraries are installed automatically. Older library versions will be removed as coexistence of a new library version and an older library version is not possible. Check the available library version in the Library Manager.

Usually, when upgrading Automation Builder or an existing project, new AC500 V2 system libraries are installed automatically and older library versions are removed.

As an exception, for the CANopen device CM598-CN both library versions are available in the Library Manager due to compatibility reasons. However, coexistence of a new library version and an older library version is not possible. In order to avoid compile errors remove the older library version.

↳ *Chapter 1.6.2.9 “Converting an AC500 V2 project to an AC500 V3 project” on page 2430*

Customer libraries

Target change from AC500 V2 to AC500 V3

After a target change from AC500 V2 to AC500 V3 the customer libraries have to be converted manually using the Library Converter . For further information see ↳ *Chapter 1.6.6.1.3 “Later change-over of a target system” on page 3648*.

Some Standard CODESYS libraries are automatically converted during the target change.

Documentation for libraries

- Description for the use of and information about selected libraries ↳ *Chapter 1.5 “Libraries and solutions” on page 2146*.
- Reference for function blocks, functions, structures etc. ↳ *Chapter 1.10 “Reference, function blocks” on page 4292*

1.5.2 Reference to CODESYS (V3)

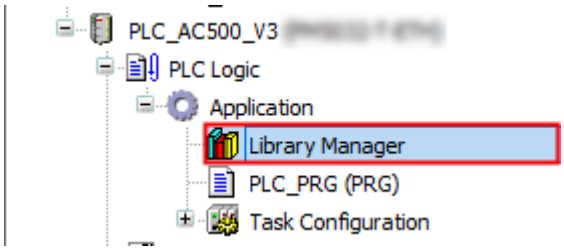
Note that CODESYS V3 libraries are used.

For information on programming, see ↳ *Chapter 1.4.1.16.1 “Information for Library Developers” on page 449*.

1.5.3 Library Manager functionality

The Library Manager contains descriptions of libraries and function blocks.

In the Automation Builder the Library Manager is located under the node *“Application”*.



The Library Manager offers a wide array of functionality for the user. Use cases and how to handle the function blocks of a certain library is described in application examples:

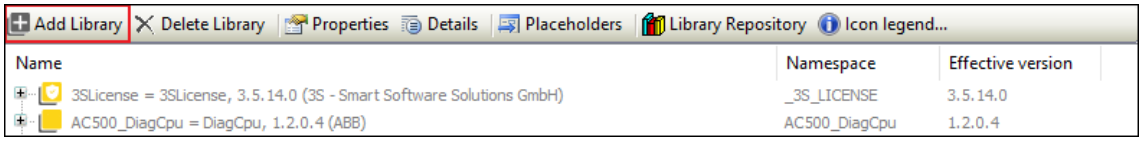
- *StringUtils library*.

1.5.3.1 Search function

In the Library Manager the search function allows you to quickly find any library or function.

To search for a library or function:

1. Select *“Add Library”*.



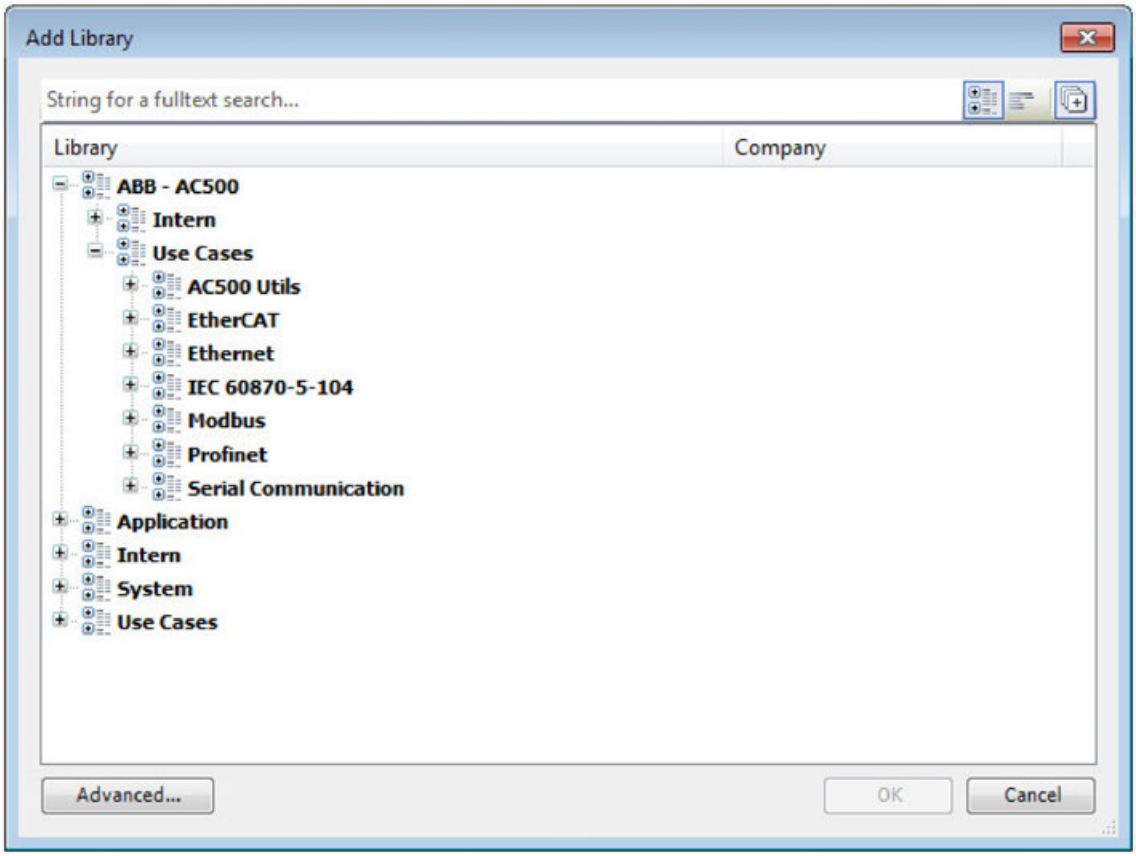
⇨ The Add Library Window opens and a list of all available libararies is displayed.

Libraries in folder *“ABB - AC500”* are created by ABB and tested in combination with Automation Builder.

We recommend to use libraries of subfolder *“Use Cases”* for your project.

Libaries in subfolder *“Intern”* are necessary for internal procedures.

All 3S libraries distributed with Automation Builder are required by ABB libraries and have been tested in combination with AC500 and Automation Builder. Additional 3S libaries that are not distributed with Automation Builder can easily be added. There are no known major issues with using them, however, be aware that they are not tested by ABB.



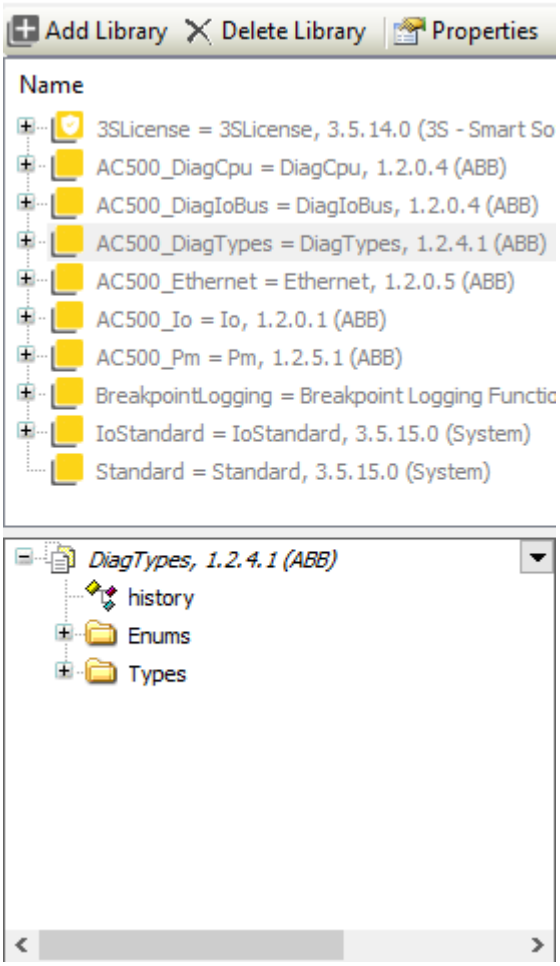
2. Enter the name of the library or function you are searching for.

1.5.3.2 View embedded documentation of all libraries

In the Library Manager you can view embedded documentation of any ABB and 3S libraries.

Precondition: Library must be available in Library Manager.

1. Select a library.
   ⇨ The contents of the library are shown.



2. From the contents select an object.
   ⇨ The corresponding documentation is opened.

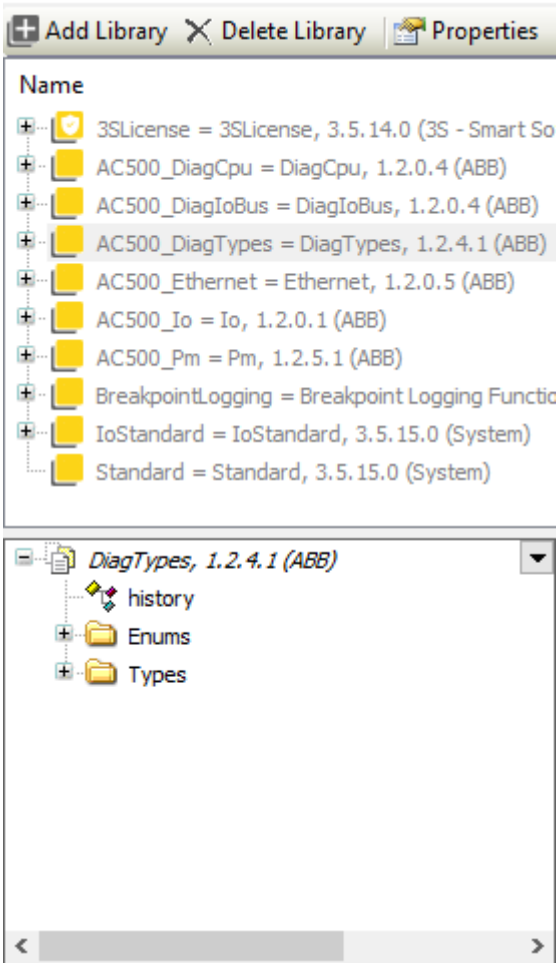
1.5.3.3 Access version history

The Library Manager allows you to access the version history of ABB libraries.

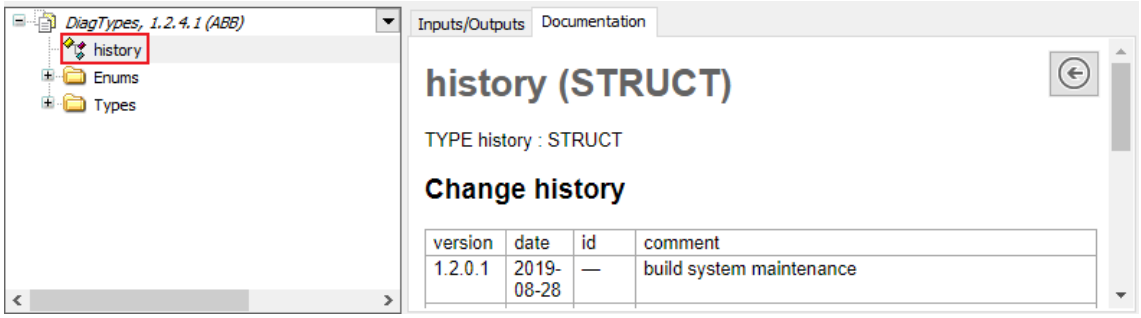
The version history is not available for non ABB libraries

To access the version history of an ABB library:

1. Select a library.
   ⇨ The contents of the library are shown.



2. Select *“history”*.
   ⇨ The version history is shown.

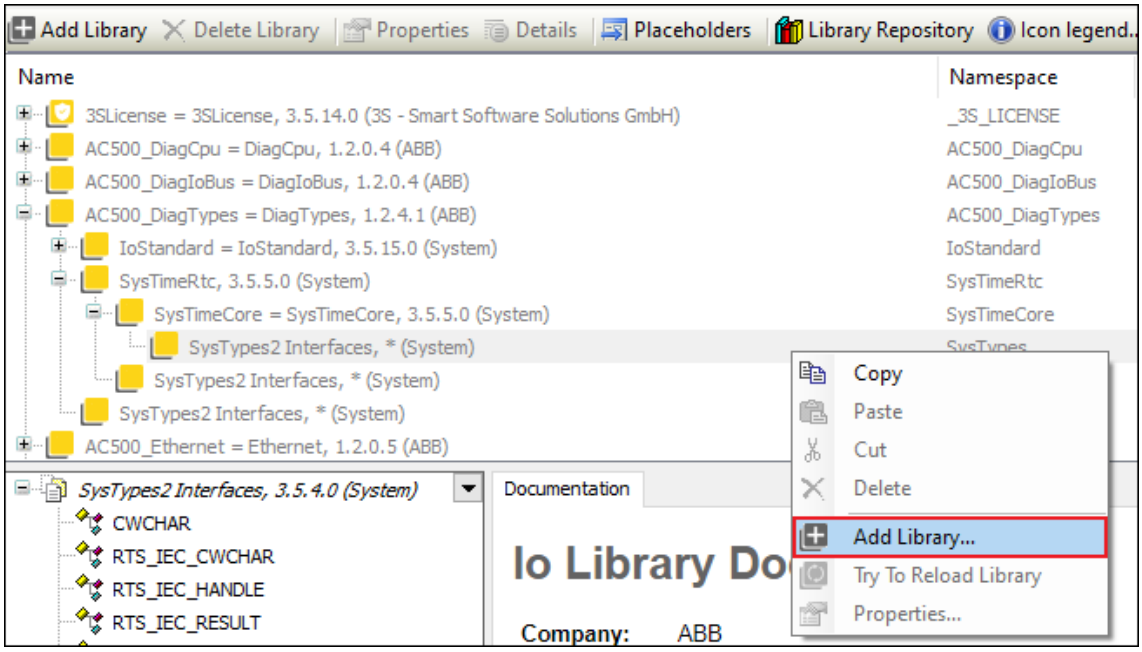


1.5.3.4 Add user defined libraries

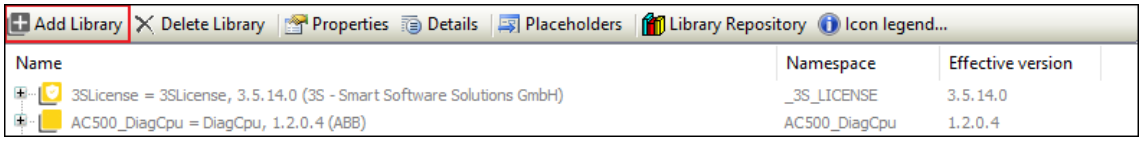
If there are any unresolved library references, you can add user defined libraries.

To add libraries:

1. Right-click on a library.
2. Select *“Add Library”*.



⇨ The Add Library Window opens.



3. Choose the library you want to add.

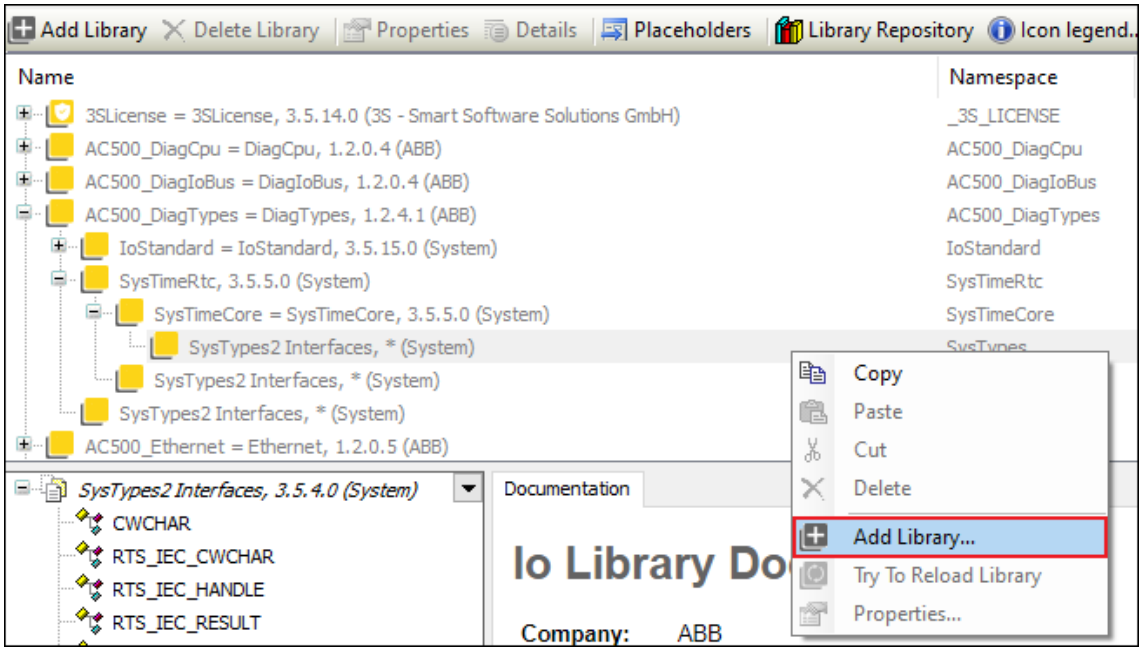
1.5.3.5 Download missing libraries

The Library Manager allows you to automatically download missing 3S libraries from the project that are not available from the library repository.

To download missing libraries:

1. Select *“Download missing libraries”*.

   ⇨ The 'Download missing libraries' window is opened.



2. Select which libraries you want to download.

1.5.4 ACS/DCS drives libraries

1.5.4.1 Introduction

1.5.4.1.1 Scope of the document

The purpose of system technology document is to give an overview and explain the overall concepts of the Drives library in V3. The library contains function blocks to establish communication, to control the ABB ACS / DCS drives from AC500 V3 PLCs.

1.5.4.1.2 Safety instructions and preconditions to use drives library

The user has to read the following instructions and documents before using the libraries:

- All pertinent state, regional, and local safety regulations must be observed when installing and using this product. When functions or devices are used for applications with technical safety requirements, the relevant instructions must be followed.
- Read the complete safety instructions of the user's manuals for the devices you are using, before installation and commissioning.
- Read all safety instructions of the AC500 PLC. See System description AC500 in the online help in Automation Builder.
- Read the user information of the devices and functions you are using, see online help in Automation Builder.
- Installation and commissioning of the drive(s) is not part of this document nor the online help of Automation Builder. Installation and commissioning of the drive(s) must be done according to the related drives manuals and safety instructions.

The library package has been released for the software and firmware versions listed in the readme file of the package only.

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the Drives V3 Library with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e.g. of example programs.

The user must follow all applicable safety instructions and the guidelines mentioned in the user documents of the ABB products.

Read the complete safety instructions for the AC500 before installation and commissioning.

CAUTION!
Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

1.5.4.1.3 Comparison of V2 and V3 drives library

The below table compares the FBs in the V2 library package and corresponding adapted FBs in the V3 library package.

The V2 package (PS553-Drives) has different library files for each protocol and the same is replaced with a single library in V3 (ABB_Drives_AC500).

PS553-Drives Library package (V2)		PS5605-Drives Library package (V3)	
Library Name	**Function Block**	**Library**	**Function Block**
ACSDrives-Base_AC500_V2	ACS3XX_DRIVES_CTRL_BASIC	Not supported – use DrvControlModbusACS	
	ACS_DRIVES_CTRL_ENG	ABB_Drives_AC500	DrvControlModbusEng
	ACS_DRIVES_CTRL_STANDARD		DrvControlModbusACS
	ACS_DRIVES_CTRL_STANDARD_GEN		DrvControlACS
	ACS_MOD_READ_N_PRM		DrvModbusRead
	ACS_MOD_WRITE_N_PRM		DrvModbusWrite
	ACS_REF_SCALING		DrvScaling
ACSDrivesCom-ModRTU_AC500_V20	ACS3XX_COM_MOD_RTU	Not supported	
	ACS_COM_MOD_RTU	ABB_Drives_AC500	DrvModbusRtu
	ACS_COM_MOD_RTU_ENHANCED		DrvModbusRtu
	ACS_COM_MOD_RTU_GEN	ABB_ModbusRtu_AC500	ModRtuToken
	ACS_COM_MOD_RTU_GEN_READ_N_PRM		ModRtuRead
	ACS_COM_MOD_RTU_GEN_WRITE_N_PRM		ModRtuWrite
			ModRtuReadWrite23
ACSDrivesCom-ModTCP_AC500_V22	ACS_COM_MOD_TCP	ABB_Drives_AC500	DrvModbusTcp
	ACS_COM_MOD_TCP_ENHANCED		DrvModbusTcp
ACSDrivesCom-ModTCP_Ext_AC500_V24	ACS_COM_MOD_TCPx	ABB_Drives_AC500	DrvModbusTcp
	ACS_COM_MOD_TCPx_ENHANCED		DrvModbusTcp
DCSDrives_AC500_V24	DCS_DRIVES_CTRL	ABB_Drives_AC500	DrvControlModbusDCS
	DCS_DRIVES_CTRL_GEN		DrvControlDCS
ACSDrives-ComPN_AC500_V24	ACS_PN_WRITE_N_PRM_DPV1	ABB_Drives_AC500	DrvPnWrite
	ACS_PN_READ_N_PRM_DPV1		DrvPnRead
ACSDrives-ComPB_AC500_V24	ACS_PB_READ_N_PRM_DPV1	Will be supported in next Release	
	ACS_PB_WRITE_N_PRM_DPV1		
	ACS_COM_PB	Not supported	
	ACS_COM_PB_PZD		
	ACS_PB_READ_PRM_DPV0		

PS553-Drives Library package (V2)		PS5605-Drives Library package (V3)	
Library Name	**Function Block**	**Library**	**Function Block**
	ACS_PB_WRITE_PRM_DPV0		
		ABB_Drives_AC500	DrvModbusReadWrite23
		ABB_Drives_AC500	DrvModbusRtuBroadcast
		ABB_Drives_AC500	DrvControlCANCiA402

1.5.4.1.4 Overview of the drives library for V3 PLC

This document will briefly explain about communication settings between PLCs with drives, how to control the drives from PLC using the control function blocks.

Each input and output of the function blocks are explained in the integrated documentation in the library.

This library is released for the following products:

- AC500 V3 CPU
- ABB Drives:
  - ACS380, ACS480, ACS580, ACH580, ACQ580, ACSM1, ACS880, DCS550, DCS800, DCS880. Other drives may still work, but are not tested.
  - To use the control blocks the Communication Profile must be *“ABB Drives Profile”* or *“ABB Drives Profile enhanced”*
- Fieldbus Adapters: FENA-01, FENA-11, FENA-21, FSCA-01, FCAN-01, FECA-01, RETA-01, RETA-02, RCAN-01, FPNO-21, FMBT-21.
  Fieldbus adapter support is dependent on the drive and for more details refer the corresponding drive manual.

Drives Library in V3 will support following protocols for the communication:

- Modbus TCP (onboard ETH1 and ETH2 ports)
- Modbus RTU (onboard COM1 port)
- PROFINET (using communication module CM579-PNIO)
- EtherCAT (using communication module CM579-ETHCAT)
- CANopen (onboard CAN port)

Modbus TCP

The following hardware components must be available:

- AC500 V3 PLC with ETH option. Configure onboard ETH1 or ETH2 for Modbus TCP.
- Drive with fieldbus adapter module
  - ACS Drives and DCS880: FENA-01 or FENA-11 or FENA-21 or FMBT-21
  - DCS550 and DCS800: RETA-01
- RJ45 Ethernet cable

ACS drives

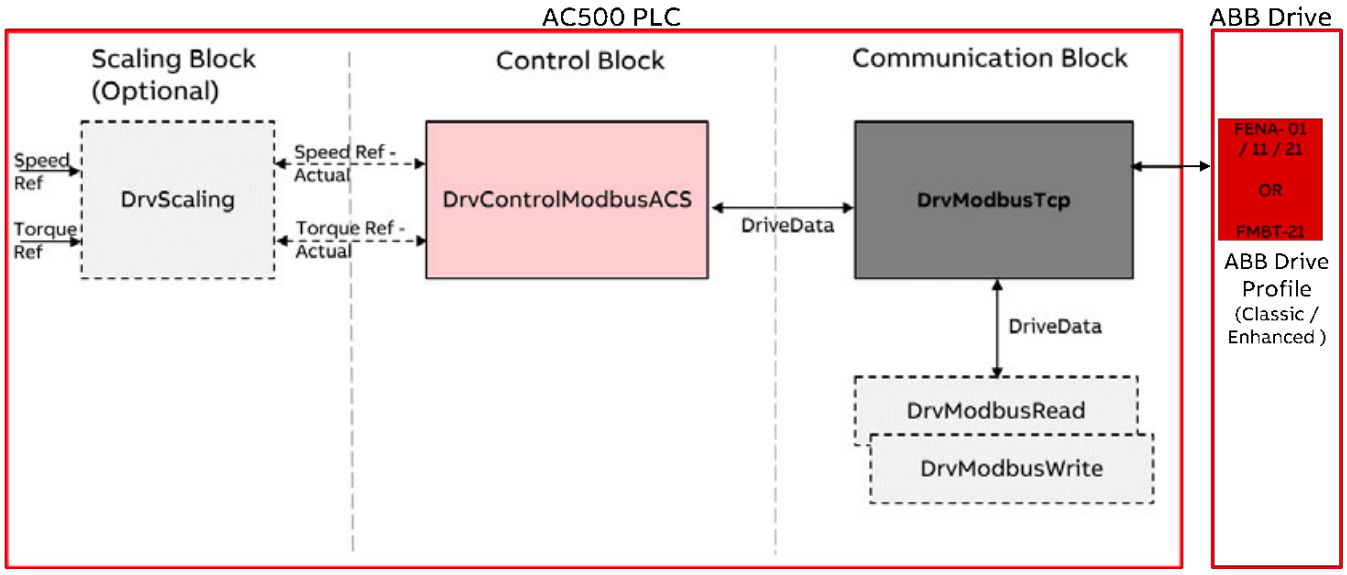


Fig. 12: FB - Overview of Modbus TCP connection with ACS drives

To exchange only status word, actual speed, control word and speed reference:

- Communication profile in drive parameters: ABB Drives classic
- Communication function block in AC500 program:
  Use function block ‘DrvModbusTcp’. *Chapter 1.5.4.2.3.1.8 “DrvModbusTcp” on page 2181*
- DrvModbusTcp
- Control function block in AC500 program:
  Use function block ‘DrvControlModbusACS’. *Chapter 1.5.4.2.3.1.4 “DrvControlModbusACS” on page 2177*
- Scaling of the speed or torque (optional):
  Use function block ‘DrvScaling’. *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive:

- Communication profile in drive parameters: ABB Drives enhanced
- Communication function block in AC500 program:
  Use the function block ‘DrvModbusTcp’ with input EnhancedProfile = TRUE. *Chapter 1.5.4.2.3.1.8 “DrvModbusTcp” on page 2181*
- DrvModbusTcp
- Control function block in AC500 program:
  Use function block ‘DrvControlModbusACS’. *Chapter 1.5.4.2.3.1.4 “DrvControlModbusACS” on page 2177*
- Scaling of the speed or torque (optional): Use function block ‘DrvScaling’. *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange more than above mentioned values use additionally the following blocks:

- Read the values:
  Use the function block ‘DrvModbusRead’. ↳ *Chapter 1.5.4.2.3.1.6 “DrvModbusRead” on page 2180*
- Write the values:
  Use the function block ‘DrvModbusWrite’. ↳ *Chapter 1.5.4.2.3.1.7 “DrvModbusWrite” on page 2181*
- Read Write the values:
  Use the function block ‘DrvModbusReadWrite23’. ↳ *Chapter 1.5.4.2.3.1.11 “DrvModbus-ReadWrite23” on page 2201*

DCS drives

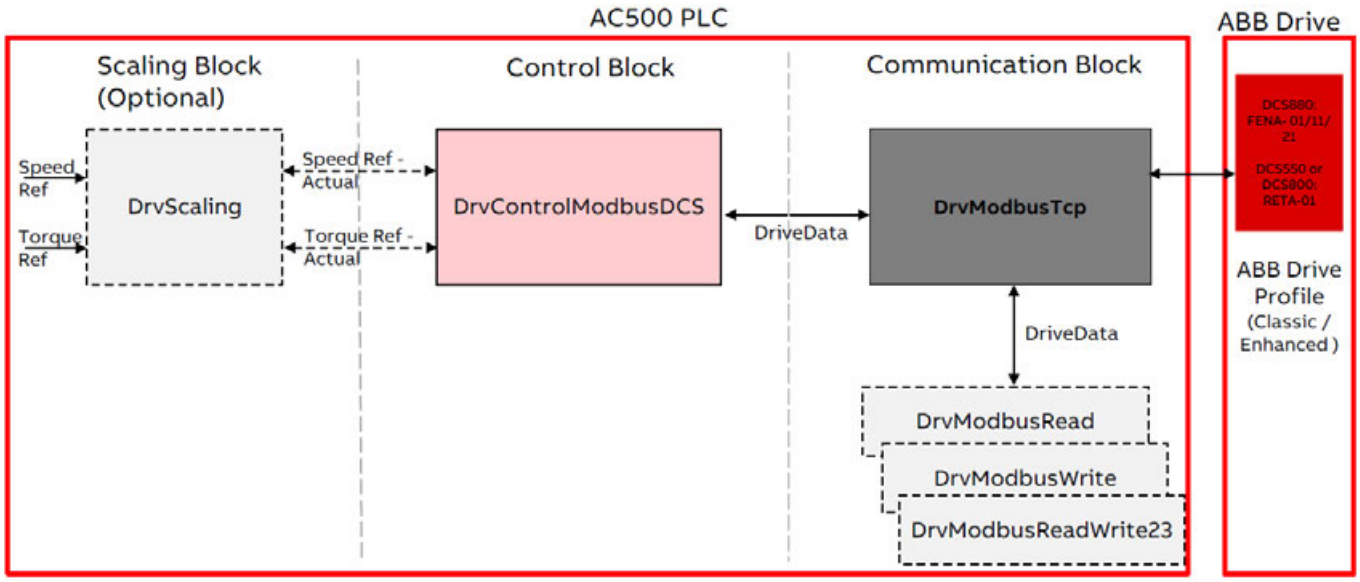


Fig. 13: FB - Overview of Modbus TCP connection with DCS drives

DCS550 and DCS800 drives

To exchange only status word, actual speed, control word and speed reference:

- Communication function block in AC500 program:
  Use function block *“DrvModbusTcp”*. ↳ *Chapter 1.5.4.2.3.1.8 “DrvModbusTcp” on page 2181*
- *“DrvModbusTcp”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusDCS”*. ↳ *Chapter 1.5.4.2.3.1.5 “DrvControlMod-busDCS” on page 2179*
- Scaling of the speed or torque (optional):
  Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange more than above mentioned values use additionally the following blocks:

- Read the values:
  Use the function block *“DrvModbusRead”*. ↳ *Chapter 1.5.4.2.3.1.6 “DrvModbusRead” on page 2180*
- Write the values:
  Use the function block *“DrvModbusWrite”*. ↳ *Chapter 1.5.4.2.3.1.7 “DrvModbusWrite” on page 2181*
- Read write the values:
  Use the function block *“DrvModbusReadWrite23”*. ↳ *Chapter 1.5.4.2.3.1.11 “DrvModbusReadWrite23” on page 2201*

DCS880 drives

To exchange only status word, actual speed, control word and speed reference:

- Communication profile in drive parameters: ABB Drives classic
- Communication function block in AC500 program:
  Use function block *“DrvModbusTcp”*. ↳ *Chapter 1.5.4.2.3.1.8 “DrvModbusTcp” on page 2181*
- *“DrvModbusTcp”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusDCS”*. ↳ *Chapter 1.5.4.2.3.1.5 “DrvControlModbusDCS” on page 2179*
- Scaling of the speed or torque (optional):
  Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive:

- Communication profile in drive parameters: ABB drives enhanced
- Communication function block in AC500 program:
  Use the function block *“DrvModbusTcp”* with input EnhancedProfile = TRUE. ↳ *Chapter 1.5.4.2.3.1.8 “DrvModbusTcp” on page 2181*
- *“DrvModbusTcp”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusDCS”*. ↳ *Chapter 1.5.4.2.3.1.5 “DrvControlModbusDCS” on page 2179*
- Scaling of the speed or torque (optional): Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange more than above mentioned values use additionally the following blocks:

- Read the values:
  Use the function block *“DrvModbusRead”*. ↳ *Chapter 1.5.4.2.3.1.6 “DrvModbusRead” on page 2180*
- Write the values:
  Use the function block *“DrvModbusWrite”*. ↳ *Chapter 1.5.4.2.3.1.7 “DrvModbusWrite” on page 2181*
- Read Write the values:
  Use the function block *“DrvModbusReadWrite23”*. ↳ *Chapter 1.5.4.2.3.1.11 “DrvModbusReadWrite23” on page 2201*

Modbus RTU

The following hardware components must be available:

- AC500 V3 PLC. Configure onboard COM1 for the Modbus RTU communication.
- Drive with
  - ACS Drives and DCS880: Embedded fieldbus or FSCA-01
  - DCS550 and DCS800: Embedded fieldbus or RMBA-01
  - Twisted pair serial cable

ACS drives

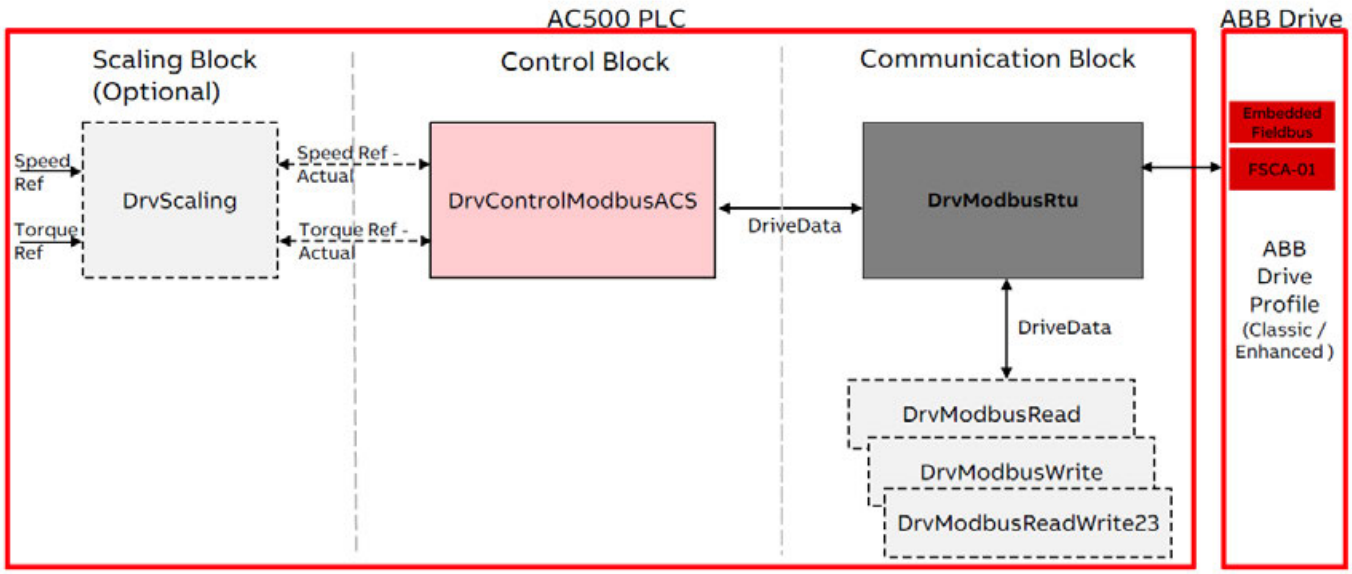


Fig. 14: FB - Overview of Modbus RTU connection with ACS drives

To exchange only status word, actual speed, control word and speed reference:

- Communication profile in drive parameters: ABB Drives classic
- Communication function block in AC500 program:
  Use function block *“DrvModbusRtu”*. ↳ *Chapter 1.5.4.2.3.1.9 “DrvModbusRtu” on page 2188*
- DrvModbusRtu
- Control function block in AC500 program:
  Use function block *“DrvControlModbusACS”* . ↳ *Chapter 1.5.4.2.3.1.4 “DrvControlModbusACS” on page 2177*
- Scaling of the speed or torque (optional):
  Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive:

- Communication profile in drive parameters: ABB Drives enhanced
- Communication function block in AC500 program:
  Use the function block *“DrvModbusRtu”*. ↳ *Chapter 1.5.4.2.3.1.9 “DrvModbusRtu” on page 2188*
- *“DrvModbusRtu”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusACS”*. ↳ *Chapter 1.5.4.2.3.1.4 “DrvControlModbusACS” on page 2177*
- Scaling of the speed or torque (optional): Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange more than above mentioned values use additionally the following blocks:

- Read the values:
  Use the function block *“DrvModbusRead”*. ↳ *Chapter 1.5.4.2.3.1.6 “DrvModbusRead” on page 2180*
- Write the values:
  Use the function block *“DrvModbusWrite”*. ↳ *Chapter 1.5.4.2.3.1.7 “DrvModbusWrite” on page 2181*
- Read Write the values:
  Use the function block *“DrvModbusReadWrite23”*. ↳ *Chapter 1.5.4.2.3.1.11 “DrvModbusReadWrite23” on page 2201*

DCS drives

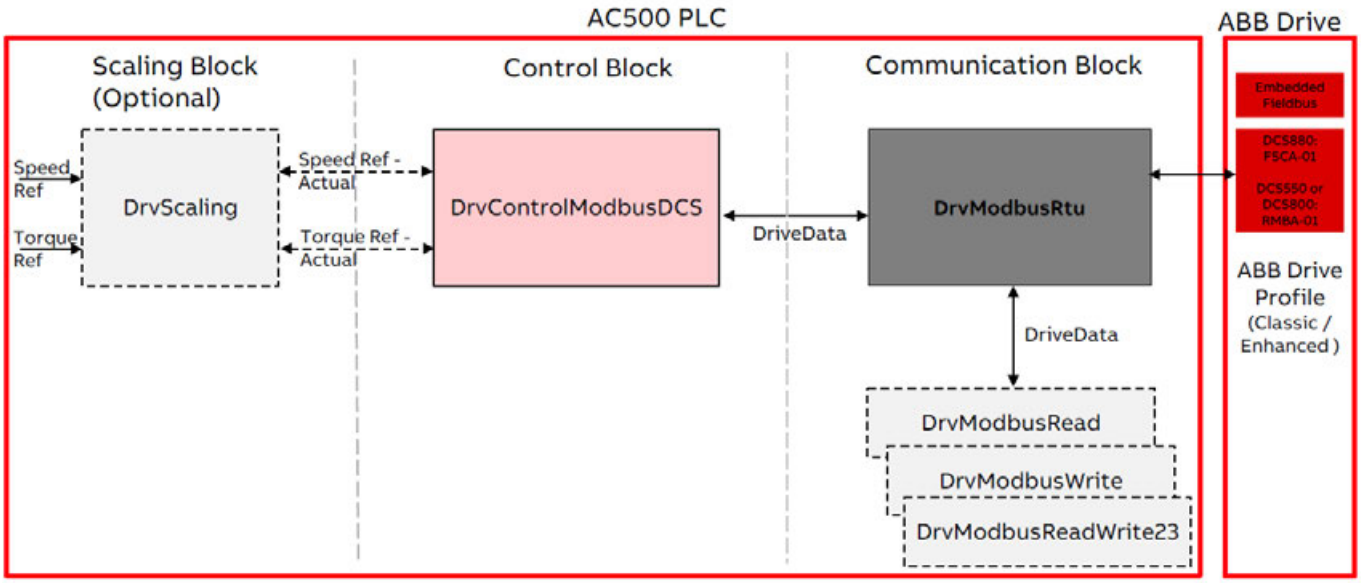


Fig. 15: FB - Overview of Modbus RTU connection with DCS drives

DCS550 and DCS800 drives

To exchange only status word, actual speed, control word and speed reference:

- Communication function block in AC500 program:
  Use function block *“DrvModbusRtu”*. ↳ *Chapter 1.5.4.2.3.1.9 “DrvModbusRtu” on page 2188*
- *“DrvModbusRtu”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusDCS”*. ↳ *Chapter 1.5.4.2.3.1.5 “DrvControlModbusDCS” on page 2179*
- Scaling of the speed or torque (optional):
  Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange more than above mentioned values use additionally the following blocks:

- Read the values:
  Use the function block *“DrvModbusRead”*. ↳ *Chapter 1.5.4.2.3.1.6 “DrvModbusRead” on page 2180*
- Write the values:
  Use the function block *“DrvModbusWrite”*. ↳ *Chapter 1.5.4.2.3.1.7 “DrvModbusWrite” on page 2181*
- Read Write the values:
  Use the function block *“DrvModbusReadWrite23”*. ↳ *Chapter 1.5.4.2.3.1.11 “DrvModbusReadWrite23” on page 2201*

DCS880 drives

To exchange only status word, actual speed, control word and speed reference:

- Communication profile in drive parameters: ABB Drives classic
- Communication function block in AC500 program:
  Use function block *“DrvModbusRtu”*. ↳ *Chapter 1.5.4.2.3.1.9 “DrvModbusRtu” on page 2188*
- *“DrvModbusRtu”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusDCS”*. ↳ *Chapter 1.5.4.2.3.1.5 “DrvControlModbusDCS” on page 2179*
- Scaling of the speed or torque (optional):
  Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive:

- Communication profile in drive parameters: ABB Drives enhanced
- Communication function block in AC500 program:
  Use the function block *“DrvModbusRtu”*. ↳ *Chapter 1.5.4.2.3.1.9 “DrvModbusRtu” on page 2188*
- *“DrvModbusRtu”*
- Control function block in AC500 program:
  Use function block *“DrvControlModbusDCS”*. ↳ *Chapter 1.5.4.2.3.1.5 “DrvControlModbusDCS” on page 2179*
- Scaling of the speed or torque (optional): Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

To exchange more than above mentioned values use additionally the following blocks:

- Read the values:
  Use the function block *“DrvModbusRead”*. *Chapter 1.5.4.2.3.1.6 “DrvModbusRead” on page 2180*
- Write the values:
  Use the function block *“DrvModbusWrite”*. *Chapter 1.5.4.2.3.1.7 “DrvModbusWrite” on page 2181*
- Read Write the values:
  Use the function block *“DrvModbusReadWrite23”*. *Chapter 1.5.4.2.3.1.11 “DrvModbus-ReadWrite23” on page 2201*

PROFINET

The following hardware components must be available:

- AC500 V3 PLC with CM579-PNIO (PROFINET Master communication module)
- Drive with fieldbus adapter module
  - ACS Drives and DCS880: FENA-01 or FENA-11 or FENA-21
  - DCS550 and DCS800: RETA-02
- RJ45 Ethernet cable

The following values should be mapped in the fieldbus configuration of the drive and the configuration of AC500. These settings must be done in the Automation Builder hardware configuration.

- Drive → AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 → Drive: Control word and reference value 1 (speed) and optional reference value 2 (torque).

The following function blocks can be configured in the AC500 program.

- Communication profile: ABB Drives Profile
- Control block:
  - ACS Drives: Use function block ‘DrvControlACS’. *Chapter 1.5.4.2.3.1.2 “DrvControlACS” on page 2173*.
  - DCS Drives: Use function block ‘DrvControlDCS’. *Chapter 1.5.4.2.3.1.3 “DrvControlDCS” on page 2175*.
- Scaling of the speed or torque (optional): Use function block ‘DrvScaling’. *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*.
- PROFINET read function block. *Chapter 1.5.4.2.3.1.14 “DrvPNRead” on page 2205*
- PROFINET write function block. *Chapter 1.5.4.2.3.1.15 “DrvPnWrite” on page 2206*

ACS drives

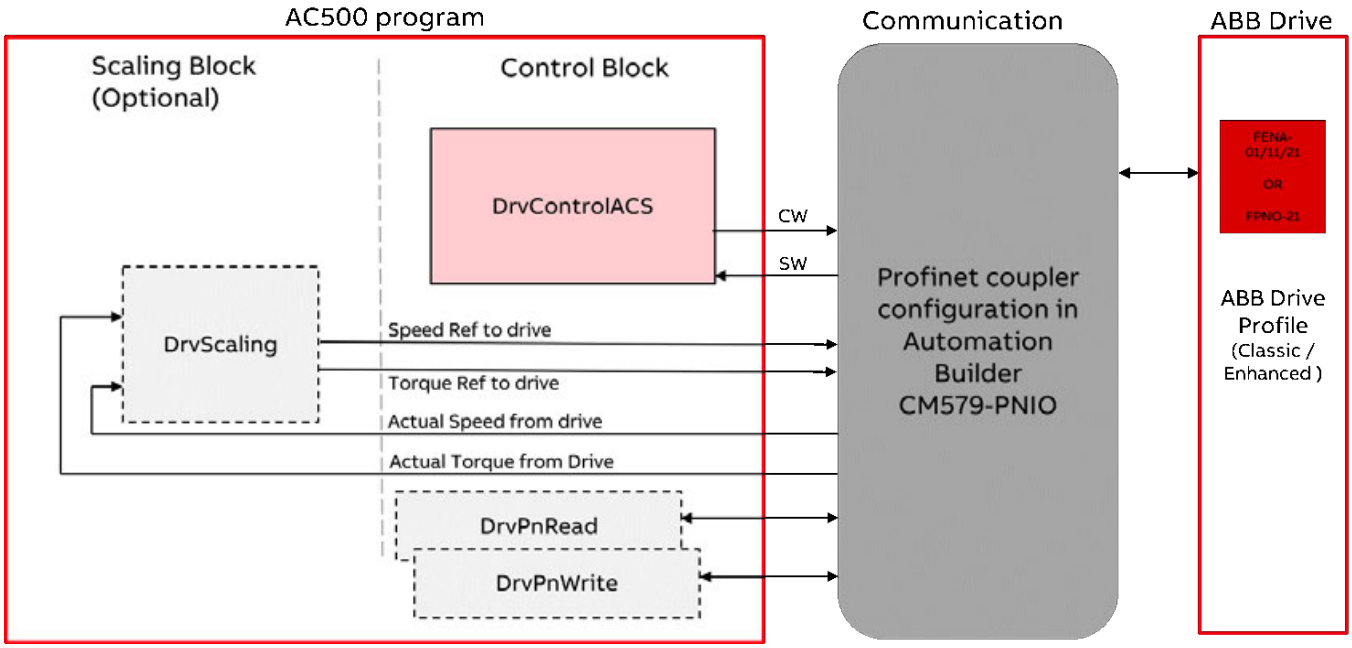


Fig. 16: FB - Overview of PROFINET connection with ACS drives

DCS drives

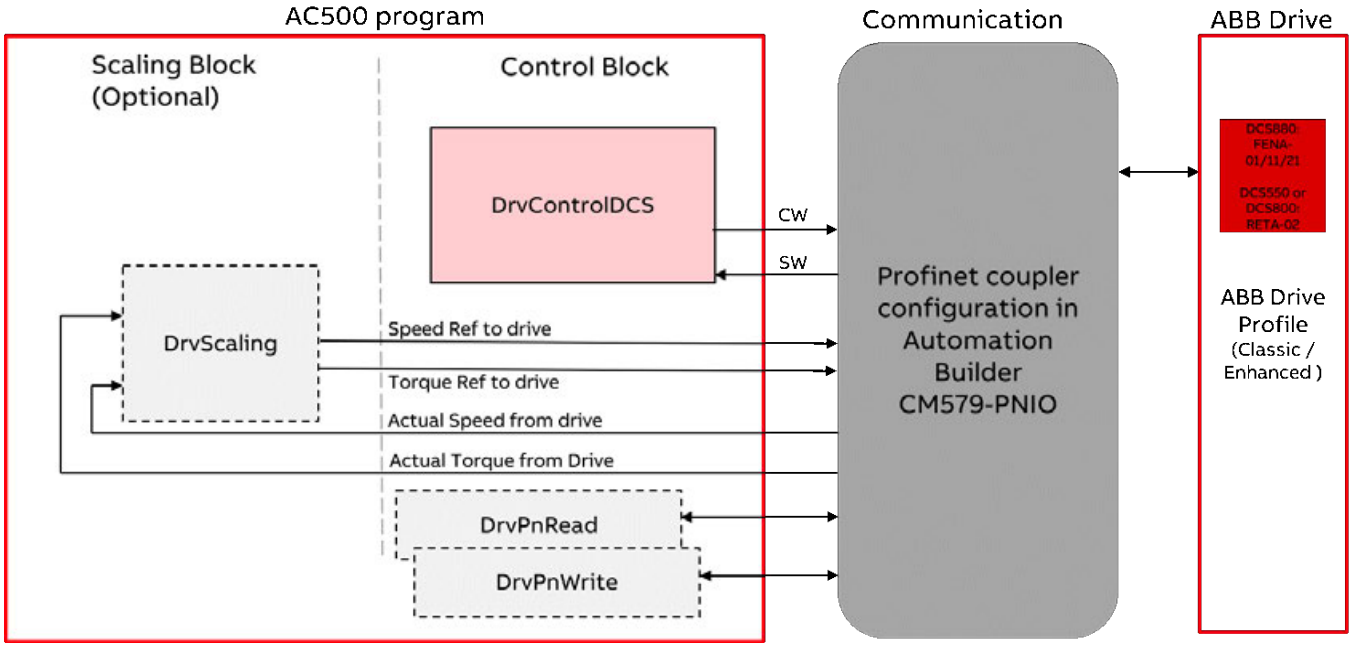


Fig. 17: FB - Overview of PROFINET connection with DCS drives

EtherCAT

The following hardware components must be available:

- AC500 V3 PLC with CM579-ETHCAT (EtherCAT Master communication module)
- Drive with fieldbus adapter module
  - ACS Drives and DCS880: FECA-01
  - DCS550 and DCS800: RECA-01
- RJ45 Ethernet cable

The following values should be mapped in the fieldbus configuration of the drive and the configuration of AC500. These settings must be done in the Automation Builder hardware configuration.

- Drive → AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 → Drive: Control word and reference value 1 (speed) and optional reference value 2 (torque).

A direct Ethernet cable from CM579-ETHCAT to FECA-01 module is recommended, connection through switch is not recommended since it will slow down the connectivity. Also, the drives need to be connected in the same sequence as they are added in the Automation Builder when multiple drives are connected.

The following function blocks can be configured in the AC500 program.

- Communication profile: ABB Drives Profile
- Control block:
  - ACS Drives: Use function block *“DrvControlACS”*. ↳ *Chapter 1.5.4.2.3.1.2 “DrvControlACS” on page 2173*
  - DCS Drives: Use function block *“DrvControlDCS”*. ↳ *Chapter 1.5.4.2.3.1.3 “DrvControlDCS” on page 2175*
- Scaling of the speed or torque (optional): Use function block *“DrvScaling”*. ↳ *Chapter 1.5.4.2.3.1.1 “DrvScaling” on page 2172*

ACS drives

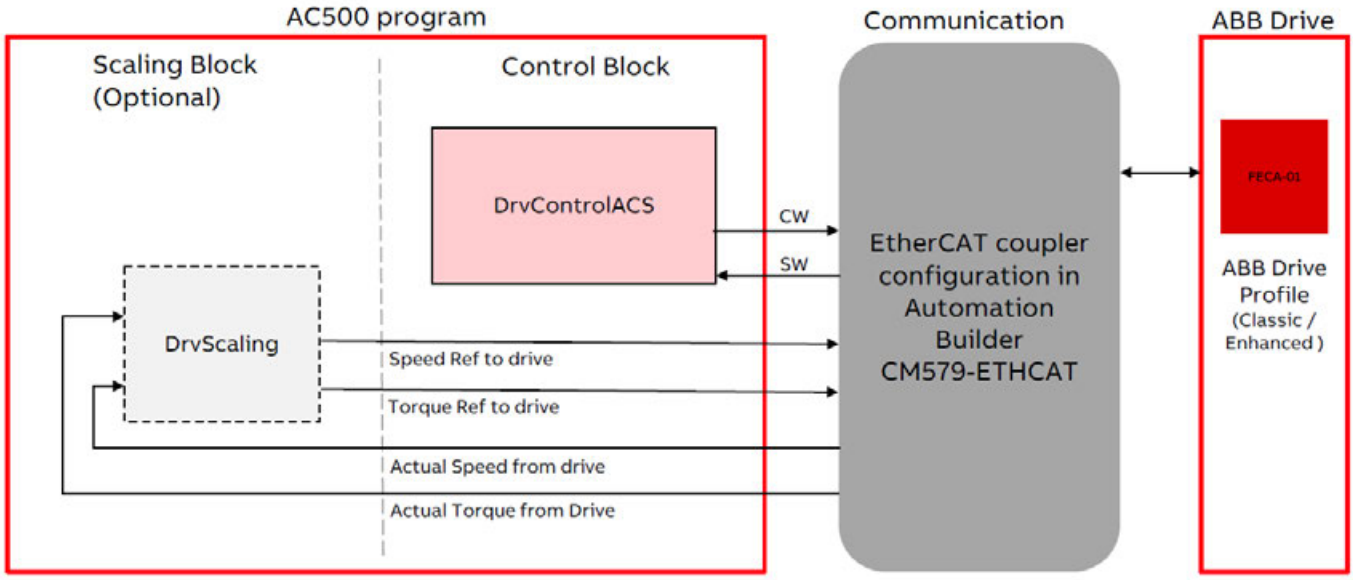


Fig. 18: FB - Overview of EtherCAT connection with ACS drives

DCS drives

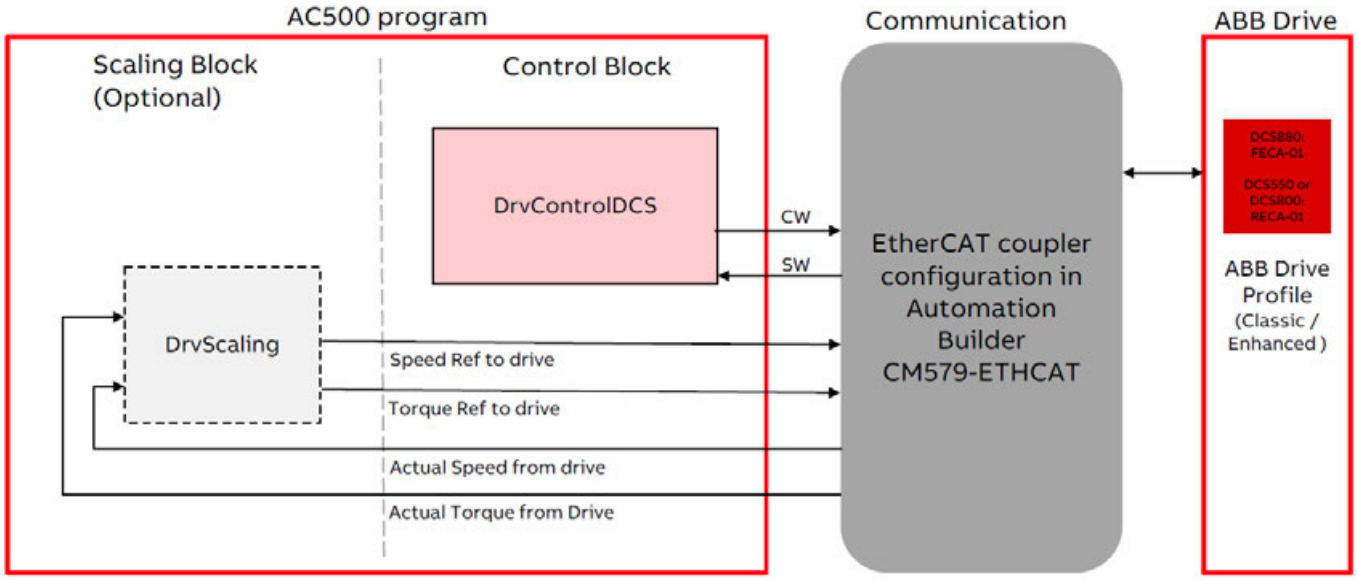


Fig. 19: FB - Overview of EtherCAT connection with DCS drives

CANopen

The following hardware components must be available:

- AC500 V3 PLC. Configure onboard CAN port for CANopen communication.
- Drive with fieldbus adapter module
  - ACS Drives and DCS880: FCAN-01
  - DCS550 and DCS800: RCAN-01
- CANopen communication cable with 120 Ω resistor.

The following values should be mapped in the fieldbus configuration of the drive and the configuration of AC500. These settings must be done in the Automation Builder hardware configuration.

- Drive → AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 → Drive: Control word and reference value 1 (speed) and optional reference value 2 (torque).

The following function blocks can be configured in the AC500 program.

- Communication profile: ABB Drives Profile
- Control block:
  - ACS Drives: Use function block *"DrvControlACS"*. ↳ *Chapter 1.5.4.2.3.1.2 "DrvControlACS" on page 2173*
  - DCS Drives: Use function block *"DrvControlDCS"*. ↳ *Chapter 1.5.4.2.3.1.3 "DrvControlDCS" on page 2175*
- Scaling of the speed or torque (optional): Use function block *"DrvScaling"*. ↳ *Chapter 1.5.4.2.3.1.1 "DrvScaling" on page 2172*

ACS drives

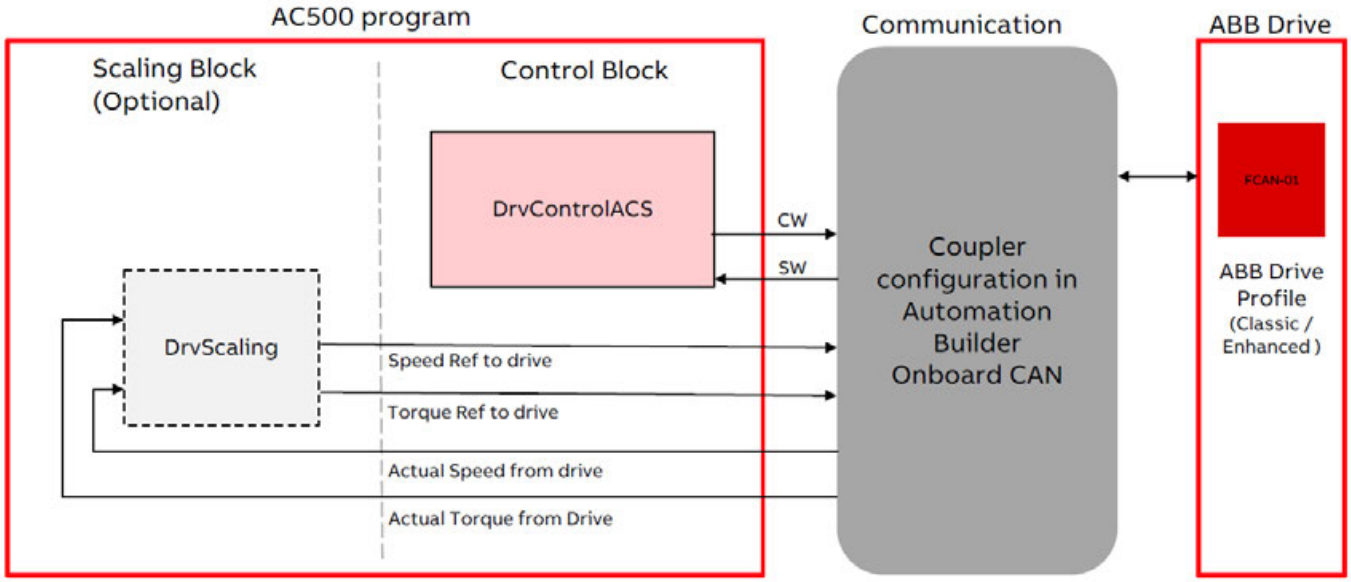


Fig. 20: FB - Overview of CANopen connection with ACS drives

DCS drives

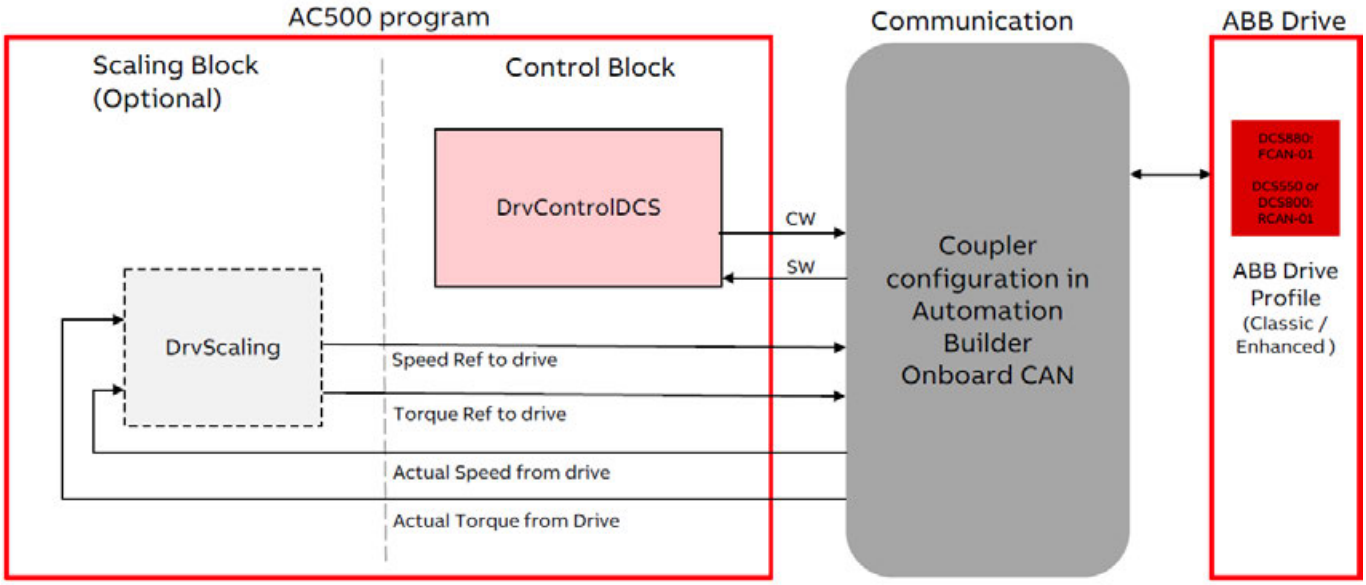


Fig. 21: FB - Overview of CANopen connection with DCS drives

CANopen with CAN CiA402 Profile for generic Drives

The following hardware components must be available:

- AC500 V3 PLC. Configure onboard CAN port for CANopen communication.
- Any drive with CAN fieldbus adapter module and CAN CiA402 profile.
- CANopen communication cable with 120 Ω resistor.

The following values should be mapped in the fieldbus configuration of the drive and the configuration of AC500. These settings must be done in the Automation Builder hardware configuration.

- Drive → AC500: Status word and actual speed.
- AC500 → Drive: Control word and reference speed.

The following function blocks can be configured in the AC500 program.

- Communication profile: CANopen device profile CiA402
- Control block: Use function block *“DrvControlCANCiA402”*. ↳ *Chapter 1.5.4.2.3.1.13 “DrvControlCANCiA402” on page 2204*

General drives with CAN CiA402 interface

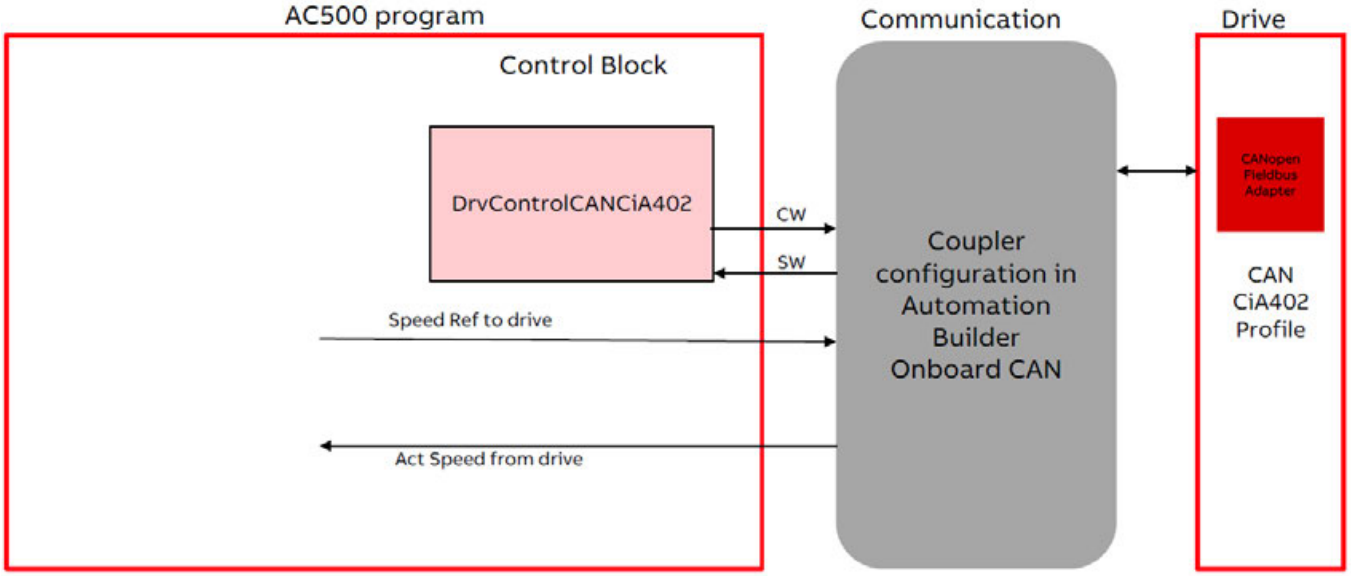


Fig. 22: FB - Overview of CANopen CiA402 with any drives

1.5.4.1.5 Compatibility

To check the compatibility of the drives and their communication modules please refer to the following table, it shows the tested combinations.

Communication	PLC communication modules		Drive fieldbus adapter module			Drive		
	PLC coupler	Firmware version	FBA	FBA comm sw ver	FBA appl sw ver	Drive	Firmware version	Drive rating ID
Modbus RTU - Classic	Onboard		FSCA-01		1.63	ACS580	1.70.4.0 (CCON-11)	ACS580-01-12A6-4
			Embedded			ACS380	2.04.0.3	ACS380-04-XX-01A8-4
			Embedded			ACS480	2.06.255.5	ACS480-04-02A7-4
Modbus RTU – Enhanced	Onboard		FSCA-01		1.63	ACS880	2.8.2	ACS880-01-04A0-3
Modbus TCP	Onboard ETH1 / ETH2		RETA-01	1.30	3.05	DCS800	3.7	
Modbus TCP - Enhanced	Onboard ETH1 / ETH2		FENA-21		3.20	ACH580	2.06.0.2	ACH580-01-02A6-4
PROFINET	CM579-PNIO	2.8.6.21	FENA-21		3.20	ACS880	2.82	ACS880-01-04A0-3
			FENA-21		3.20	ACSM1	UMFI2000 (N2020)	ACSM1-03A0-4
EtherCAT	CM579-ETHCAT	4.4.3.21	FECA-01		1.31	ACQ580	2.05.0.4	ACQ580-01-02A6-4
CANopen (ABB Profile)	Onboard		FCAN-01		1.16	ACSM1	UMFI2000 (N2020)	ACSM1-03A0-4
CANopen (CiA402)	Onboard		FCAN-01		1.16	ACS380	2.04.0.3	ACS380-04-XX-01A8-4

1.5.4.2 Overview of the library

1.5.4.2.1 Installation

The library is part of the Automation Builder 2.2. or higher. Use the Library manager to add the library into project.

For more details on the package, refer to the release notes of the latest Automation Builder.

1.5.4.2.2 Hardware and software requirement

Hardware	Software
AC500 V3 PLCs: PM5630-2ETH, PM5650-2ETH, PM5670-2ETH and PM5675-2ETH	Automation Builder 2.2. or higher
ABB Drive: ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1, DCS550, DCS800, DCS880. (other drives may work, but are not tested)	Drive Composer Pro, Drive Studio, Drive Window or Drive Window Light
Fieldbus adapter module: FSCA-01, RMBA-01 FENA-01 / FENA-11 / FENA-21, RETA-01, FPNO-21, FMBT-21 FECA-01, RETA-02 FCAN-01, RCAN-01 (other fieldbus adapter modules may work, but are not tested)	

Drive configuration tool and fieldbus adapter module support is dependent on the drive used, for the compatible tool details refer to the drive manual.

1.5.4.2.3 Description of the library

This chapter briefly explains the functions, function blocks, structures, enumerations and visualization present in the library.

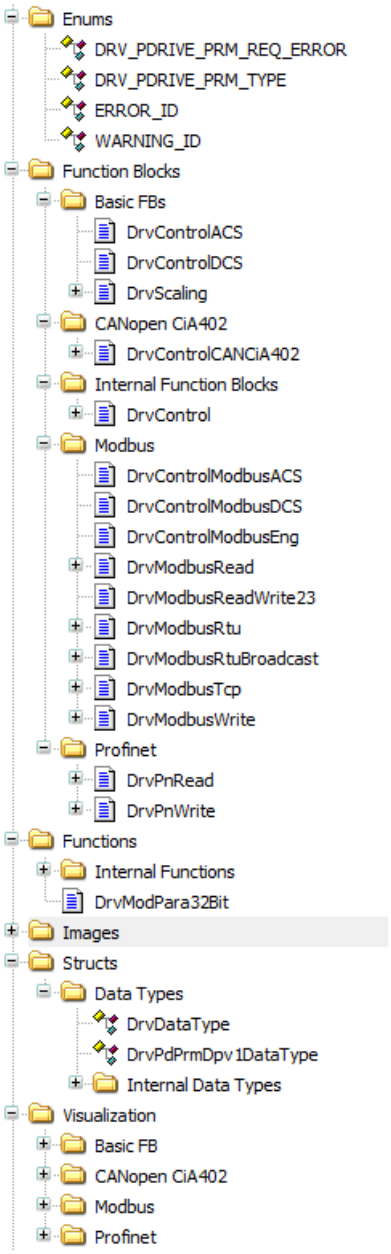


Fig. 23: FB - Overview of the Drives Library

Function blocks

DrvScaling

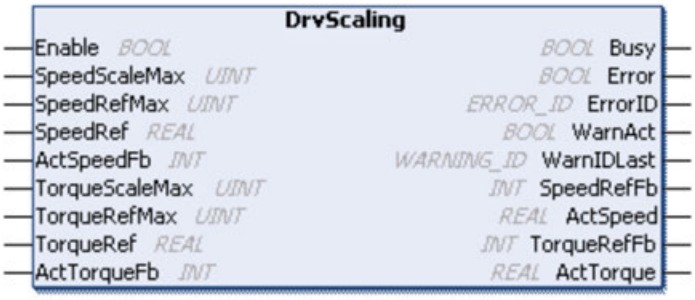
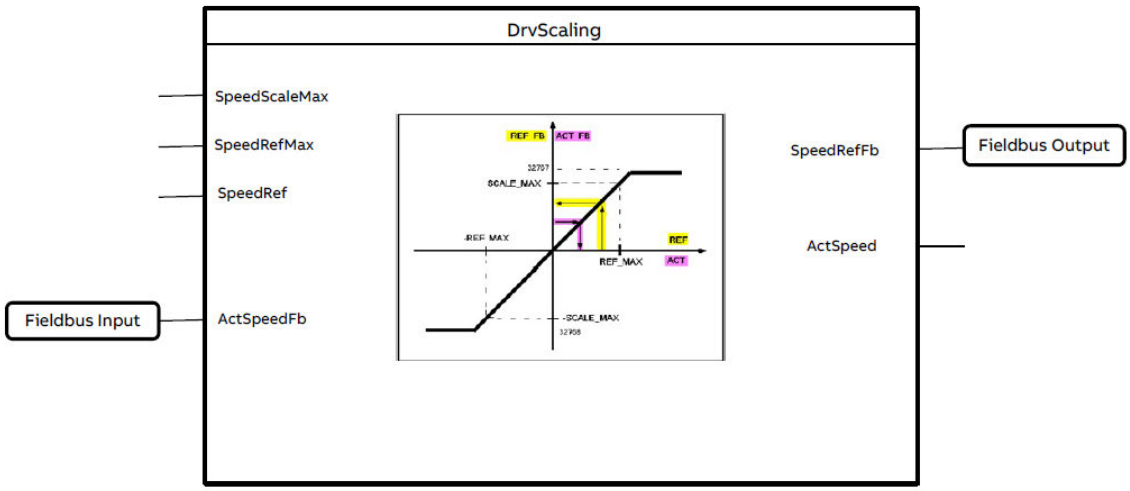


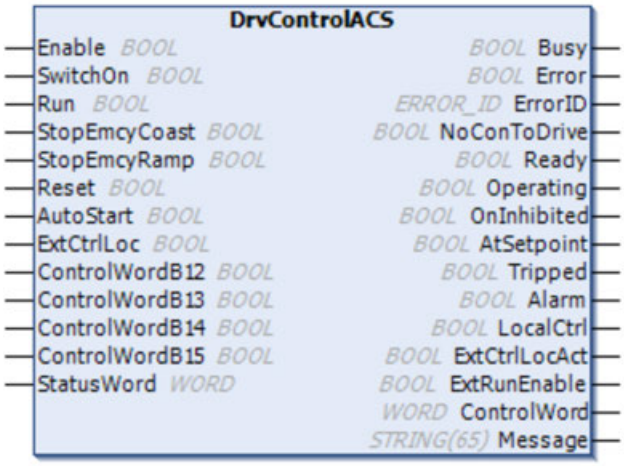
Fig. 24: DrvScaling

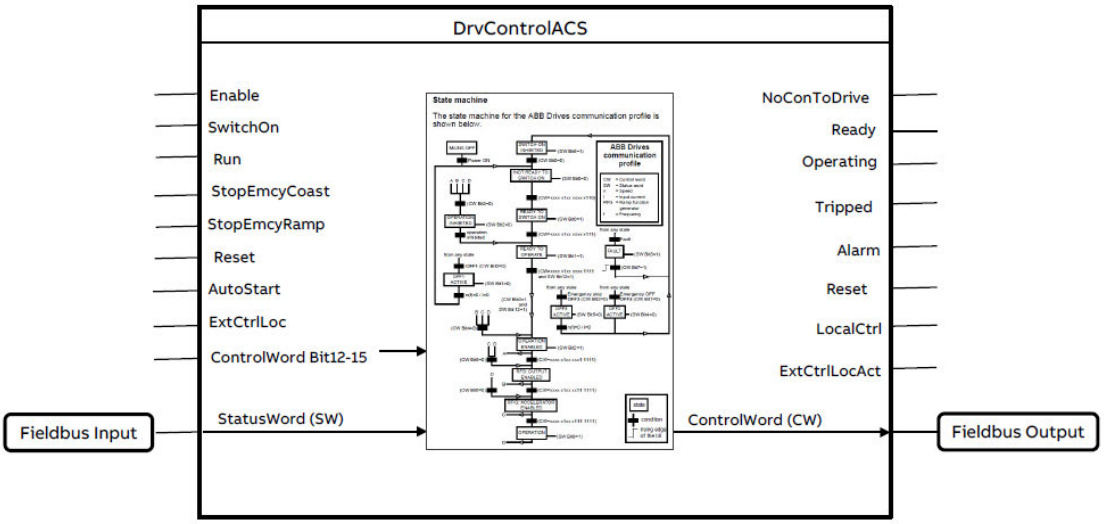


DrvScaling function block is used to scale the speed or torque reference to the drive based on the maximum values defined.

Function block *"DrvScaling"* can be used to scale the variables from fieldbus equivalent values to values used in the program. Fieldbus variables are given in fieldbus equivalent values as INT values. With the scaling a conversion from INT (fieldbus) to REAL (program) and vice versa is performed. Reference1 and Actual Value1 (speed) are mostly given in the range of -20000 ... +20000. Reference2 and Actual Value2 (torque) are mostly given in the range of 0 ... +10000.

DrvControlACS



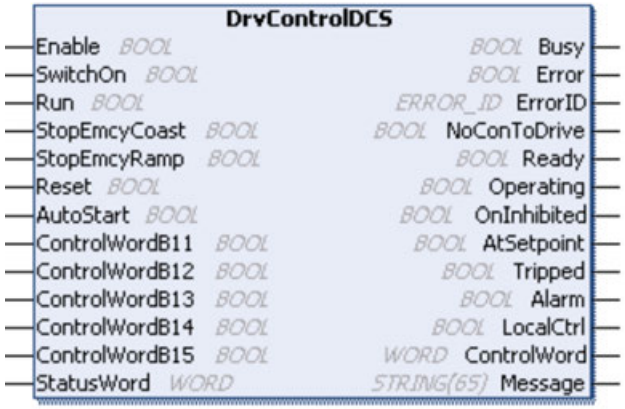


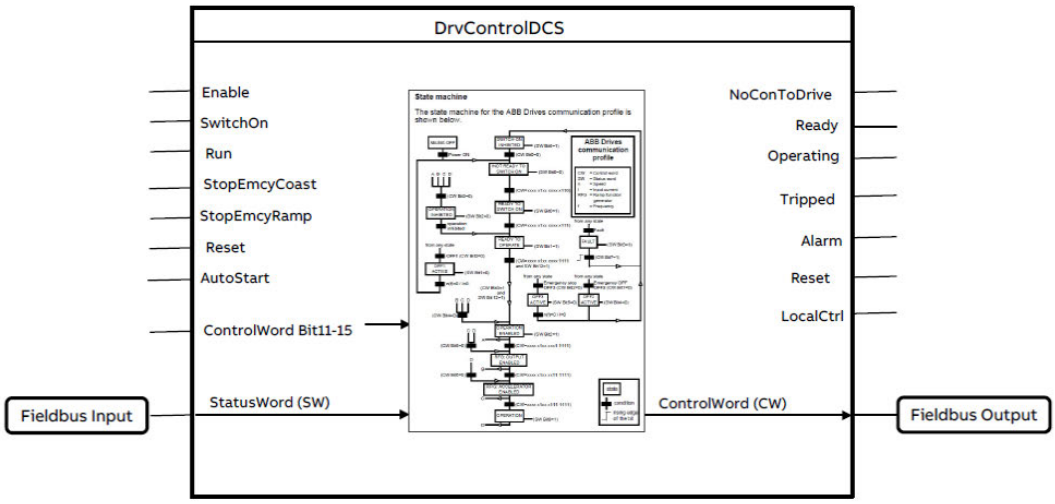
This function block can be used to control ACS drives with ABB drives profile using direct input of status word (SW) from drive via any supported fieldbus communication like PROFINET, EtherCAT, CANopen.

Control word (CW) will be built by the function block according to the ABB drives profile state machine. Output CW has to be send to the drive via any fieldbus communication supported. Function block provides standard start/stop signals to control the drive and standard diagnosis signals are read from the drive.

Drive Parameter	**ACS380/ ACS480/ ACS580/ ACH580/ ACQ580/ ACS880**	**ACSM1**	**Comment**
EXT1 COMMANDS	20.01 = Fieldbus A	10.01 = FBA	Fieldbus interface as source for start and stop
EXT1 / EXT2 SEL	19.11 = MCW Bit11 (06.01)	34.01 = P02.12 bit 15	Fieldbus interface as source to switch to EXT2 control place
REF1 SELECT	22.11 = FBA ref1	24.01 = FBA ref1	Fieldbus interface as source to speed reference
FAULT RESET SELECT	31.11 = P06.01 bit 7	10.08 = P02.12 bit 8	Fieldbus interface as source for fault reset
PROFILE	51.02 = Drives Classic / Enhanced	51.02 = Drives Classic / Enhanced	Control profile to ABB Drives profile classic or enhanced

DrvControlDCS



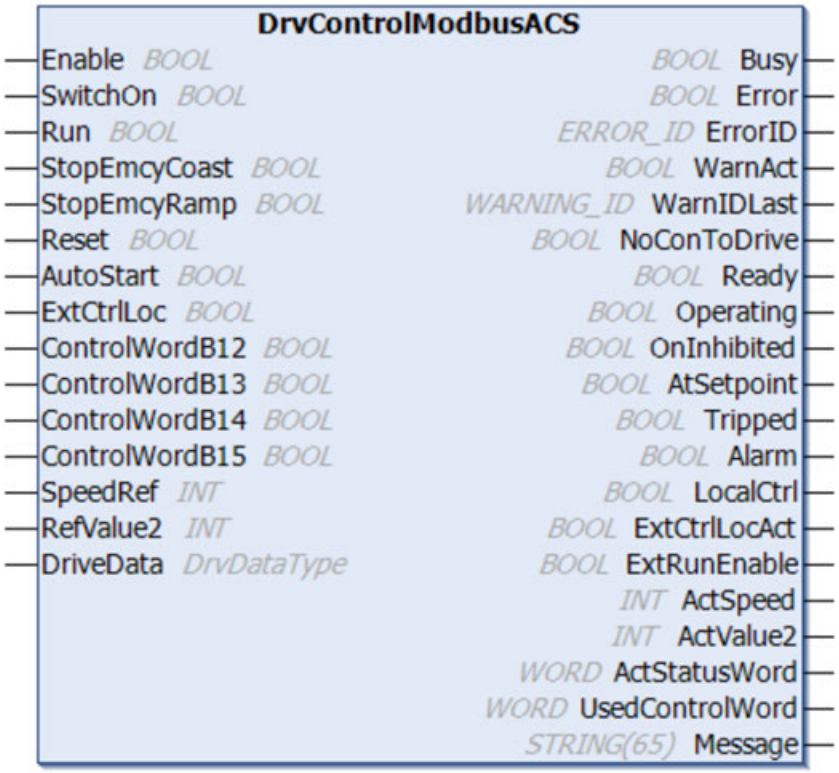


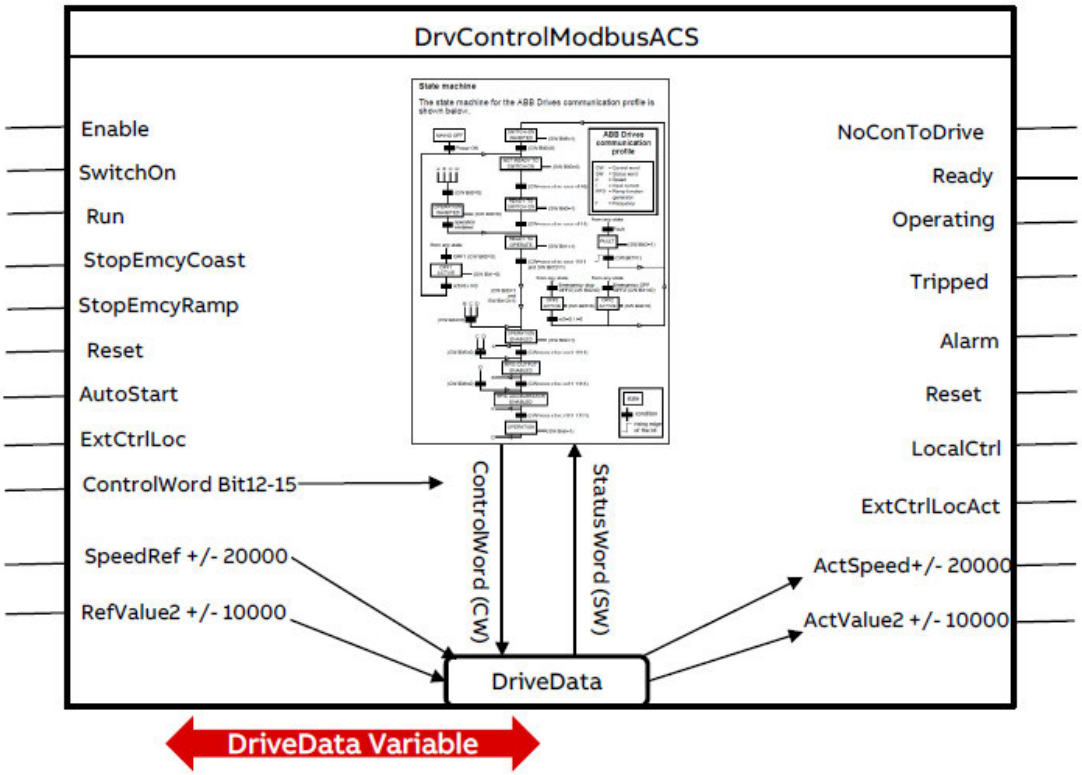
This function block can be used to control DCS drives with ABB drives profile using direct input of status word (SW) from drive via any supported fieldbus communication like PROFINET, EtherCAT, CANopen.

Control word (CW) will be built by the function block according to the ABB drives profile state machine. Output CW must be sent to the drive via any fieldbus communication supported. Function block provides standard start/stop signals to control the drive and standard diagnosis signals are read from the drive.

Drive Parameter	DCS550	DCS800	DCS880	Comment
EXT1 COMMANDS	10.01 = Main Ctrl Word	10.01 = Main Ctrl Word	20.01 = Main Ctrl Word	Fieldbus interface as source for start and stop
EXT1 / EXT2 SEL	10.07 (HandAuto) MCW: Bit11 11.02 (Ref1Mux) MCW: Bit11 11.12 (Ref2Mux) Invert 11.02	10.07 (HandAuto) MCW: Bit11 11.02 (Ref1Mux) MCW: Bit11 11.12 (Ref2Mux) Invert 11.02	19.11 = MCW Bit11 (06.01)	Fieldbus interface as source to switch to EXT2 control place
REF1 SELECT	11.03 = SpeedRef2301	11.03 = SpeedRef2301	22.11 = FBA ref1	Fieldbus interface as source to speed reference
FAULT RESET SELECT	NA	NA	NA	Fieldbus interface as source for fault reset
PROFILE	NA	NA	51.02 = Drives Classic / Enhanced	Control profile to ABB Drives profile classic or enhanced

DrvControlModbusACS





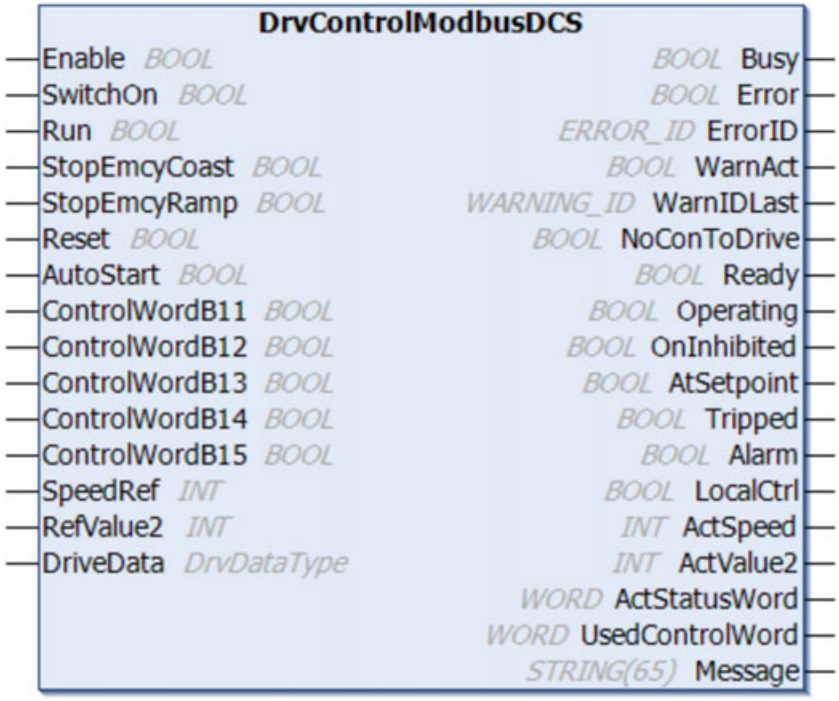
This function block can be used to control ACS drives with ABB Drives profile or ABB Drives enhanced profile using Modbus communication block like DrvModbusTcp or DrvModbusRtu.

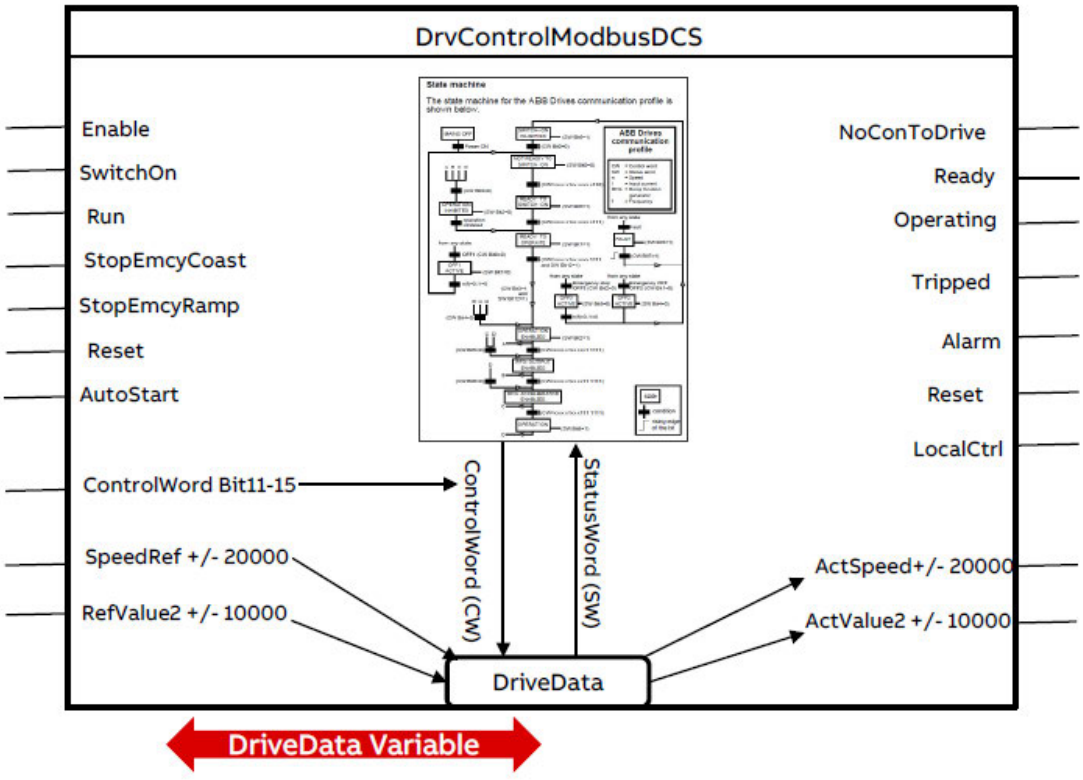
Status Word (SW) is read from drive through Modbus communication block using *“DriveData”* interface. *Chapter 1.5.4.2.3.3 “Structure: DrvDataType” on page 2207*

Control Word (CW) will be built by the function block according to the ABB drives profile state machine. CW will be sent via DriveData and the used communication block to the drive. Function block provides standard start/stop signals to control the drive and standard diagnosis signals are read from the drive.

The function block should be used for ACS drives using ABB drive (Classic/ Enhanced) profile for Modbus protocol only. The data transfer to the ACS drive is realized via the “IN_OUT” variable DriveData, which must be connected to “DrvModbusTcp” or “DrvModbusRtu” function block.

DrvControlModbusDCS





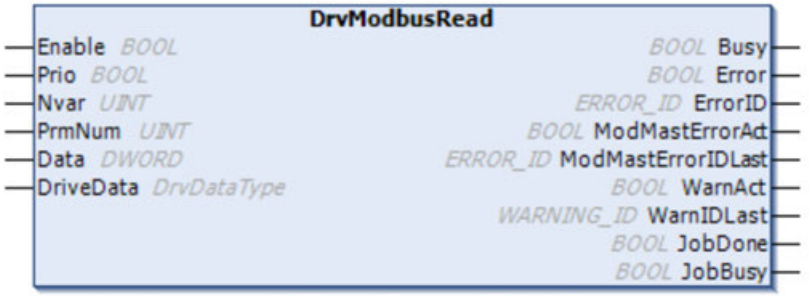
This function block can be used to control DCS drives with ABB Drives profile using Modbus communication block like *“DrvModbusTcp”* or *“DrvModbusRtu”*.

Status Word (SW) is read from drive through Modbus communication block using DriveData interface. *Chapter 1.5.4.2.3.3 “Structure: DrvDataType” on page 2207*

Control Word (CW) will be built by the function block according to the ABB drives profile state machine. CW will be sent via DriveData and the used communication block to the drive. Function block provides standard start/stop signals to control the drive and standard diagnosis signals are read from the drive.

The function block should be used for DCS drives using ABB drive profile for Modbus protocol only. The data transfer to the DCS drive is realized via the "IN_OUT" variable DriveData, which must be connected to "DrvModbusTcp" or "DrvModbusRtu" function block.

DrvModbusRead



The function block 'DrvModbusRead' reads one or more parameters / values of the drive. The number of data to be read is specified at the input 'Nvar'. The first parameter number is specified at the input 'PrmNum'. All parameters must be accessible from consecutive Modbus registers in the drive. The values of the parameters are stored in the PLC memory area, defined at the input 'Data'.

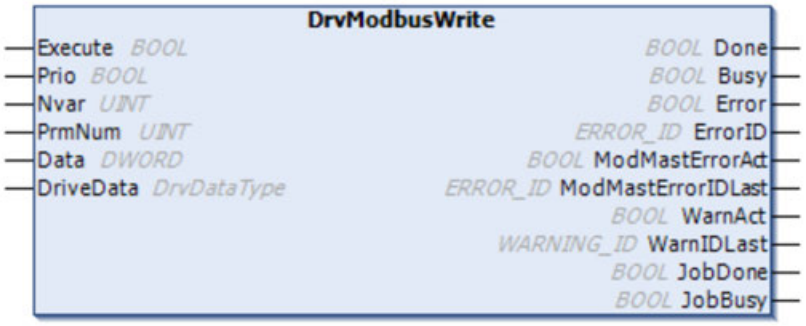
The values in the PLC memory area are updated when the read job was performed without error. This is indicated by JobDone = TRUE and ModMastErrorAct = FALSE.

If the Modbus job was finished with an error, the output ModMastErrorAct is set for one cycle. The Error ID returned by the Modbus job is shown at the output ModMastErrorIDLast. The output ModMastErrorIDLast will show that last Error ID until the input Enable is set from TRUE to FALSE.

As long as the Enable = TRUE a new read job is requested automatically one cycle after the further read job was terminated. The Modbus job is started from the Communication Block which is connected to the same 'DriveData' variable. It uses the Modbus function code 03 (read n words). The drive (Modbus device) from which the parameter is read is specified at this Communication Block. The Communication Blocks are available from the library e.g. DrvModbusTcp or DrvModbusRtu.

The function block is activated (Enable = TRUE) or deactivated (Enable = FALSE) via input Enable. If the block is active, the current values are available at the outputs. To start a new read job the input Enable must be set to TRUE. If the input values are valid, a request to perform a Modbus job is send to the Communication Block via the 'DriveData' variable. If at least 1 input is invalid, no job is generated, and the error is displayed at the outputs Error and ErrorID instead.

DrvModbusWrite



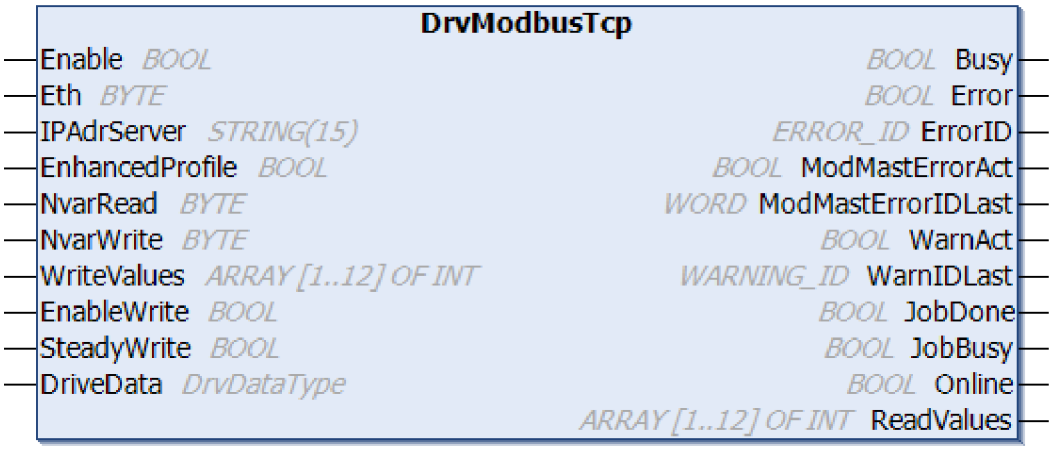
Function block 'DrvModbusWrite' writes 'n' parameters to the drive. The number of parameters to be written must be available in the PLC memory area, defined at the input Data. The write job has been performed without error if JobDone = TRUE and ModMastErrorAct = FALSE.

If the Modbus job was finished with an error, the output ModMastErrorAct is set for one cycle. The Error ID returned by the Modbus job is shown at the output ModMastErrorIDLast. The output ModMastErrorIDLast will show that last Error ID until the input Execute is set from TRUE to FALSE.

To start a new write job the input Execute must be set from FALSE to TRUE (edge sensitive). The Modbus job is started from the Communication Block which is connected to the same DriveData variable. It uses the Modbus function code 16 (write n words). The drive (Modbus device) to which the parameter is written is specified at the Communication Block.

> *Drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" must be set.*
>
> *Please see drive manuals for the parameter details.*

DrvModbusTcp



Function block DrvModbusTcp controls the Modbus TCP communication to ACS/DCS drives and provides the basic values (CW, Ref1, Ref2, SW, Act1, Act2) which are used for the basic control of drives with ABB Drives Profile or ABB Drives Enhanced Profile.

ABB drives classic profile

With input parameter EnhancedProfile = FALSE, the function block works for ABB Drives Classic Profile.

Reading status information from drives

The function block continuously reads data from the drive starting at Modbus register 400004. So at least the Status Word (SW), Actual Value 1 (Speed Reference), Actual Value 2 (Actual Value 2) are continuously read from the drive and written to the DriveData variable.

These values are stored in DriveData.StatusWord, DriveData.ActValue1 and DriveData.ActValue2.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	**Mapping configuration in drive**			**Written in AC500**	**Condition at function block**
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	**DCS550, DCS800**	**DCS880**		
Communication module	FENA-01/ 11/21 FMBT-21	RETA-01	FENA-01/11/21 FMBT-21		
400004	Status Word (SW)	Status Word (SW)	Status Word (SW)	DriveData.wStatusWord	Enable = TRUE
400005	Actual Value 1	Actual Value 1	Actual Value 1	DriveData.iActValue1	Enable = TRUE
400006	Actual Value 2	Actual Value 2	Actual Value 2	DriveData.iActValue2	Enable = TRUE

Writing control word and reference value to drives

To write the Control Word (CW), Reference Value 1 (Speed Reference) or Reference Value 2 (Reference Value 2) from the DriveData variable (DriveData.ControlWord, DriveData.Reference1, DriveData.Reference2) to the drive, the input EnableWrite has to be TRUE (default).

If the input SteadyWrite = TRUE (default = FALSE) these values are written steadily.

If the input SteadyWrite = FALSE (default) these values are only written if there was a change on any of those values.

These 3 values are written to the ACS drive starting at Modbus register 400001.

The function block checks if there are changes of the Control Word (wControlWord), Reference Value 1 (iRefValue1) or Reference Value 2 (iRefValue2) on the DriveData variable. If there is a change a write job is requested to send these 3 values to the ACS/DCS drive starting at Modbus register 400001.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive			Written from AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	**DCS550, DCS800**	**DCS880**		
Communication module	FENA-01/ 11/21 FMBT-21	RETA-01	FENA-01/11/21 FMBT-21		
400001	Control Word (CW)	Control Word (CW)	Control Word (CW)	DriveData.wControlWord	Enable = TRUE
400002	Reference Value 1	Reference Value 1	Reference Value 1	DriveData.iRefValue1	Enable = TRUE
400003	Reference Value 2	Reference Value 2	Reference Value 2	DriveData.iRefValue2	Enable = TRUE

ABB drives enhanced profile

With input parameter EnhancedProfile = TRUE, the function block works for ABB Drives Enhanced Profile.

Reading status information from drives

The function block continuously reads data from the drive starting at Modbus register 400051. So at least the Status Word (SW), Actual Value 1 (Speed Reference), Actual Value 2 (Actual Value 2) are continuously read from the drive and written to the DriveData variable.

These values are stored in DriveData.StatusWord, DriveData.ActValue1 and DriveData.ActValue2.

Apart from these three parameters there is also an option to read 12 additional drive parameters.

Using the input NvarRead the function block can be configured to read between 0 and 12 parameters from the drive. All read data is then written to the array at the ReadValue output array. Configuration in ACS drive is depending on configured parameters in group FBA DATA IN.

Modbus register address in drive	Mapping configuration in drive			Written in AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	**DCS550, DCS800**	**DCS880**		
Communication module	FENA-01/ 11/21 FMBT-21	RETA-01	FENA-01/11/21 FMBT-21		
400051	Status Word (SW)	Status Word (SW)	Status Word (SW)	DriveData.wStatusWord	Enable = TRUE
400052	Actual Value 1	Actual Value 1	Actual Value 1	DriveData.iActValue1	Enable = TRUE
400053	Actual Value 2	Actual Value 2	Actual Value 2	DriveData.iActValue2	Enable = TRUE
400054	FBA Data IN 1	FBA Data IN 1	FBA Data IN 1	ReadValues[1]	Enable = TRUE NVarRead >= 1
400055	FBA Data IN 2	FBA Data IN 2	FBA Data IN 2	ReadValues[2]	Enable = TRUE NVarRead >= 2
...	...	...	...	...	...
400064	FBA Data IN 11	FBA Data IN 11	FBA Data IN 11	ReadValues[11]	Enable = TRUE NVarRead >= 11
400065	FBA Data IN 12	FBA Data IN 12	FBA Data IN 12	ReadValues[12]	Enable = TRUE NVarRead >= 12

If 32-bit parameters are mapped to DATA IN,

- *The following field in DATA IN must be left open (= 0)*
- *The word order of the High-Word (HW) and Low-Word (LW) can be configured in the drive.*
  (using FENA-X1: Par. 51.22)
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from ReadValues fields must be recombined in the program.*

Function block DATA IN has to be configured in drive in the following groups see also FENA-x1 manual.

Drive	Parameter Group
ACS355	54.01 ... 54.10
ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	52.01 ... 52.12 52.01 ... 52.12 if installed as adapter A

Writing control word and reference value to drives

To write the Control Word (CW), Reference Value 1 (Speed Reference) or Reference Value 2 (Reference Value 2) from the DriveData variable (DriveData.ControlWord, DriveData.Reference1, DriveData.Reference2) to the drive, the input EnableWrite has to be TRUE (default).

If the input SteadyWrite = TRUE (default = FALSE) these values are written steadily.

If the input SteadyWrite = FALSE (default) these values are only written if there was a change on any of those values.

These 3 values are written to the ACS drive starting at Modbus register 400001.

Apart from these three there parameters there is also an option to write 12 additional drive parameters.

Using the input NvarWrite the function block can be configured to write between 0 and 12 parameters to the drive. The necessary values must be present in the array connected to WriteValues input. Configuration in ACS drive is depending on configured parameters in group FBA DATA OUT.

Modbus register address in drive	**Mapping configuration in drive**			**Written from AC500**	**Condition at function block**
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	**DCS550, DCS800**	**DCS880**		
Communication module	FENA-01/ 11/21 FMBT-21	RETA-01	FENA-01/11/21 FMBT-21		
400001	Control Word (CW)	Control Word (CW)	Control Word (CW)	DriveData.wControlWord	Enable = TRUE
400002	Reference Value 1	Reference Value 1	Reference Value 1	DriveData.iRefValue1	Enable = TRUE
400003	Reference Value 2	Reference Value 2	Reference Value 2	DriveData.iRefValue2	Enable = TRUE
400004	FBA Data OUT 1	FBA Data OUT 1	FBA Data OUT 1	WriteValues[1]	Enable = TRUE NVarWrite >= 1
400005	FBA Data OUT 2	FBA Data OUT 2	FBA Data OUT 2	WriteValues[2]	Enable = TRUE NVarWrite >= 2
...	...	...	...	...	...
400014	FBA Data OUT 11	FBA Data OUT 11	FBA Data OUT 11	WriteValues[11]	Enable = TRUE NVarWrite >= 11
400015	FBA Data OUT 12	FBA Data OUT 12	FBA Data OUT 12	WriteValues[12]	Enable = TRUE NVarWrite >= 12

If a Modbus TCP job tries to access a register in the drive which has no valid mapping information then job is aborted with an error.

Therefore, the drive parameters in FBA DATA IN group and FBA DATA OUT must be configured according to the used 'NvarRead' and 'NvarWrite' input number respectively.

If 32-bit parameters are mapped to DATA OUT,

- *The next/following field in DATA OUT must be left open (= 0)*
- *The word order of the High-Word (HW) and Low-Word (LW) can be configured in the drive.*
  (using FENA-X1: Par. 51.22)
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from WriteValues fields must be recombined in the program.*

ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" must be set.

Please see also drive manuals which parameter must be set.

For ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880 and DCS880 – Par 96.07 = 1

For ACSM1, DCS800 and DCS550 – Par 16.06 = 1

Diagnosis

If a Modbus TCP job tries to access a register in the drive which has no valid mapping information the job is aborted with an error.

The output ModMastErrorAct reflects that an actual error occurred. This output is only TRUE for one cycle. At that cycle the output ModMastErrorIDLast reflects the actual ErrorID from the ModTcpMast job. The ModMastErrorIDLast will keep this Error ID until a new rising edge of the Enable is given.

However, there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance.

These additional diagnosis variables can be accessed by opening the function block instance or through the block visualization “VisuDrvModbusTcp”.

- iWriteErrCnt: number of errors in write jobs since Enable = TRUE.
- wLastWriteErno: holds the error number of the last executed write job.
- iReadErrCnt: number of errors in read jobs since Enable = TRUE.
- wLastReadErno: holds the error number of the last executed read job.
- iReadWriteErrCnt: number of errors in read write jobs since Enable = TRUE.
- wLastReadWriteErno: holds the error number of the last executed read write job.

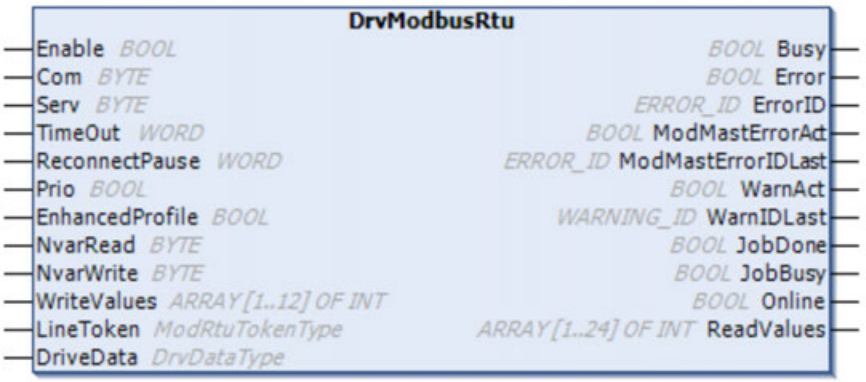
If the user changes drive profile while drive is online with PLC, function block outputs may give wrong indication.

Drive parameter settings

Settings in the drive according to AC500 configuration	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1**	**DCS550, DCS800**	**DCS880**
Communication module	FENA-01 /11/21 FMBT-21	RETA-01	FENA- 01/11/21 FMBT-21
Fieldbus activation = EXT FBA / ENABLE	50.01	98.02	50.01
FBA A Comm loss func	50.02	30.35	50.02
Comm Rate = Auto (0)	51.03	51.02	51.03
IP configuration = Static IP	51.04	51.03	51.04
IP address1 … IP address2	51.05 … 51.08	51.04 … 51.07	51.05 … 51.08
Subnet CIDR = 24 (eg: 255.255.255.0)	51.09	51.08 ... 51.11	51.09
Gateway Address (normally = 0.0.0.0)	51.10 … 51.13	51.12 … 51.15	51.10 … 51.13
Protocol / Profile = MB/TCP ABB E or MB/TCP ABB C	51.02	51.16	51.02
Word order for 32-bit parameter	51.22	No 32-bit access	51.22
Modbus timeout. Depending on timeout mode. Value in 100 ms	51.20	51.17	51.20
Modbus timeout mode: If input “SteadyWrite” is false set to “Any message“ If input “SteadyWrite” is true can also be set to “Control RW“	51.21		51.21
Refresh settings in drive	51.27	51.27	51.27

- *Please refer the respective drive / fieldbus module manual for the parameter settings if the drive setting is not mentioned in above table.*
- *For RETA-01/-02 IP address could also be set via hardware Dip-Switches.*
  If any switch is set (192.168.0.xxx) with xxx = Dip-Switches setting
- *ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" must be set.*
  Please see also drive manuals which parameter must be set.
  For ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880 and DCS880 – Par 96.07 = 1
  For ACSM1, DCS800 and DCS550 – Par 16.06 = 1

DrvModbusRtu



Function block DrvModbusRtu controls the Modbus RTU communication to ACS/DCS drives and is used for the basic control of drives with ABB Drives Profile or ABB Drives Enhanced Profile.

ABB drives classic profile

With input parameter EnhancedProfile = FALSE, the function block works for ABB Drives Classic Profile.

Reading status information from drives

The function block continuously reads data from the drive starting at Modbus register 400004. So at least the Status Word (wStatusWord), Actual Value 1 (iActValue1), Actual Value 2 (iActValue2) are continuously read from the drive and written to the DriveData variable.

These values are stored in DriveData.wStatusWord, DriveData.iActValue1 and DriveData.iActValue2.

With input NvarRead the function block can be configured to read in the same job between 0 ... 24 data more from the drive. These additional data are written to the array at the ‘ReadValues’ output. These data must be configured in the drive and are only accessible if the embedded Modbus is used.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive			Written in AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	**DCS550, DCS800**	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880,DCS880**		
Communication module	FSCA-01	RMBA-01	Embedded fieldbus		
400004	Status Word (SW)	Status Word (SW)	Status Word (SW) 58.104 = 4	DriveData.wStatusWord	Enable = TRUE
400005	Actual Value 1	Actual Value 1	Actual Value 1 58.105 = 5	DriveData.iActValue1	Enable = TRUE
400006	Actual Value 2	Actual Value 2	Actual Value 2 58.106 = 6	DriveData.iActValue2	Enable = TRUE
400007	-	-	58.107 DATA I/O 7	ReadValues[1]	Enable = TRUE
400008	-	-	58.108 DATA I/O 8	ReadValues[2]	Enable = TRUE
...	...	...	...	...	...
400014	-	-	58.114 DATA I/O 14	ReadValues[8]	Enable = TRUE
...	...	...	...	...	...
400024	-	-	58.124 DATA I/O 24	ReadValues[18]	Enable = TRUE

More details on the limits for the data read and write is explained in ↳ Chapter 1.5.4.2.4 “Limits for the data read and write between AC500 and drives” on page 2208. The value is dependent on the Drive used.

Writing control word and reference value to drives

The function block checks if there are changes of the Control Word (wControlWord), Reference Value 1 (iRefValue1) or Reference Value 2 (iRefValue2) on the DriveData variable. If there is a change a write job is requested to send these 3 values to the ACS/DCS drive starting at Modbus register 400001.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive			Written from AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1DCS880 DCS880	**DCS550, DCS800**	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880DCS880**		
Communication module	FSCA-01	RMBA-01	Embedded fieldbus		
400004	Control Word (CW)	Control Word (CW)	Control Word (CW) 58.101 = 1	DriveData.wControlWord	Enable = TRUE
400005	Reference Value 1	Reference Value 1	Reference Value 1 58.102 = 2	DriveData.iRefValue1	Enable = TRUE
400006	Reference Value 2	Reference Value 2	Reference Value 2 58.103 = 3	DriveData.iRefValue2	Enable = TRUE

More details on the limits for the data read and write is explained in ↳ Chapter 1.5.4.2.4 “Limits for the data read and write between AC500 and drives” on page 2208. The value is dependent on the Drive used.

ABB drives enhanced profile

With input parameter EnhancedProfile = TRUE, the function block works for ABB Drives Enhanced Profile.

The ABB Drives Profile Enhanced communication profile provides register mapped access to the Control, Status, Reference and Actual Values of the ABB Drives Profile Enhanced. The mapping of the registers has been enhanced to allow additional writing of up to 12 control and reading of up to 12 additional status parameters in a single Modbus job.

Reading status information from drives

The function block continuously reads data from the drive starting at Modbus register 400051. So at least the Status Word (wStatusWord), Actual Value 1 (iActValue1), Actual Value 2 (iActValue2) are continuously read from the drive and written to the DriveData variable.

These values are stored in DriveData.wStatusWord, DriveData.iActValue1 and DriveData.iActValue2.

Apart from these three parameters there is also an option to read 12 additional drive parameters in the same job.

Using the input NvarRead the function block can be configured to read between 1 and 12 more parameters from the drive. All read data is then written to the array at the ReadValues output. Configuration in ACS drive is depending on configured parameters in group FBA DATA IN.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive			Written in AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1	**DCS550, DCS800**	**DCS880**		
Communication module	FSCA-01	RMBA-01	FSCA-01		
400051	Status Word (SW)	Status Word (SW)	Status Word (SW)	DriveData.wStatusWord	Enable = TRUE
400052	Actual Value 1	Actual Value 1	Actual Value 1	DriveData.iActValue1	Enable = TRUE
400053	Actual Value 2	Actual Value 2	Actual Value 2	DriveData.iActValue2	Enable = TRUE
400054	FBA Data IN 1	FBA Data IN 1	FBA Data IN 1	ReadValues[1]	Enable = TRUE NVarRead >= 1
400055	FBA Data IN 2	FBA Data IN 2	FBA Data IN 2	ReadValues[2]	Enable = TRUE NVarRead >= 2
...	...	...	...	...	...
400064	FBA Data IN 11	FBA Data IN 11	FBA Data IN 11	ReadValues[11]	Enable = TRUE NVarRead >= 11
400065	FBA Data IN 12	FBA Data IN 12	FBA Data IN 12	ReadValues[12]	Enable = TRUE NVarRead >= 12

Writing control word and reference value to drives

The function block checks if there are changes in any of the following values since last write job:

- Control Word (wControlWord),
- Reference Value 1 (iRefValue1),
- Reference Value 2 (iRefValue2) on the DriveData variable,
- values in the input array WriteValues – WriteValues[1..NvarWrite].

If there is a change a write job is requested to send the 3 control values and the values in WriteValues array (WriteValues[1..NvarWrite]) to the ACS/DCS drive starting at Modbus register 400001. Configuration in ACS drive is depending on configured parameters in group FBA DATA OUT.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive			Written from AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1 DCS880	**DCS550, DCS800**	**DCS880**		
Communication module	FSCA-01	RMBA-01	FSCA-01		
400001	Control Word (CW)	Control Word (CW)	Control Word (CW)	DriveData.wControlWord	Enable = TRUE
400002	Reference Value 1	Reference Value 1	Reference Value 1	DriveData.iRefValue1	Enable = TRUE
400003	Reference Value 2	Reference Value 2	Reference Value 2	DriveData.iRefValue2	Enable = TRUE
400004	FBA Data OUT 1	FBA Data OUT 1	FBA Data OUT 1	WriteValues[1]	Enable = TRUE NVarWrite >= 1
400005	FBA Data OUT 2	FBA Data OUT 2	FBA Data OUT 2	WriteValues[2]	Enable = TRUE NVarWrite >= 2
...	...	...	...	...	...
400014	FBA Data OUT 11	FBA Data OUT 11	FBA Data OUT 11	WriteValues[11]	Enable = TRUE NVarWrite >= 11
400015	FBA Data OUT 12	FBA Data OUT 12	FBA Data OUT 12	WriteValues[12]	Enable = TRUE NVarWrite >= 12

Reconnection pause

When one or more drives in the Modbus RTU lines are offline, all the other drives have to wait for the TimeOut to elapse until a line token is assigned to next drive. Reconnection pause input helps in skipping the drives which are offline from the next Modbus job and execute Modbus job operations only for the drives which are online.

“ReconnectPause” is time in seconds before next retry to connect after a timeout was detected. Timeout is detected with ModMastErrorIDLast = 16#120 (ERR_TIMEOUT).

This feature can be used with the DrvModbusRtu function block in both ABB Drives Profile and ABB Drives Enhanced Profile. User must configure the reconnect pause input value using the input variable “ReconnectPause”.

For the generic RTU block ModRtuToken (part of AC500_ModbusRtu library), also the value for the reconnect pause must be configured at input variable “ReconnectPause”.

Diagnosis

The output ErrorID which reflects an actual error number is only valid for one cycle if output Error is set to TRUE. To capture this error number an external function must be programmed.

The output ModMastErrorAct reflects that an actual error occurred. This output is only TRUE for one cycle. At that cycle the output ModMastErrorIDLast reflects the actual ErrorID from the ModRtuMast job. The ModMastErrorIDLast will keep this error ID until a new rising edge of the Enable input is given.

However, there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance.

These additional diagnosis variables can be accessed by opening the function block instance or through the block visualization “VisuDrvModbusRTU”.

- iWriteErrCnt: number of errors in write jobs since Enable = TRUE.
- wLastWriteErno: holds the error number of the last executed write job.
- iReadErrCnt: number of errors in read jobs since Enable = TRUE.
- wLastReadErno: holds the error number of the last executed read job.
- iReadWriteErrCnt: number of errors in read write jobs since Enable = TRUE.
- wLastReadWriteErno: holds the error number of the last executed read write job.

If several drives are used, for each drive a communication function block such as DrvModbusRtu must be programmed. Also, every other generic Modbus server device on the same Modbus RTU line must be programmed with its own ModRtuToken function block. All those communication function blocks of one Modbus RTU line must be linked together via one variable of type ModRtuTokenType, connected to the InOut LineToken. Via this variable the Modbus token is passed to the next drive/device, so only one drive/device at a time is communicating with the PLC.

ModRtuToken function block and ModRtuTokenType structure are part of AC500_ModbusRtu library. Kindly refer the same.

If the user changes drive profile while drive is online with PLC, function block outputs may give wrong indication.

If a Modbus RTU job tries to access a register in the drive which has no valid mapping information then the job is aborted with an error.

Therefore, the drive parameters in FBA DATA IN group and FBA DATA OUT must be configured according to the used ‘NvarRead’ and ‘NvarWrite’ input number respectively.

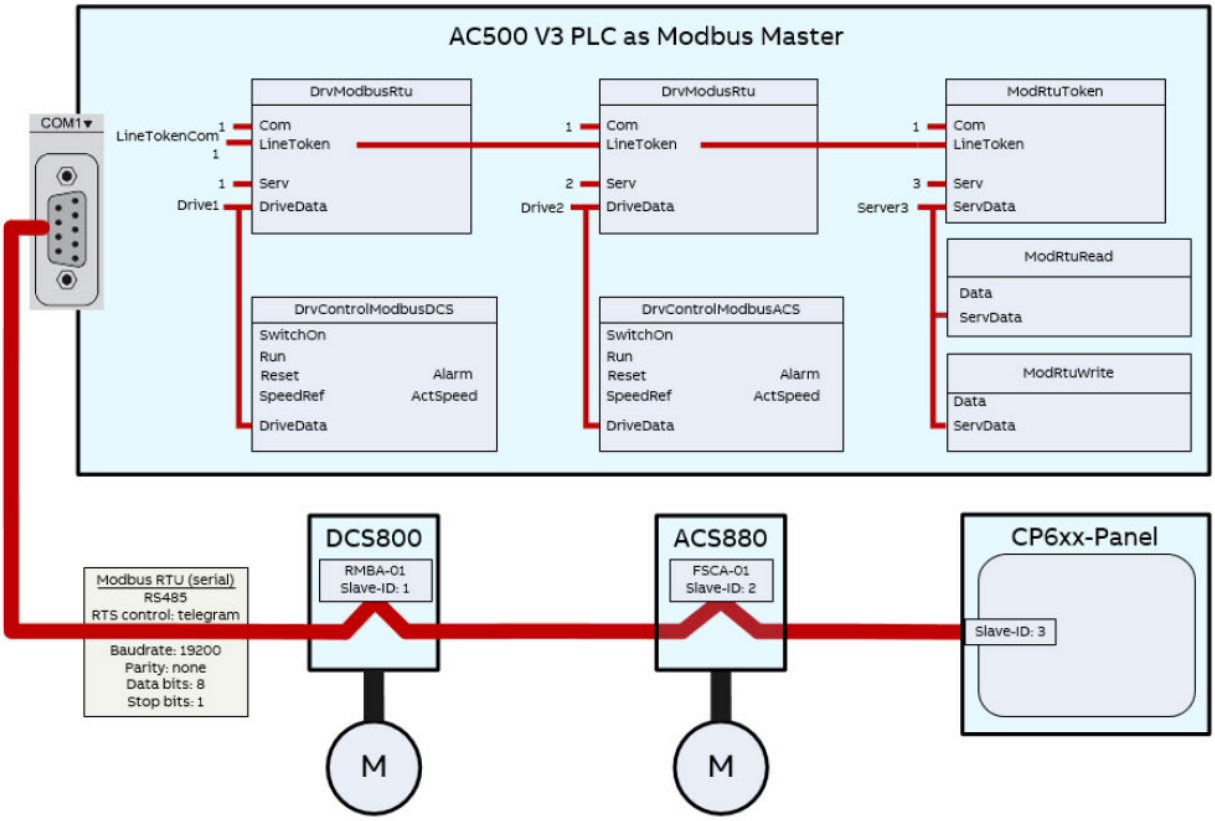
Modbus RTU using Embedded Fieldbus:

When embedded fieldbus is used for the Modbus RTU communication, user can read maximum of 24 parameters (based on the limitation in drive) from the DATA I/O parameters in the embedded fieldbus parameter group. These parameters can only be used for reading operation and cannot be configured to write data.

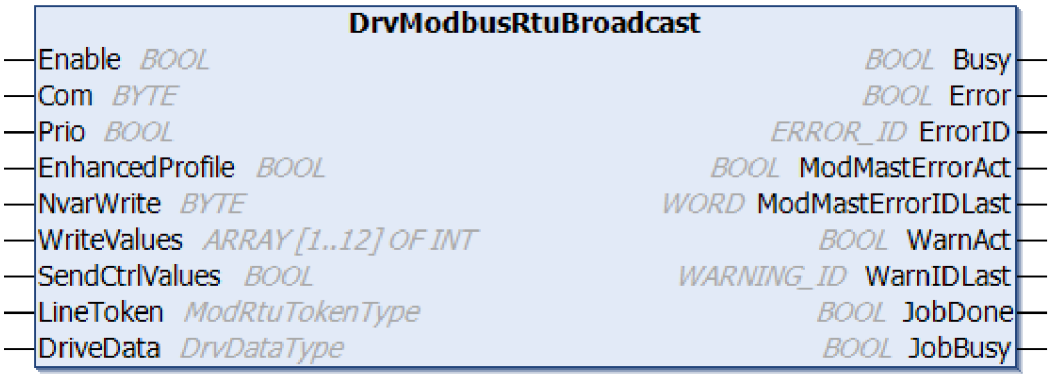
Drive parameter settings

Settings in the drive according to AC500 configuration	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1, DCS880**	**DCS550, DCS800**	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, DCS880**
Communication module	FSCA-01	RMBA-01	Embedded fieldbus
Fieldbus activation = EXT FBA / ENABLE	50.01	98.02	58.01
FBA A Comm loss func	50.02	30.35	50.02
Slave number	51.03	51.02	58.03
Transmission rate	51.04	51.03	58.04
Parity	51.05	51.05	58.05
Protocol / Profile = ABB Classic/ABB Enhanced	51.02	51.16	51.02
Word order for 32-bit parameter	51.22	No 32-bit access	51.22
Mapping of control word, Mod-bus reg 400001	Fix	Fix	58.101
Mapping of refer-ence value 1, Modbus reg 400002	Fix	Fix	58.102
Mapping of refer-ence value 2, Modbus reg 400003	Fix	Fix	58.103
Mapping of status word, Modbus reg 400004	Fix	Fix	58.104
Mapping of actual value 1, Modbus reg 400005	Fix	Fix	58.105
Mapping of actual value 2, Modbus reg 400006	Fix	Fix	58.106
Timeout mode = None (0) or Any Message (1), but not Ctrl write (2) as these values are only written after changes	51.07		58.15
Modbus timeout. Depending on timeout mode. Value in 100 ms		58.17	58.16
Refresh settings in drive	51.27	51.27	58.06

- *Please refer the respective drive / fieldbus module manual for the parameter settings if the drive setting is not mentioned in above table.*
- *ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" must be set.*
  Please see also drive manuals which parameter must be set.
  For ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880 and DCS880 – Par 96.07 = 1
  For ACSM1, DCS800 and DCS550 – Par 16.06 = 1



DrvModbusRtuBroadcast



Function block DrvModbusRtuBroadcast is a communication block which sends the broadcast messages via the Modbus RTU communication to all ACS/DCS drives and other Modbus devices connected to the same Modbus RTU line (physical line). The function block can be used with all drives with either ABB Drives Profile or ABB Drives Enhanced Profile but not a mix of both profiles.

As the broadcast job will be received by all devices on the same physical Modbus line it´s highly recommended to use this block only in case there are no other Modbus devices connected to this line and all drives use the same profile.

This function block does not perform any Modbus read operation, hence it does not read any values such as status word, actual value 1 and actual value 2 etc., from any of the drive.

This function block should not be used along with 'DrvModbusRead' and 'DrvModbusReadWrite23' function blocks. They will be ignored showing an error. This function block should be only used independently or in combination with 'DrvModbusWrite' function block for broadcasting write operation.

A successful broadcast message for writing control word, reference values and additional mapped parameters (only in case of Enhanced Profile) is indicated by JobDone = TRUE and ModMastErrorAct = FALSE. A next broadcast job for writing these values can once again started with a fresh rising edge at ‘SendCtrlValues’ input.

Apart from sending control values and up to 12 additional values from WriteValues array (only in case of ABB Drives Enhanced Profile) a normal Modbus write function block “DrvModbusWrite” can be used to send broadcast write messages to specific address on all drives connected to the Modbus RTU line. The requests to process broadcast write Modbus jobs is transferred via the DriveData structure at the InOut variable DriveData which can be connected to multiple instances of write function block 'DrvModbusWrite'.

After each successful broadcast write job a fixed pause of 250 ms is implemented before any other Modbus job within the same line will be started.

ABB drives classic profile

With input parameter EnhancedProfile = FALSE, the function block works for ABB Drives Classic Profile.

Writing control word and reference value to drives

A rising edge from FALSE to TRUE at input 'SendCtrlValues' starts sending broadcast message with Control Word and Reference Values to all the drives starting at Modbus register 400001.

Following control values: Control Word (wControlWord), Reference Value 1 (iRefValue1) or Reference Value 2 (iRefValue2) are taken from DriveData variable for sending broadcast message.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive			Written from AC500	Condition at function block and input SendCtrlvalues
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1DCS880 DCS880	**DCS550, DCS800**	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880DCS880 DCS880**		
Communication module	FSCA-01	RMBA-01	Embedded fieldbus		
400001	Control Word (CW)	Control Word (CW)	Control Word (CW) 58.101 = 1	DriveData.wControlWord	Enable = TRUE and Rising Edge at SendCtrlvalues
400002	Reference Value 1	Reference Value 1	Reference Value 1 58.102 = 2	DriveData.iRefValue1	Enable = TRUE and Rising Edge at SendCtrlvalues
400003	Reference Value 2	Reference Value 2	Reference Value 2 58.103 = 3	DriveData.iRefValue2	Enable = TRUE and Rising Edge at SendCtrlvalues

ABB drives enhanced profile

With input parameter EnhancedProfile = TRUE, the function block works for ABB Drives Enhanced Profile.

With the ABB Drives Profile Enhanced profile, along with 3 control values Control Word , Reference Value 1, Reference Value 2 , up to 12 additional values can be sent as broadcast message in a single Modbus job.

Writing control word and reference values to drives

A rising edge from FALSE to TRUE at input 'SendCtrlValues' starts sending broadcast message with Control Word and reference values to all the drives starting at Modbus register 400001.

Following control values: Control Word (wControlWord), Reference Value 1 (iRefValue1) or Reference Value 2 (iRefValue2) from DriveData along with values in the input array WriteValues – WriteValues[1..NvarWrite] are taken for sending broadcast message.

For the additional 12 values the configuration in ACS drive is depending on configured parameters in group FBA DATA OUT.

The following table shows the performed Modbus broadcast write job and the needed mapping in the drive as well as the area where the data is taken from the AC500.

Modbus register address in drive	Mapping configuration in drive			Written from AC500	Condition at function block
	ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1 DCS880	**DCS550, DCS800**	**DCS880**		
Communication module	FSCA-01	RMBA-01	FSCA-01		
400001	Control Word (CW)	Control Word (CW)	Control Word (CW)	DriveData.wControlWord	Enable = TRUE
400002	Reference Value 1	Reference Value 1	Reference Value 1	DriveData.iRefValue1	Enable = TRUE
400003	Reference Value 2	Reference Value 2	Reference Value 2	DriveData.iRefValue2	Enable = TRUE
400004	FBA Data OUT 1	FBA Data OUT 1	FBA Data OUT 1	WriteValues[1]	Enable = TRUE NVarWrite >= 1
400005	FBA Data OUT 2	FBA Data OUT 2	FBA Data OUT 2	WriteValues[2]	Enable = TRUE NVarWrite >= 2
...	...	...	...	...	...
400014	FBA Data OUT 11	FBA Data OUT 11	FBA Data OUT 11	WriteValues[11]	Enable = TRUE NVarWrite >= 11
400015	FBA Data OUT 12	FBA Data OUT 12	FBA Data OUT 12	WriteValues[12]	Enable = TRUE NVarWrite >= 12

Diagnosis

The output ErrorID which reflects an actual error number is only valid for one cycle if output Error is set to TRUE. To capture this error number an external function must be programmed.

The output ModMastErrorAct reflects an actual error occurred in Modbus job. This output is only TRUE for one cycle. At that cycle the output ModMastErrorIDLast reflects the actual ErrorID from the ModRtuMast job. The ModMastErrorIDLast will keep this error ID until a new rising edge of the Enable input is given.

However, there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance.

These additional diagnosis variables can be accessed by opening the function block instance or through the block visualization “VisuDrvModbusRTUBroadcast”.

- iWriteErrCnt: number of errors in write jobs since Enable = TRUE.
- wLastWriteErno: holds the error number of the last executed write job.

For all drives, which are connected to same Modbus RTU line, one instance of broadcast block DrvModbusRtuBroadcast is enough and it must be connected to same LineToken of DrvModbusRtu function blocks which are used for communication between PLC and each drive on Modbus RTU line. All those communication function blocks of one Modbus RTU line must be linked together via one variable of type ModRtuTokenType, connected to the InOut LineToken. Via this variable the Modbus token is passed to the next drive / device, so only one drive / device at a time is communicating with the PLC.

All the drives should be configured either in Classic Profile or Enhanced Profile and accordingly the function block DrvModbusRtuBroadcast should be parameterized. Mix of profile with few drives in Classic and few drives in Enhanced should not be used when using DrvModbusRtuBroadcast block, if using such configuration along with DrvModbusRtuBroadcast may lead to incorrect operation.

If the user changes drive profile while drive is online with PLC, function block outputs may give wrong indication.

The Modbus RTU broadcast job is sent to all devices on the same physical Modbus RTU line.

*Therefore, if other Modbus devices than ACS / DCS drives are connected to the same line using the ModRtuToken communication block it´s highly recommended **not** to use the DrvModbusRtuBroadcast function block.*

This might only be used, if the user is aware about the behavior of the connected devices if they receive the Modbus broadcast job.

If a Modbus RTU broadcast job is sent to access a register in the drive which has no valid mapping information then Modbus broadcast job is not aborted but will just send out the broadcast message without any error in the function block. This broadcast message is ignored by drives which have no valid mapping information.

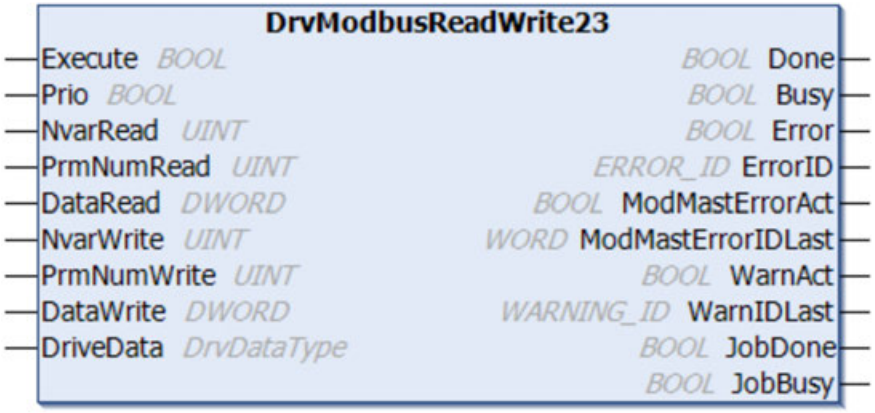
Therefore, the drive parameters in FBA DATA OUT have to be configured according to the used 'NvarWrite' input number respectively.

Drive parameter settings

Settings in the drive according to AC500 configuration	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, ACSM1, DCS880**	**DCS550, DCS800**	**ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880, DCS880**
Communication module	FSCA-01	RMBA-01	Embedded fieldbus
Fieldbus activation = Modbus / RS-485 comm	50.01	98.02	58.01
FBA A Comm loss func	50.02	30.35	50.02
Slave number	51.03	51.02	58.03
Transmission rate	51.04	51.03	58.04
Parity	51.05	51.05	58.05
Protocol / Profile = ABB Classic / ABB Enhanced	51.02	51.16	
Mapping of control word, Modbus reg 400001	Fix	Fix	58.101
Mapping of reference value 1, Modbus reg 400002	Fix	Fix	58.102
Mapping of reference value 2, Modbus reg 400003	Fix	Fix	58.103
Mapping of status word, Modbus reg 400004	Fix	Fix	58.104
Mapping of actual value 1, Modbus reg 400005	Fix	Fix	58.105
Mapping of actual value 2, Modbus reg 400006	Fix	Fix	58.106
Timeout mode = None (0) or Any Message (1), but not Ctrl write (2) as these values are only written after changes	51.07		58.15
Modbus timeout. Depending on timeout mode. Value in 100 ms		58.17	58.16
Refresh settings in drive	51.27	51.27	58.06

- *Please refer the respective drive / fieldbus module manual for the parameter settings if the drive setting is not mentioned in above table.*
- *ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" must be set.*
  Please see also drive manuals which parameter must be set.
  For ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880 and DCS880 – Par 96.07 = 1
  For ACSM1, DCS800 and DCS550 – Par 16.06 = 1

DrvModbusReadWrite23



The function block 'DrvModbusReadWrite23' reads and writes one or more parameters of the drive via DriveData connected to Modbus TCP / Modbus RTU communication blocks with Modbus function code FCT = 23. This function block internally calls DrvModbusWrite to execute ReadWrite job with FCT = 23, used along with the internal structure for Fct23, DrvModFct23Type.

The number of parameters to be read is specified at the input 'NvarRead'. The first address for read operation is specified at the input 'PrmNumRead'. The values of the data are stored in the PLC memory area, defined at the input 'DataRead'.

The number of parameters to be written is specified at the input 'NvarWrite'. The first address for write operation is specified at the input 'PrmNumWrite'. The values of the data that should be written must be stored in the PLC memory area, defined at the input 'DataWrite'.

To start a new ReadWrite job the input Execute must be set from FALSE to TRUE (edge sensitive). The Modbus job is started from the communication block DrvModbusTcp or DrvModbusRtu which is connected to the same DriveData variable. It uses the Modbus function code 23 (Read and write n words). The drive (Modbus device) to which the parameter is written is specified at the Communication Block.

The values in the PLC memory area are updated when the ReadWrite job was performed without error. The ReadWrite job has been performed without error if JobDone = TRUE and ModMastErrorAct = FALSE.

If the Modbus job was finished with an error, the output ModMastErrorAct is set for one cycle. The Error ID returned by the Modbus job is shown at the output ModMastErrorIDLast. The output ModMastErrorIDLast will show that last Error ID until the input Execute is set from TRUE to FALSE.

After termination of this job, even if it was not successful, a next ReadWrite job can once again only be started with a rising edge at 'Execute' input.

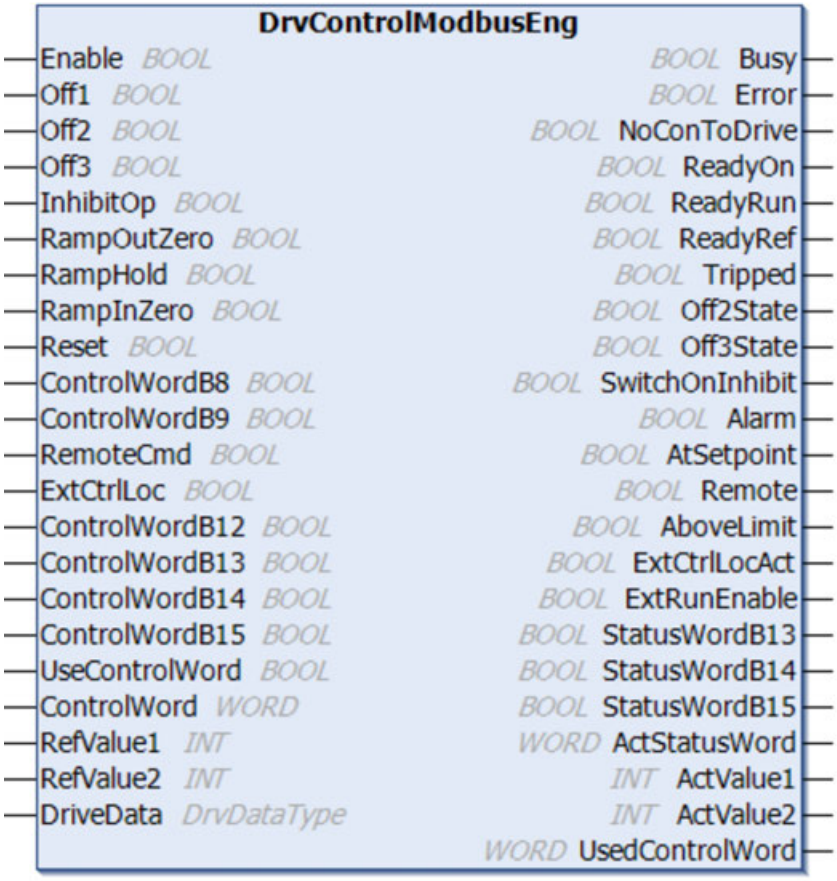
Drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" must be set.

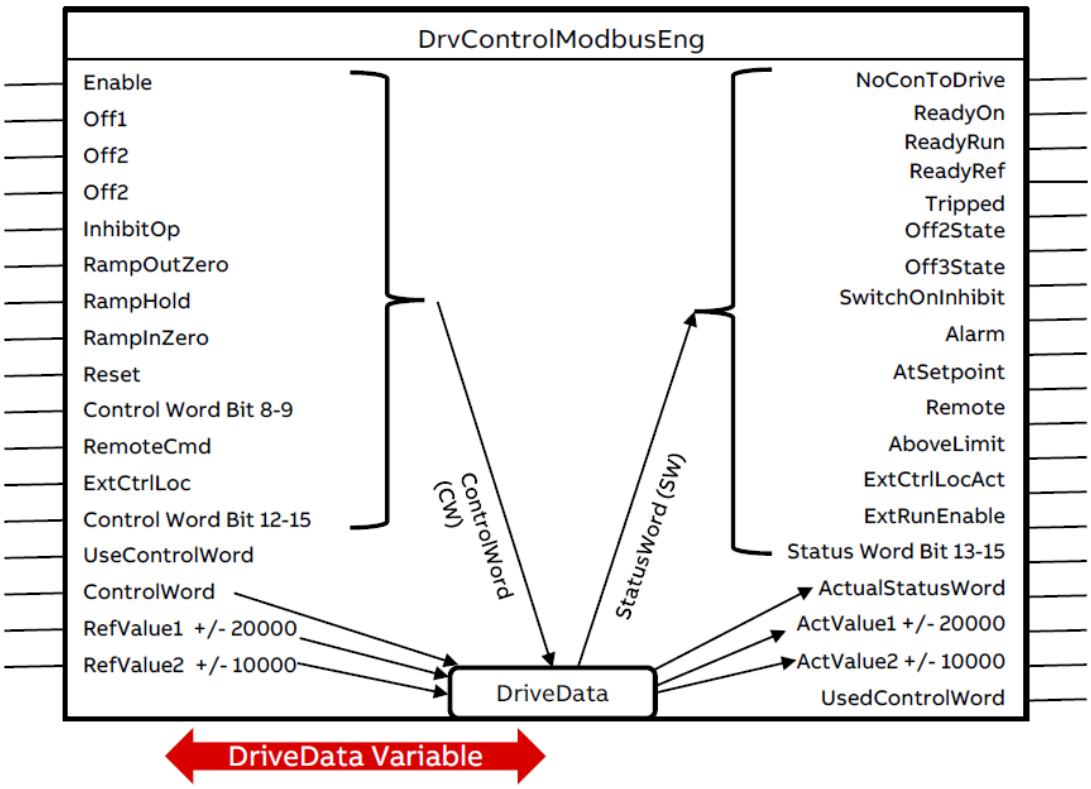
Please see drive manuals for the parameter details.

For ACS380, ACS480, ACS580, ACH580, ACQ580, ACS880 and DCS880 – Par 96.07 = 1

For ACSM1, DCS800 and DCS550 – Par 16.06 = 1

DrvControlModbusEng





The function block *“DrvControlModbusEng”* is designed for user specific control of the drive by setting the control word (CW) by the user itself in the program.

Therefore, the user should have a detailed knowledge of the ABB drives profile handling.

The reference and actual values must be given in fieldbus equivalent , e.g. range -20000 ... +20000.

Inputs *“RefValue1”*, *“RefValue2”* and the generated control word are written to the *“DriveData”* variable which transfers these values to a communication function block, e.g. *“DrvModbusRtu”*, *“DrvModbusTcp”* or *“DrvModbusRtuBroadcast”* communication function block writes to the drive. In the same way *“ActValue1”*, *“ActValue2”* and the status word are transferred from the communication function block to the *“DrvControlModbusEng”* block, where they are written to the outputs.

The control word can be generated in 2 ways.

First way is to set the single bits of the control word separately at the inputs *“Off1”*, *“Off2”* ... *“ControlWordB15”* while the input *“UseControlWord”* = FALSE.

Second way is to set the input *“UseControlWord”* = TRUE and write the control word as a whole word directly to the input control word. The generated control word is written to the *“DriveData”* variable and for diagnosis purpose also available at output *“UsedControlWord”*.

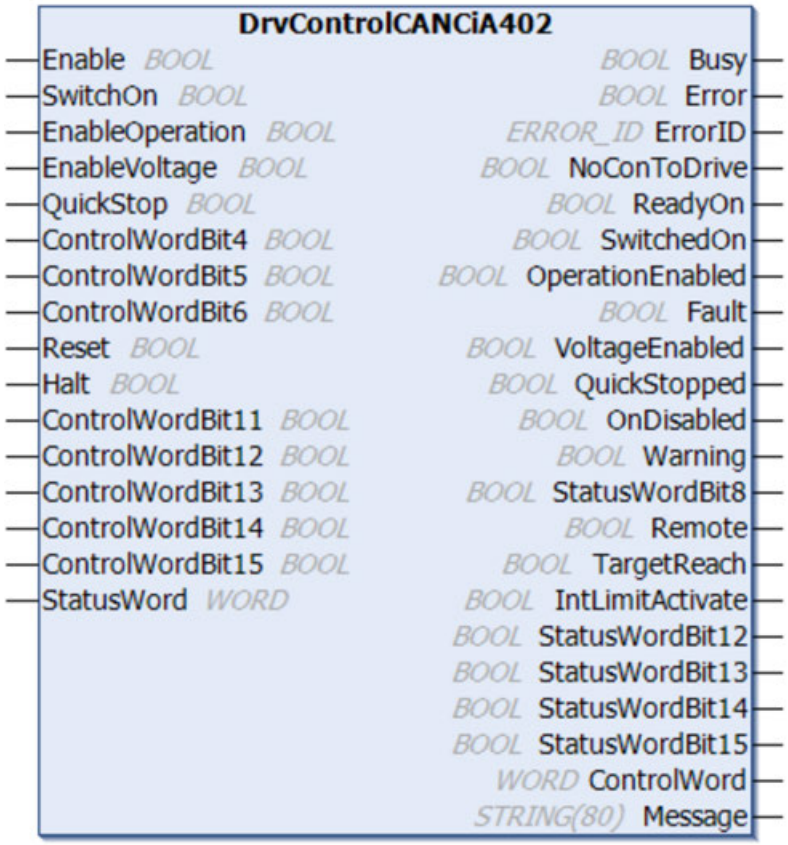
The input and output names of the bits in control word and status word reflect the functions used with ABB Drive Profile. So the block should be used with ABB Drives Profil setting in the drive.

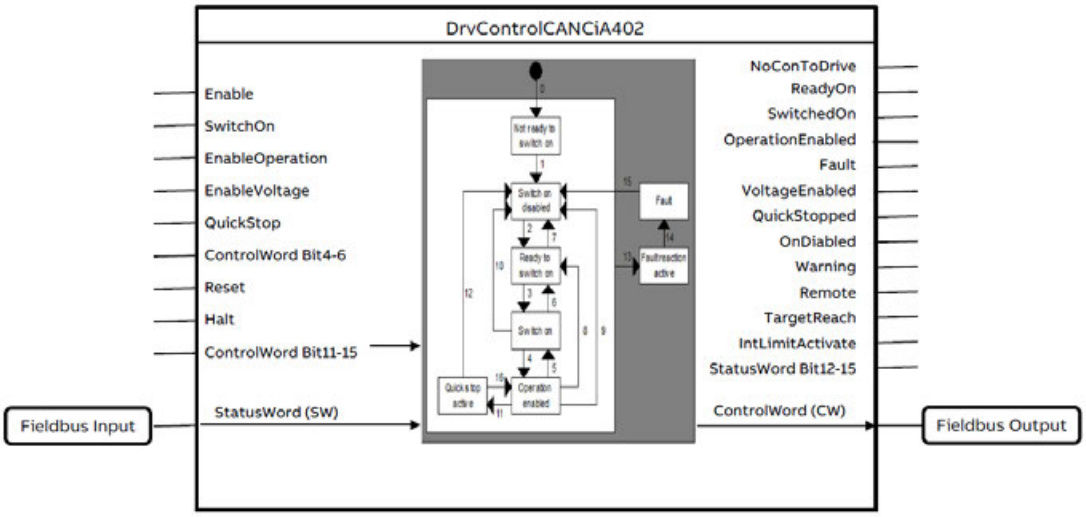
The function block does not execute any functionality expect data transfer to and from the “DriveData” variable. There is no special drive parameter setting necessary to use this block.

The programmer using this block should have a detailed understanding of how to set the control word according to the status word and the description of the used drive.

For standard speed and torque control application it is recommended to use the “DrvControlModbusACS” instead.

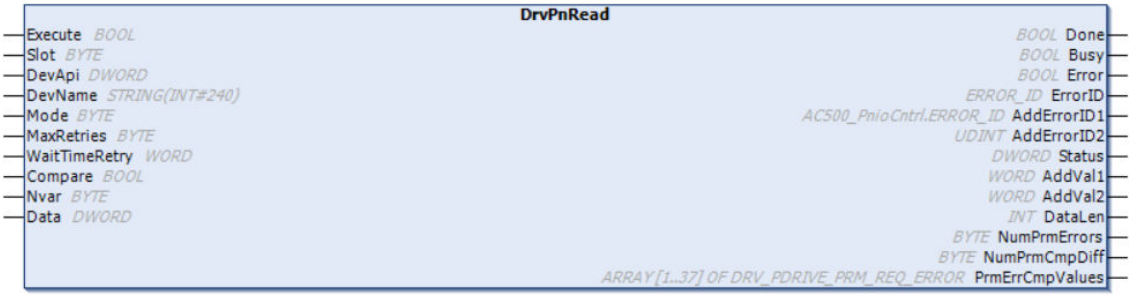
DrvControlCANCiA402





The function block *"DrvControlCANCiA402"* is used for the control of ABB ACS Drive or non-ABB drives from AC500 using CANCiA402 Profile . The CANopen CiA402 function block and visualization from the library can also be used for the 3rd party drives which comply to the CANopen CiA402 Profile.

DrvPNRead



Function block *"DrvPnRead"* reads maximum 37 parameters from the drive in a single DPV1 query. The number of parameters to be read is specified at the input Nvar.

Parameters to read from the drive are specified at the Data input. *"DrvPbPnPrmDpv1DataType"* structure must be declared to a variable and connected to Data input using ADR. This structure contains the group, index, which must be given to the variable. Read parameter type and values are stored in the same variable.

"DrvPdPrmDpv1DataType" structure has the following array elements:

Chapter 1.5.4.2.3.4 "Structure: DrvPdPrmDpv1DataType" on page 2208

- *"abyPrmGroup"*: Array of 37 WORD for specifying parameter group.
- *"abyPrmIndex"*: Array of 37 WORD for specifying parameter index.
- *"abyPrmType"*: Array of 37 DRV_PDRIVE_PRM_TYPE. READ parameter data type will be available here. For details refer to DRV_PDRIVE_PRM_TYPE. If a type is set here at the start, it can be compared with the type read from the drive if the compare input is TRUE.
- *"adwPrmValue"*: Array of 37 DWORD. Read parameter value will be available here. If a value is set at the start, it can be compared with the value read from the drive if the compare input is TRUE.

Read errors:

If the drive rejects to read a specific parameter, it returns the error code DRV_ERROR_PRM (16#44) in the corresponding abyPrmType element and a more specific error value in the corresponding adwPrmValue element.

The number of elements with errors from the drive are given at the output *"NumPrmErrors"*. The output *"PrmErrCmpValues"* gives an array, which contains the more specific error values in the elements (index). This can be used to quickly identify the erroneous elements.

Compare input:

As the *"DrvPnWrite"* function block does not return an error in case a parameter in the drive could not be written correctly it is recommended to verify the writing.

This can be done with the call of this function block *"DrvPnRead"* if the same parameters, types and values are connected to the DATA input (use the same struct as for the writing), and the input "Compare" is set to TRUE.

Then the types and values of the connected struct are copied at the rising edge of execute inputs and compared with the returned types and values from the drive.

In case of a difference this is set into the corresponding element of the output array *"PrmErrCmpValues"* with the possible three error codes DRV_CMP_DIFF_TYPE, DRV_CMP_DIFF_VALUE or DRV_CMP_DIFF_TYPE_AND_VALUE.

Mode input:

- Mode = 16#00 => Read direct variables and parameters via an Fxxx module, e.g. FENA-21 or FPNO-21. Group and Index have to be used as in the *"Data.awPrmGroup"* and *"Data.awPrmIndex"* array. (Number of Elements in the PN Data block is set to 16#01)
- Mode = 16#01 => Read direct variables and parameters via an Rxxx module, e.g. RETA-21 for ACS800 or DCS500. Group and Index have to be used as in the *"Data.awPrmGroup"* and *"Data.awPrmIndex"* array. (Number of Elements in the PN Data block is set to 16#01)
- Mode = 16#1x => to be used to access *"PROFIDrive"* parameters with Attribute = 16#10 (Value) and Number of Elements = x.

- Mode = 16#2x => to be used to access *"PROFIDrive"* parameters with Attribute = 16#20 (Description) and Number of Elements = x. (Not supported with Fxxx or Rxxx modules)
- Mode = 16#3x => to be used to access PROFIDrive parameters with Attribute = 16#30 (Text) and Number of Elements = x. (Not supported with Fxxx or Rxxx modules)

For *"PROFIDrive"* parameters using Mode = 16#1x, 16#2x or 16#3x the Number of Elements = x is used for all the parameters in the Data array.

DrvPnWrite

Function block *"DrvPnRead"* reads maximum 37 parameters from the drive in a single DPV1 query. The number of parameters to be written is specified at the input Nvar.

Another limit while using the *"DrvPnWrite"* function block is, it can process only up to 240-byte data in one request or 37 drive parameters whichever is lower. If the write data length is more than 240 bytes, the function block generates an error code `WRITE_PACKAGE_SIZE_TOO_LONG` 16#0004. At the output *"PackageSize"* the precalculated size of the request is shown.

Parameters to write to the drive are specified at the data input. *"DrvPdPrmDpv1DataType"* structure must be declared to a variable and connected to data input using ADR.

"DrvPdPrmDpv1DataType" structure has the following array elements:

↳ *Chapter 1.5.4.2.3.4 "Structure: DrvPdPrmDpv1DataType" on page 2208*

- *"abyPrmGroup"*: Array of 37 WORD for specifying parameter group.
- *"abyPrmIndex"*: Array of 37 WORD for specifying parameter index.
- *"abyPrmType"*: Array of 37 DRV_PDRIVE_PRM_TYPE for specifying parameter type, refer the respective drives manual for parameter data type and enter the respective enumeration. For details about enumeration refer DRV_PDRIVE_PRM_TYPE in the library.
- *"adwPrmValue"*: Array of 37 DWORD for specifying parameter value that should be written.

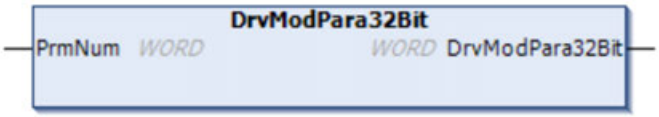
The values in the structure area are updated when the write job was performed without error. This is indicated by Done=TRUE.

Mode input:

- Mode = 16#00 => Write direct variables and parameters via an Fxxx module, e.g. FENA-21 or FPNO-21. Group and Index have to be used as in the *"Data.awPrmGroup"* and *"Data.awPrmIndex"* array. (Number of Elements in the PN Data block is set to 16#01)
- Mode = 16#01 => Write direct variables and parameters via an Rxxx module, e.g. RETA-21 for ACS800 or DCS500. Group and Index have to be used as in the *"Data.awPrmGroup"* and *"Data.awPrmIndex"* array. (Number of Elements in the PN Data block is set to 16#01)
- Mode = 16#1x => to be used to access *"PROFIDrive"* parameters with Attribute = 16#10 (Value) and Number of Elements = x.
- Mode = 16#2x => to be used to access *"PROFIDrive"* parameters with Attribute = 16#20 (Description) and Number of Elements = x. (Not supported with Fxxx or Rxxx modules)
- Mode = 16#3x => to be used to access PROFIDrive parameters with Attribute = 16#30 (Text) and Number of Elements = x. (Not supported with Fxxx or Rxxx modules)

For *"PROFIDrive"* parameters using Mode = 16#1x, 16#2x or 16#3x the Number of Elements = x is used for ALL the parameters in the Data array.

Function: DrvModPara32Bit



Creates the Modbus address for 32-bit parameters of the ACSxxx drives.

To access 32-bit parameters in ACSxxx drives using Modbus a special address calculation must be performed.

This block calculates the 6-digit address out of the 5-digit address used for 16-bit parameters.

Input is the 5-digit address: GGii, where GG = parameter group and ii = the index.

E.g. Par 12.02 ➨ address = 1202.

Output is the calculated address for 32-bit parameters according to the following rule:

DrvModPara32Bit = 20000 + (200 * GG) + (2 * ii) e.g. Par. 14.54 ➨ output = 22908

This output can be connected directly to the input "PrmNum" of one of the blocks DrvModbusRead or DrvModbusWrite or inputs “PrmNumRead” and “PrmNumWrite” of the block DrvModbusReadWrite23.

Structure: DrvDataType

Structure DrvDataType is used for the DriveData variable to exchange the data for one drive.

Structure DrvDataType is used for the DriveData variable which must be connected to all function blocks related to the same drive.

Besides the element “sName” all variables should not be written by the user directly. They are read and written within the function blocks. The DrvDataType contains some more internal, invisible variables which are used for internal functionality and not meant for user access.

The following table shows the visible variables of DrvDataType.

Variable	Data Type	Default value	Description
wStatusWord	WORD	0	Actual status word from drive
iActValue1	INT	0	Actual value1 from drive – mostly equal speed
iActValue2	INT	0	Actual value2 from drive – mapping is made in drive configuration
wControlWord	WORD	0	Control word to drive
iRefValue1	INT	0	Reference value1 to drive – mostly speed reference
iRefValue2	INT	0	Reference value2 to drive – mapping is made in drive configuration
xOnline	BOOL	FALSE	Connection established – set in Modbus communication function block after successful reading and writing one Modbus job
xCtrlBlockUsedf	BOOL	FALSE	A control block is used to generate the control word, ref1 and ref2 values
sName	STRING	‘Default Drive Name’	Name for drive, which can be set by user directly to DriveData variable

Structure: DrvPdPrmDpv1DataType

Structure *"DrvPdPrmDpv1DataType"* is required to exchange the data between AC500 and Drives using the PROFINET communication. *"DrvPdPrmDpv1DataType"* structure must be declared to a variable and connected to data input using ADR in the *"DrvPnRead"* or *"DrvPnWrite"* function blocks.

Variable	Data Type	Default value	Description
awPrmGroup	ARRAY [1..37] OF WORD	[37(0)]	ABB drive Group number from where the parameter to read or write - Profidrive Parameter Index.
awPrmIndex	ARRAY [1..37] OF WORD	[37(0)]	ABB drive index number from where the parameter to read or write - Profidrive Parameter Subindex.
abyPrmType	ARRAY [1..37] OFDRV_PDRIVE_PRM_TYPE	[37(DRV_EMPTY_PRM)]	Parameter Type in array. While using read block it will act as an output (and input at start of block with Compare=TRUE). While using write block it will act as an input.
adwPrmValue	ARRAY [1..37] OF DWORD	[37(0)]	Parameter Value in array. While using read block it will act as an output (and input at start of block with Compare=TRUE). While using write block it will act as an input.

1.5.4.2.4 Limits for the data read and write between AC500 and drives

The below table defines the limits for the reading of data from the drive and limits for writing data to drives from AC500 with cyclic data exchange.

If fieldbus adapter Plug (FBA) is used, then parameter group FBA DATA IN (e.g. 52) and group FBA DATA OUT (e.g. 53) is accessed in the drive. For the embedded fieldbus (EFB) parameters are used in EFB group (e.g. 58).

According to the table below, limits are defined for the variables 'NVarRead', 'NVarWrite' in DrvModbusRtu and DrvModbusTcp blocks.

Drive	Fieldbus Adapter (FBA)		Embedded Fieldbus (EFB)
	Data In (Group 52)	Data Out (Group 53)	Data I/O (group 58) Configurable as input or output.
ACS380	12	12	14
ACS480	12	12	14
ACS580	12	12	14
ACQ580	12	12	14
ACH580	12	12	14
ACS880	12	12	24
ACSM1	12	12	Not supported
DCS550			
DCS800			
DCS880	12	12	24

1.5.5 BACnet-BC

1.5.5.1 Introduction to BACnet

BACnet is a standardized data communication protocol for Building Automation and Control networks as defined in the ANSI/ASHRAE standard 135 and ISO 16484-5.

The advantage is interoperability between devices of different vendors.

The BACnet protocol defines services to allow communication between devices. Examples include 'Who is', 'I am', 'Who has' and 'I have' for device and object search and identification, *“Read Property”* and *“Write Property”* for the exchange of data, up to more complex services for alarm and event management, scheduling and trending.

The BACnet protocol defines a number of object types on which the services operate. Each object is characterized by its properties.

The BACnet objects are combined in a BACnet device. A BACnet device represents the functionality of a physical device.

More background information and introduction can be found here:

http://www.bacnet.org

http://www.bacnet.org/Bibliography

1.5.5.2 AC500 and BACnet

A BACnet device can be described by its *“BACnet Interoperability Building Blocks”* (BIBB)s, which are needed to establish services. They are grouped in different areas:

- *“Data Sharing”* (DS)
- *“Alarm and Event Management”*(AE)
- *“Scheduling”* (SCHED)
- *“Trending”* (T)
- *“Device and Network Management”* (DM)

“Data Sharing” for example contains two BIBBs which are needed for the *“Service Read Property”*:

- Client side: DS-RP-A (Data Sharing - Read Property - A)
- Server side: DS-RP-B (Data Sharing - Read Property - B)

The BACnet standard defines profiles by the minimum required BIBBs, see table below. *“BACnet Simple Sensor”* (B-SS) is the simplest one, only containing one BIBB. More complex devices contain more BIBBs (from right to left).

		CP600	AC500 V3		AC500 V2		
Interoperability Areas (IA)	**BACnet Device-profiles**						
	B-OWS	**B-OD**	**B-BC**	**B-AAC**	**B-ASC**	**B-SA**	**B-SS**
Data Sharing	DS-RP-A, B	DS-RP-A, B	DS-RP-A, B	DS-RP-B	DS-RP-B	DS-RP-B	DS-RP-B
	DS-RPM-A	DS-RPM-A, B	DS-RPM-A, B	DS-RPM-B			
	DS-WP-A	DS-WP-A, B	DS-WP-A, B	DS-WP-B	DS-WP-B	DS-WP-B	
	DS-WPM-A		DS-WPM-B	DS-WPM-B			
		DS-V-A					
		DS-M-A					
Alarm & Event Management	AE-N-A	AE-N-A	AE-N-I-B	AE-N-I-B			
	AE-ACK-A		AE-ACK-B	AE-ACK-B			
	AE-INFO-B		AE-INFO-B	AE-INFO-B			
	AE-ESUM-A		AE-ESUM-B				
		AE-VN-A					
Scheduling	SCHED-A		SCHED-E-B	SCHED-E-B			
Trending	T-VMT-A		T-VMT-I-B				
	T-ATR-A		T-ATR-B				
Device & Network Management	DM-DDB-A, B	DM-DDB-A, B	DM-DDB-A, B	DM-DDB-B	DM-DDB-B		
	DM-DOB-A, B	DM-DOB-B	DM-DOB-B	DM-DOB-B	DM-DOB-B		
	DM-DCC-A		DM-DCC-B	DM-DCC-B	DM-DCC-B		
	DM-TS-A		DM-TS-B or DM-UTC-B	DM-TS-B or DM-UTC-B			
	DM-UTC-A						
	DM-RD-A		DM-RD-B	DM-RD-B			
	DM-BR-A		DM-BR-B				

The AC500 V2 supports BIBBs qualifying it as *"BACnet Application Specific Controller"* (B-ASC), by installing the BACnet B-ASC library.

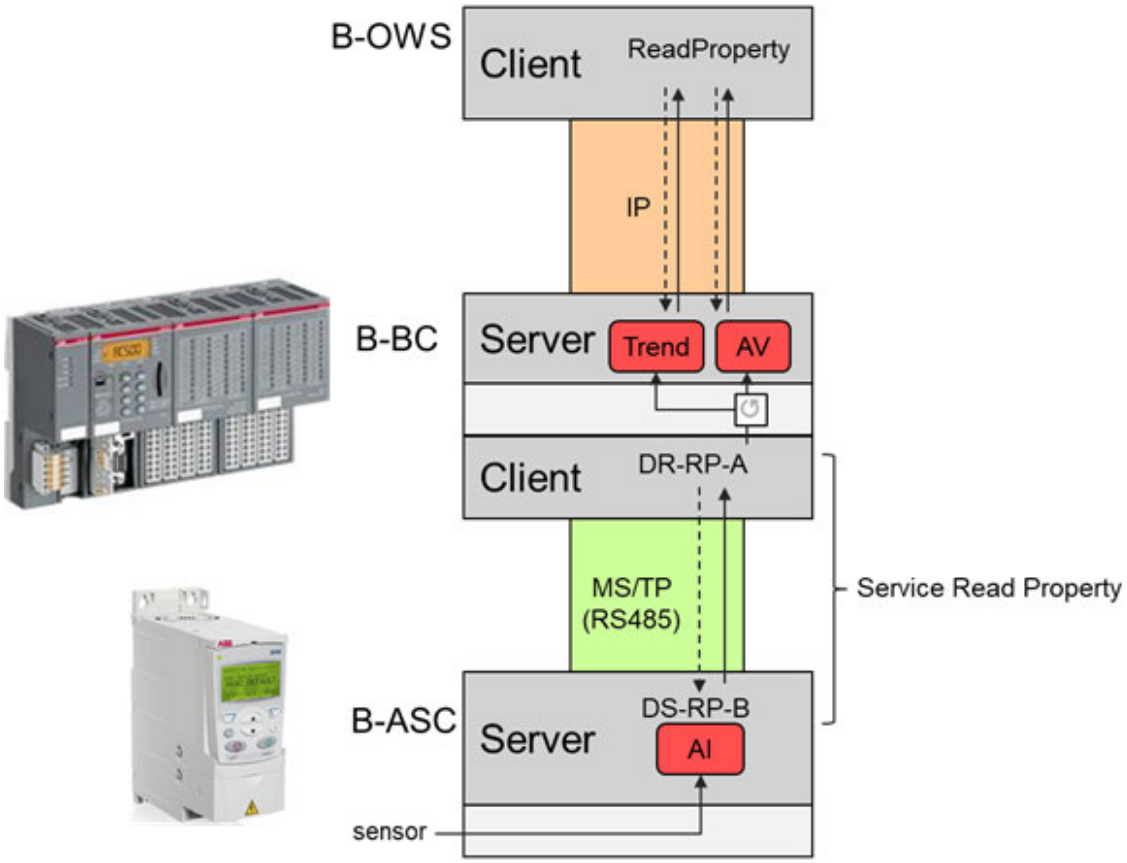
AC500 V3 supports many more BIBBs qualifying it as *"BACnet Building Controller"* (B-BC), which contains a server (all BIBBs ending with -B) and a client (all BIBBs ending with -A). In fact, the AC500 contains some more BIBBs. All BIBBs under B-BC in the table above, plus:

DS-COV-A, -B	(Change of Value-A, -B)
DS-COVP-A, -B	(Change of Value of Properties-A, -B)
AE-N-E-B	(Alarm and Event-Notification External-B)
AE-ASUM-B	(Alarm and Event-Alarm Summary-B)
SCHED-I-B	(Scheduling-Internal-B)
T-VMT-E-B	(Viewing and Modifying Trends External-B)
DM-TS-B	(Time Synchronization-B)
DM-UTC-B	(UTC Time Synchronization-B)
DM-MTS-A	(Manual Time Synchronization-A)
DM-LM-B	(List Manipulation-B)
DM-OCD-B	(Object Creation and Deletion-B)
NM-BBMDC-B	(BBMD Configuration-B)

...

A list with all details can be found in the Automation Builder pdf document ABB-B-BC-PICS-AC500_V3.pdf. Direction: Help/Project examples/Examples.

The figure below shows a typical application for an AC500 V3, acting as B-BC.



A drive with several actuators and sensors is acting as B-ASC, for example providing a temperature value as *"Analog Input"* (AI) object on the MS/TP network.

Chapter 1.5.5.3.1 "Supported BACnet networks " on page 2211

AC500 B-BC as client can read this temperature value, perform some processing (scaling, limit check) and on the server side provide the processed value as *"Analog Value"* (AV) object and as *"Trend"* object on the IP network. Higher level clients like BACnet Operator Workstation (B-OWS) can access the processed objects *"Analog Value"* and *"Trend"* for supervision.

The following chapters describe the possible applications and how to configure an AC500 V3 as B-BC.

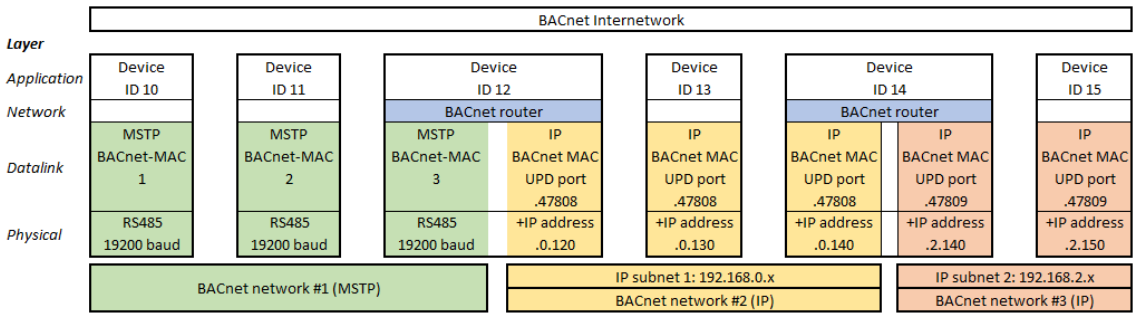
1.5.5.3 AC500 V3 as BACnet Building Controller (B-BC)

The BACnet integration into CODESYS implements the ANSI/ASHRAE standard 135-2012 (ISO 16484-5) protocol revision 14 and is based on the AMEV AS-A and AS-B standards. Integration allows access to the properties of BACnet objects and the configuration parameters of a BACnet device by means of an IEC application. You can program a dynamic BACnet configuration and have access to the BACnet functions in the BACnet network by reading and writing BACnet object properties.

1.5.5.3.1 Supported BACnet networks

BACnet can run on different local area network types. The AC500 B-BC supports the following ones:

- MS/TP (Master Slave / Token Passing), based on serial RS-485
- BACnet IP, based on Ethernet / UDP / IP



Different networks can be combined to one common *"BACnet internetwork"*. The figure above shows an example of some BACnet devices in one *"BACnet internetwork"*. Each device has a device ID (10 to 15) which must be unique on application level. Services on application level (e.g. read or write request) are working with these device IDs and need no addressing information of the lower levels.

The example *"BACnet internetwork"* consists of different BACnet networks:

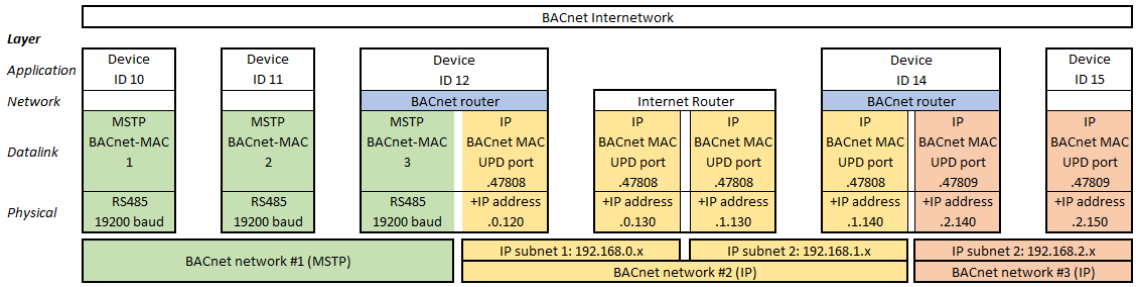
- BACnet MS/TP network connecting device 10, 11 and 12
- BACnet IP network (UDP port 47808), consisting of one IP subnets with IP range 192.168.0.x, connecting device 12, 13 and 14
- BACnet IP network (UDP port 47809), consisting of one IP subnet with IP range 192.168.2.x, connecting device 14 and 15

Addressing in a BACnet network is done through datalinks which must have a unique BACnet MAC address (which is different to an Ethernet MAC address).

- In a MS/TP network the BACnet MAC address is just one octet (1, 2, 3 in the example).
  ↳ *Chapter 1.5.5.3.4.4 "Configuration of datalinks " on page 2220*
- In an IP network the BACnet MAC address is the combination of the IP address and the UDP port number (for example 192.168.0.130.47808 for device 13). The following 16 UDP ports are reserved for BACnet: BAC0 (=47808 decimal) to BACF.
  ↳ *Chapter 1.5.5.3.4.4 "Configuration of datalinks " on page 2220*

To form a common *"BACnet internetwork"* the single BACnet networks must be combined by BACnet routers. AC500 can act as a BACnet router between BACnet MS/TP and IP networks (device 12 in the figure above) or between two different BACnet IP networks (device 14).

Two IP subnets using the same UDP ports can be combined to one BACnet IP network with an internet router.



The problem is that internet routers block local broadcast messages, which are required for BACnet communication. This can be solved by *"Broadcast Management Devices"* (BBDM). AC500 V3 can be configured as BBDM. In the figure above the devices 12 and 14 should be configured as BBDM in order to enable the BACnet communication across the internet router.

An alternative is to configure AC500 V3 as foreign BACnet device if an IP subnet contains no BBDM device to pass broadcast messages over internet routers.

Configuring the AC500 as BBDM or foreign device is described in ↳ *Chapter 1.5.5.3.4.4 "Configuration of datalinks " on page 2220*.

1.5.5.3.2 Supported objects and properties

Communication with BACnet is done through objects and properties.

The AC500 B-BC server of the figure below is represented as a BACnet device object with *"ID 12"*. The device contains more objects like the *Analog Input* object, representing the input of a temperature measurement device. An object contains several properties, like *"ID, Description, Present Value, Unit"* etc.

Further possible objects of an AC500 B-BC are:

- *"Binary Input"* for example from connected to a switch
- *"Analog / Binary Output"* for actuators
- *"Analog / Binary Values"* for local variables
- *"Calender"*
- *"Schedule"*
- *"Trend Log"*
- ...
- A list with all details can be found in the Automation Builder pdf document ABB-B-BC-PICS-AC500_V3.pdf. Help/Project examples/.

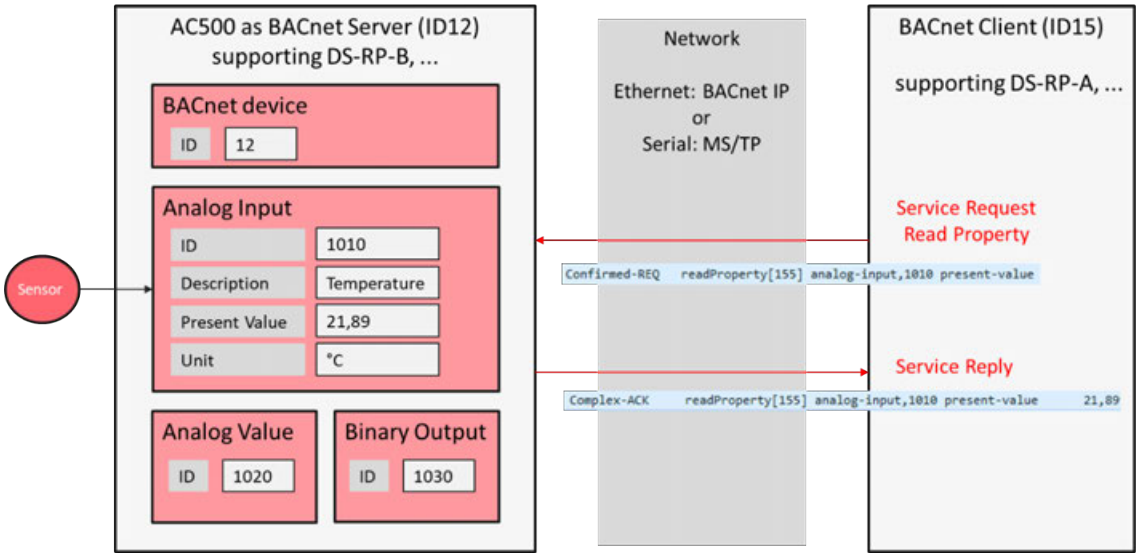


Fig. 25: BACnet objects, properties, services and BIBBs

1.5.5.3.3 Supported BIBBs and services

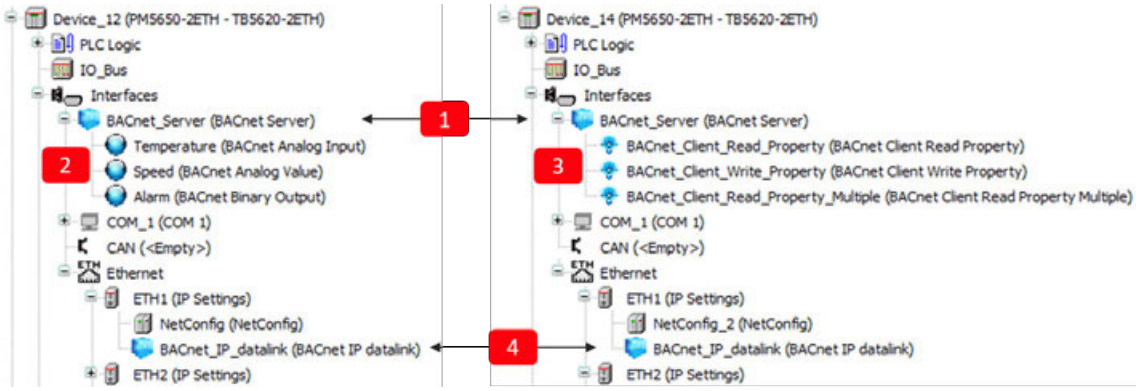
While objects and properties describe which data are communicated, the communication itself is done with services between clients and servers. A certain service can only be executed if client and server have the related BIBBs. The Fig. 25 *BACnet objects, properties, services and BIBBs* shows a simple *"Service Read Property"* which is possible because the client on the right supports DS-RP-A and the server on the left supports DS-RP-B. The service is executed in two steps:

1. The client initiates a confirmed request *"Read Property"*, asking for the present value of the *"Analog Input"* of object with *"ID 1010"*.
2. The server answers with an acknowledge, sending the present value which is 21,89°C in the example.

A list of all supported BIBBs and services of AC500 V3 is given in the Automation Builder pdf document ABB-B-BC-PICS-AC500_V3.pdf. Help/Project examples/Examples.

1.5.5.3.4 BACnet configuration in Automation Builder

To act as a BACnet server or client, the AC500 must be configured accordingly. The figure below shows the basic configuration of a BACnet server (left) and a BACnet server with client functionality (right). It is also possible to have server and client functionality in parallel.



Following objects need to be created:

1 *"BACnet Server"* root object. This is the root object for the server functionality, as well as for the client functionality. It is mandatory, even if only client functionality is required. ↳ *Chapter 1.5.5.3.4.1 "Configuration of BACnet server root object " on page 2214*

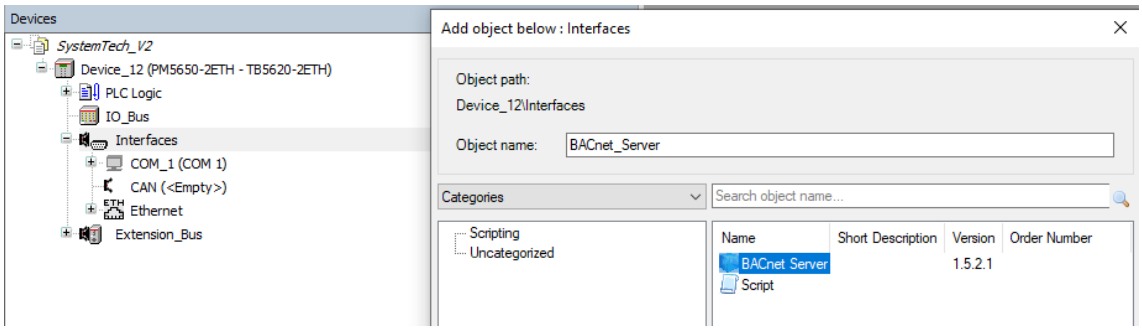
2 BACnet server objects, for example *"BACnet Analog Input"* `Temperature`. The properties of the objects must be controlled (written or read) by the PLC logic. ↳ *Chapter 1.5.5.3.4.2 "Adding BACnet server objects" on page 2216*

3 BACnet client objects, represented by a different symbol. For example, *"BACnet Client Read Property"*. The functionality of the client objects must be programmed in the PLC logic. Inserting the client objects below the server is optional. It is also possible to instantiate the objects only in a PLC logic. ↳ *Chapter 1.5.5.3.4.3 "Adding BACnet client functionality" on page 2217*

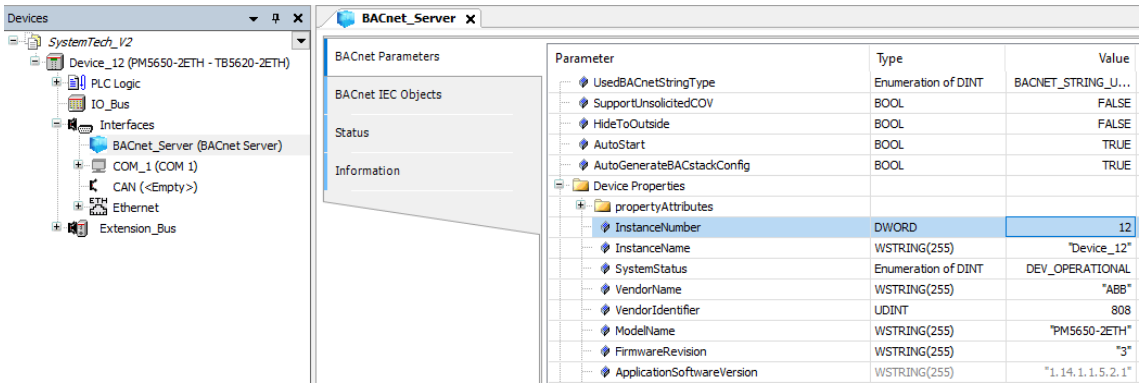
4 Datalink for the physical layer. This object links the physical interface (Ethernet IP or serial MS/TP) to the *"BACnet Server"* object. In the example above the IP address of `ETH1` is automatically retrieved by inserting the *"BACnet IP datalink"* below the `ETH1` port. ↳ *"Configuration of an IP datalink" on page 2221*. For MS/TP refer to ↳ *"Configuration of an MS/TP datalink" on page 2220*.

Configuration of BACnet server root object

1. Create an empty project with an AC500 V3 CPU type and call it fpr example *"Device_12"*.
2. Insert a *"BACnet Server"* object below the interfaces object in the device tree.

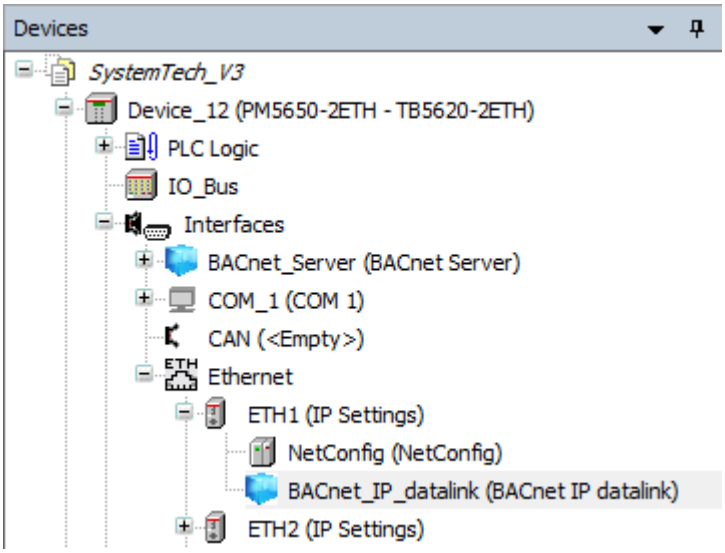


3. Set the device `InstanceNumber` in the *"BACnet Parameters"* of the *"BACnet Server"*, e.g. to `12` and the `InstanceName` to `Device_12` (according to Fig. 25 *BACnet objects, properties, services and BIBBs*).



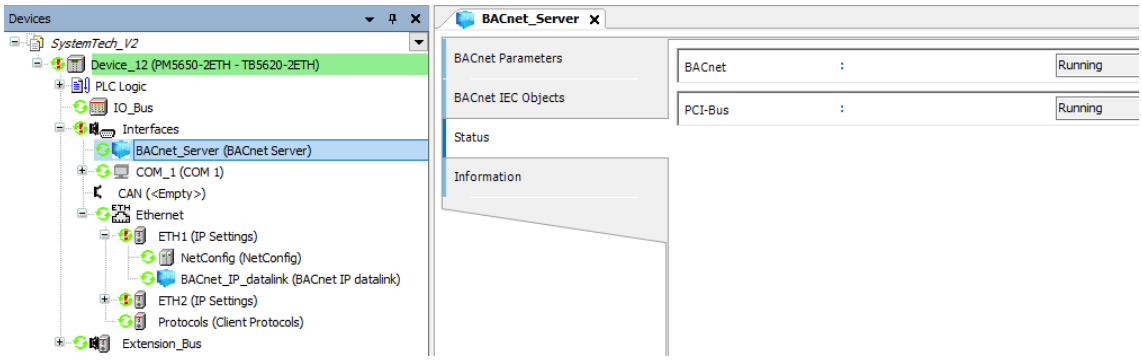
4. Add a datalink, IP or MS/TP. In the example an IP datalink is inserted below `ETH1`. Default parameters are sufficient if only one datalink is used.

↳ *"Configuration of an IP datalink" on page 2221*

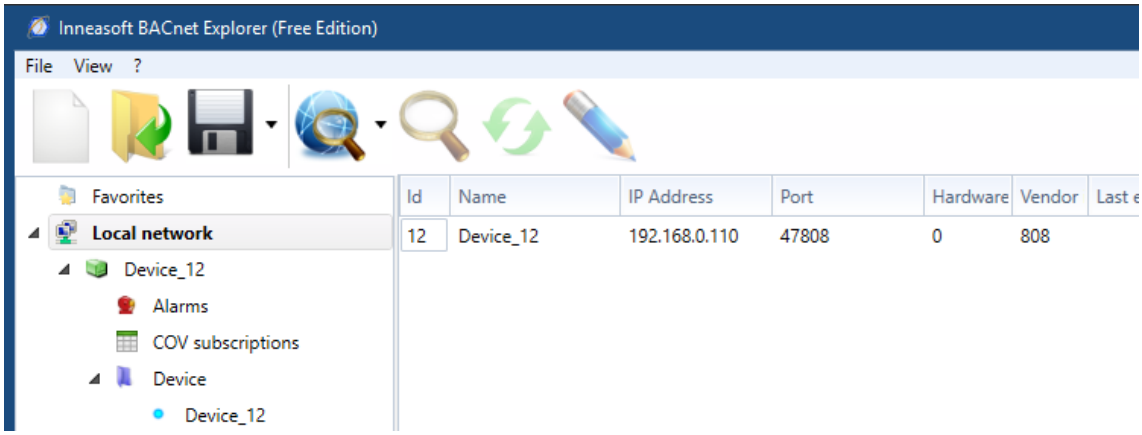


5. Build the project, download to the PLC and set it to *[RUN]*. The status of the *"BACnet Server"* should be green (running). If not, please ensure that you have installed the runtime license BACnet Protocol B-BC Runtime, verifiable by right-click on the PLC node and select *[Show license information]* from the runtime licensing menu. The project is scanned for required licenses. If you are logged in to a PLC, then the licenses available on the PLC are displayed. A missing required license is highlighted.

↳ *Chapter 1.6.6.2.2.2 "PLC runtime licensing" on page 3665*



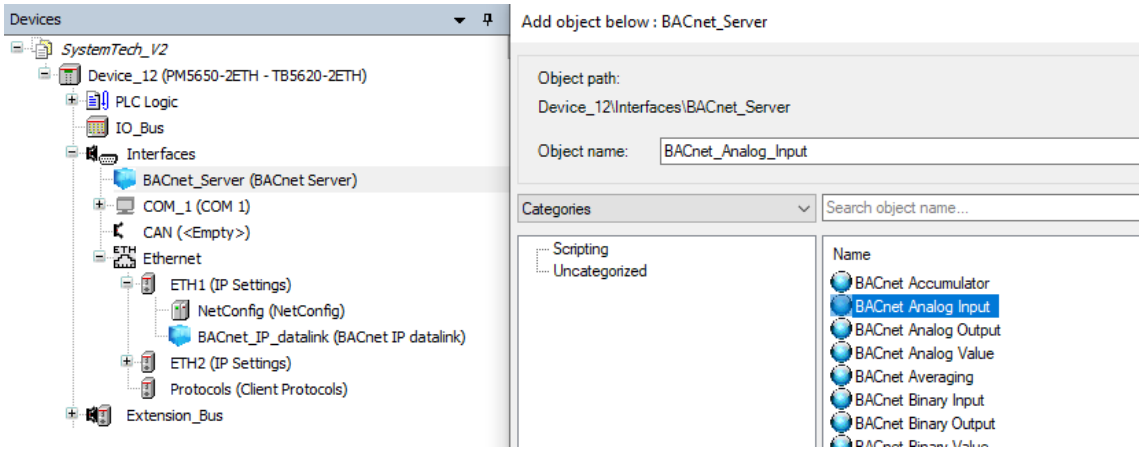
6. Start any BACnet client to find the server, for example Inneasoft BACnet Explorer.



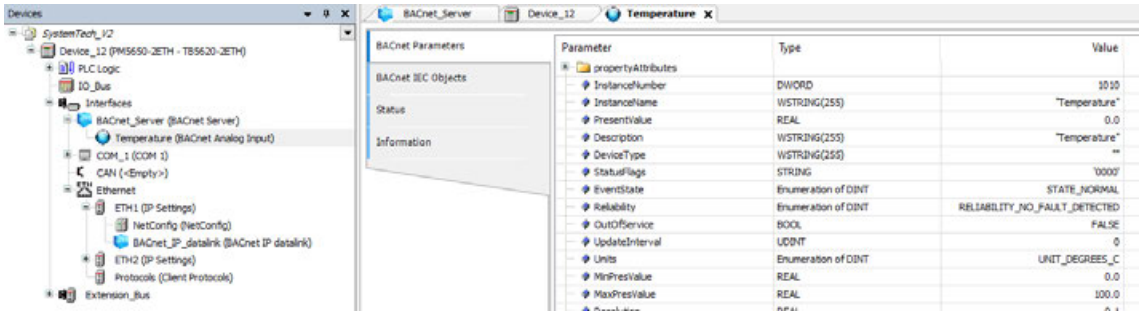
Adding BACnet server objects

Goal is to publish an analog value as BACnet server object. This example is according to Fig. 25 *BACnet objects, properties, services and BIBBs*, left part containing a temperature value.

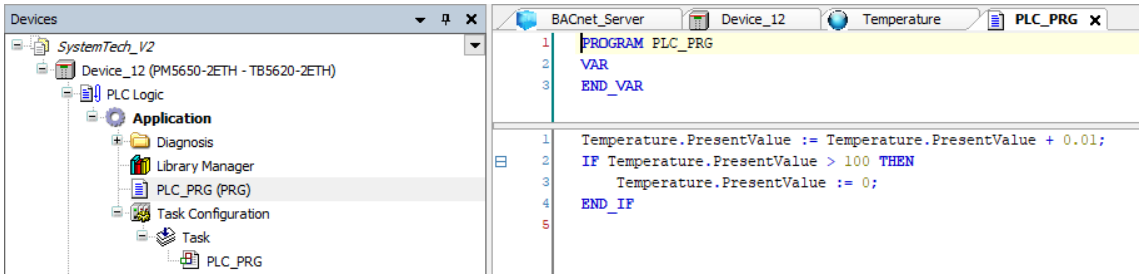
1. Configure a *"BACnet Server"* root object according to ↬ *Chapter 1.5.5.3.4.1 "Configuration of BACnet server root object " on page 2214*.
2. Add a *"BACnet Analog Input"* object below the *"BACnet Server"*.



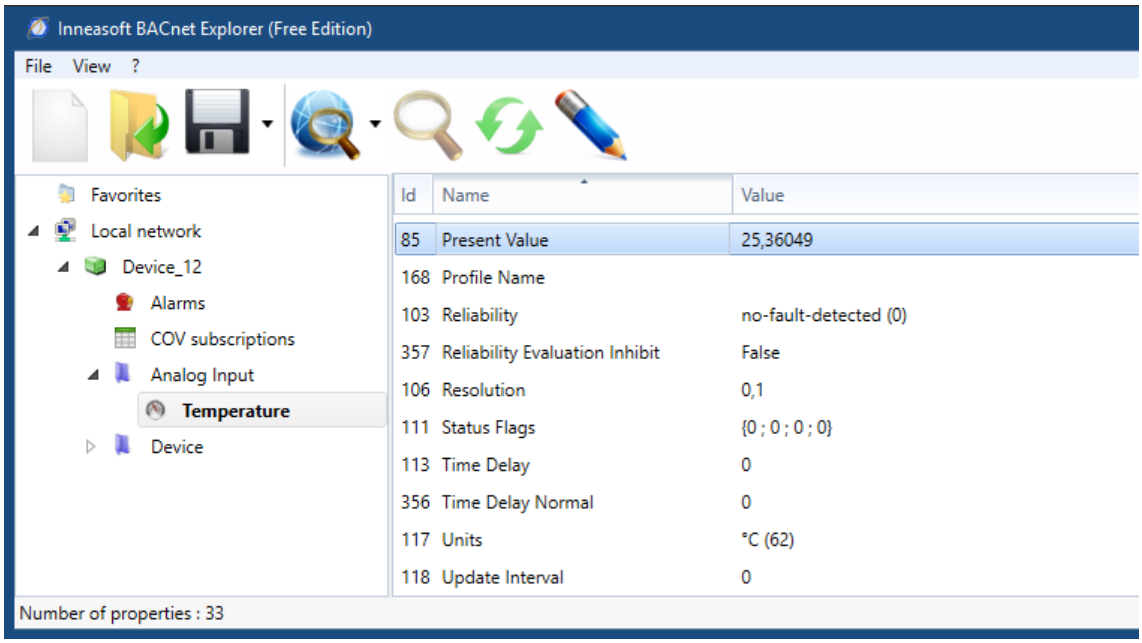
3. Rename it to `Temperature`, adjust the parameters: `InstanceNumber`: `1010`, `Description`: `Temperature`, `Units`: `UNIT_DEGREES_C`.



4. The present value of the objects `Temperature` needs to be fed with the value from the real temperature device. Alternatively, a simple PLC program can simulate this value.



5. Download the program and observe the temperature value in the BACnet client.

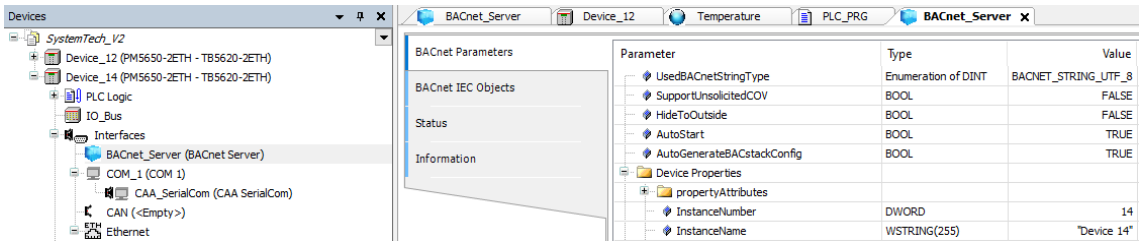


Adding BACnet client functionality

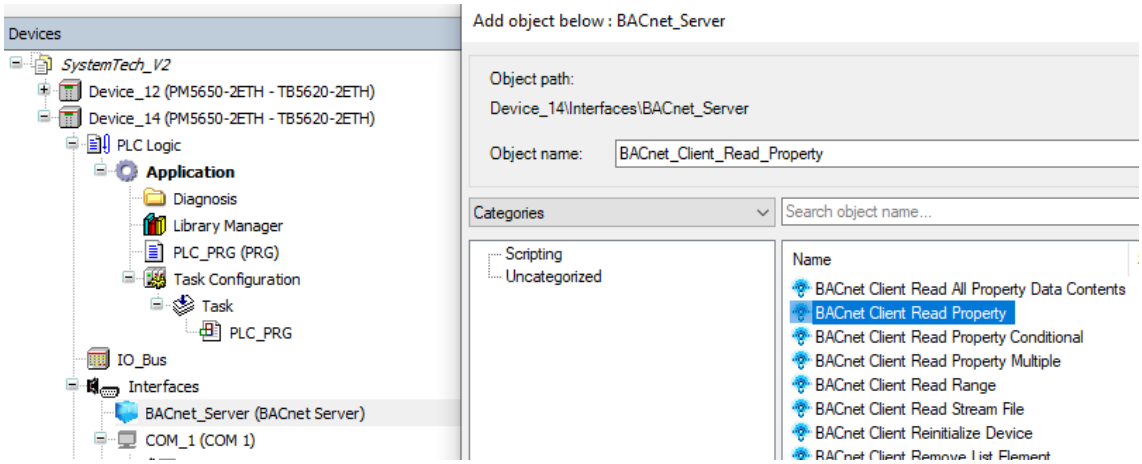
Goal is to configure a second AC500 controller as BACnet client which reads an analog value from a server. This example is according to Fig. 25 *BACnet objects, properties, services and BIBBs*, right part.

1. Add a new controller and configure a *"BACnet Server"* root object according to ↳ *Chapter 1.5.5.3.4.1 "Configuration of BACnet server root object " on page 2214*.

2. Set `InstanceNumber` to `14` and `InstanceName` to `Device 14`.



3. In addition to BACnet objects, BACnet clients can also be inserted as devices under a *“BACnet Server”*. Add a *“BACnet Client Read Property”* below the *“BACnet Server”* node.

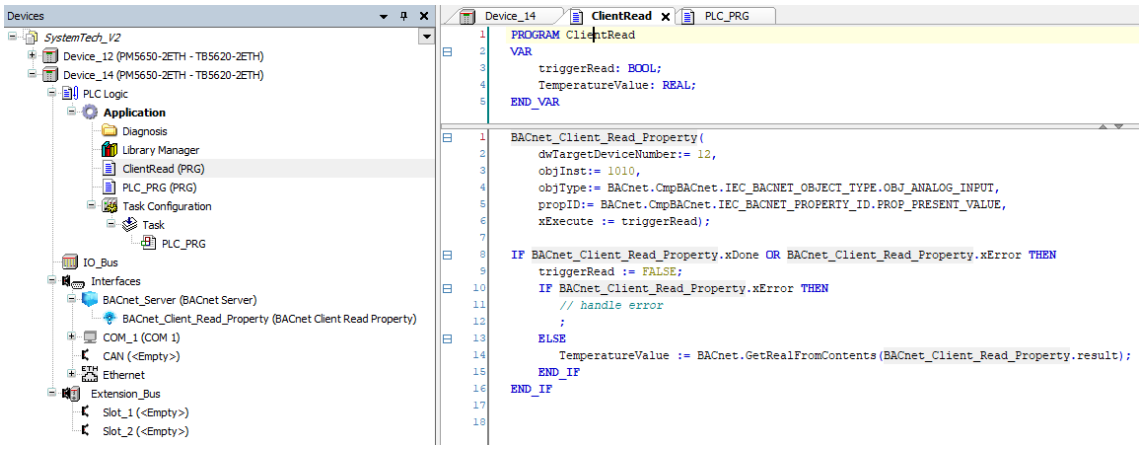


4. The created object *“BACnet Client Read Property”* generates a function block instance which can be used to program the client read functionality. The figure below shows a simple example.

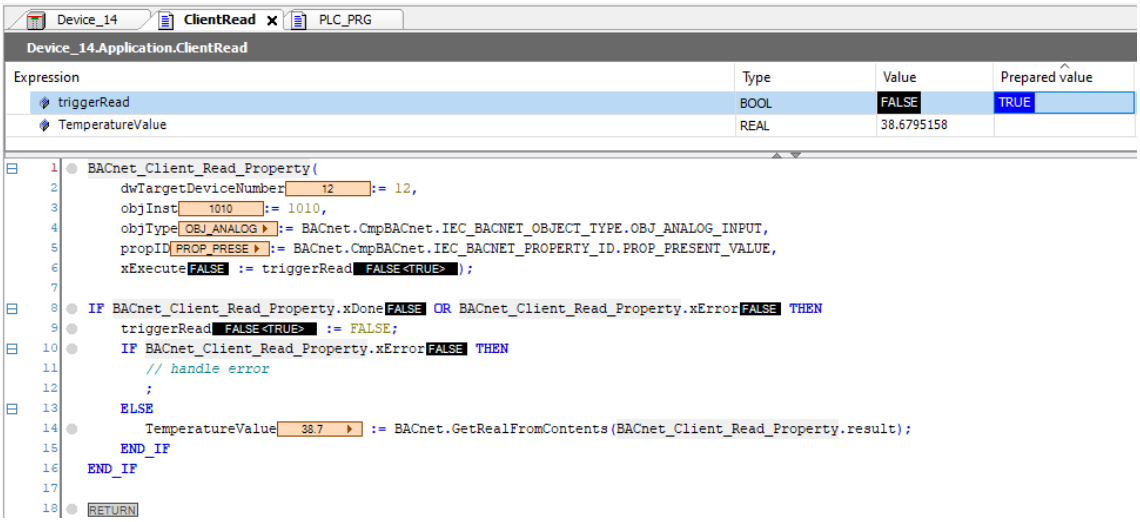
   In line 1-5 of the code part the function block is called with the following parameter:

   - Device ID of the server to read from (`12`) *Chapter 1.5.5.3.2 “Supported objects and properties ” on page 2213*
   - Object ID of the object to read from (`1010` for the *“Analog Input”*)
   - Object type (*“Analog Input”*)
   - Property to read (*“present value”*)
   - `triggerRead` to start the read operation

   When the user (or another program part) sets the variable `triggerRead` from `FALSE` to `TRUE` the edge triggered function block `BACnet_Client_Read_Property` starts operation and sends the read request to the server device. After receiving the reply from the Server, the output `.xDone` gets `TRUE` (line 8) and the temperature value can be read from the output `.result` (line 14).

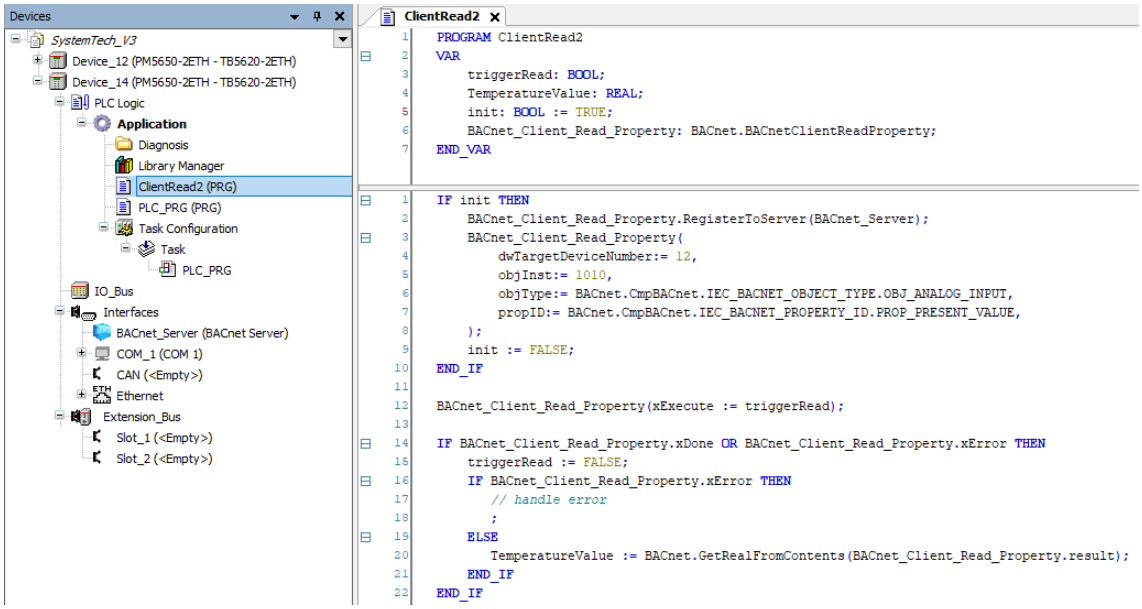


5. Download this program to another AC500 V3 controller, which is in the same IP network as the server. Set it to run and read the temperature value by setting `triggerRead` to `TRUE`. In online mode the read temperature value can be observed in line 14.

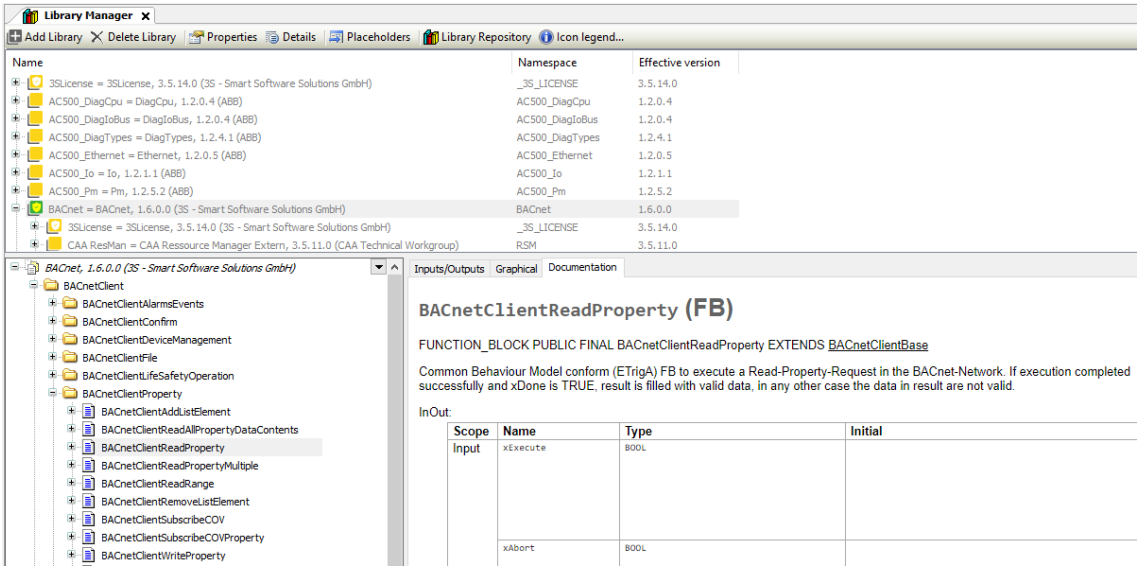


Alternative configuration

Unlike BACnet objects, a BACnet client does not require a complex (static) configuration, thus a client function block can be used without creating a BACnet client as device.



There is no `BACnet_Client_Read_Property` object created below the *"BACnet Server"*. Instead a function block `BACnet_Client_Read_Property` must be declared in the PRG (line 6 in the declaration) and initially "connected" to its *"BACnet Server"* in IEC-code via `RegisterToServer()`, and thus get activated (line 2 in the code) ↳ *Chapter 1.10 "Reference, function blocks" on page 4292*.



Configuration of datalinks

For communication with other BACnet devices AC500 provides two different possibilities: MS/TP and IP.

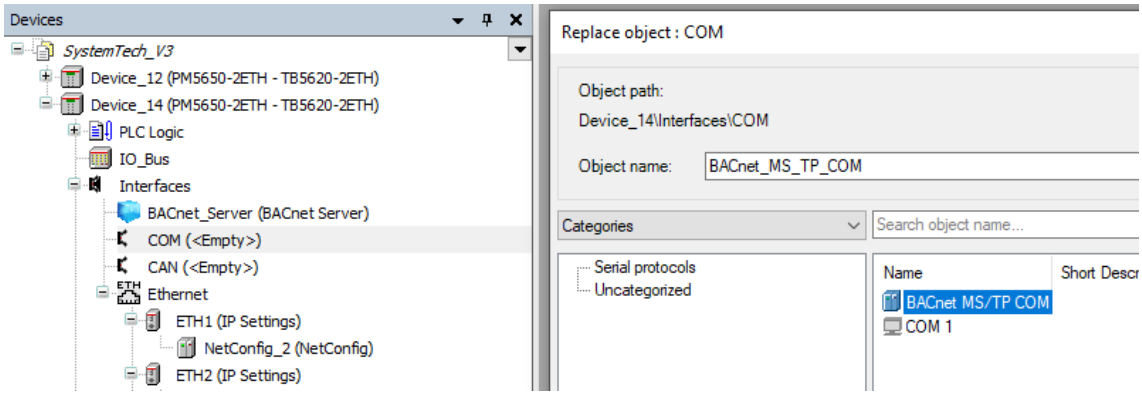
↳ *Chapter 1.5.5.3.1 “Supported BACnet networks ” on page 2211*

For a non-routing device one MS/TP or IP datalink must be configured.

If more than one datalink is configured, routing between the datalinks is automatically enabled.

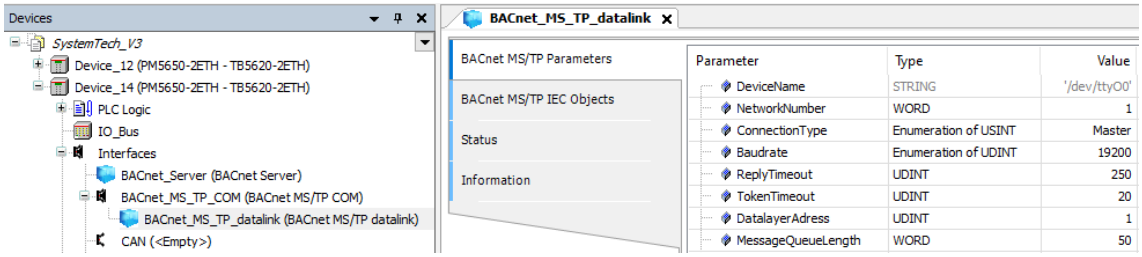
Configuration of an MS/TP datalink

- Add the *“BACnet MS/TP COM”* object below the COM port.



In fact the empty COM port is replaced by the *“BACnet MS/TP COM”*. By that the COM port is configured as RS-485 with fixed settings for MS/TP: No parity, 8 data bits, 1 stop bits.

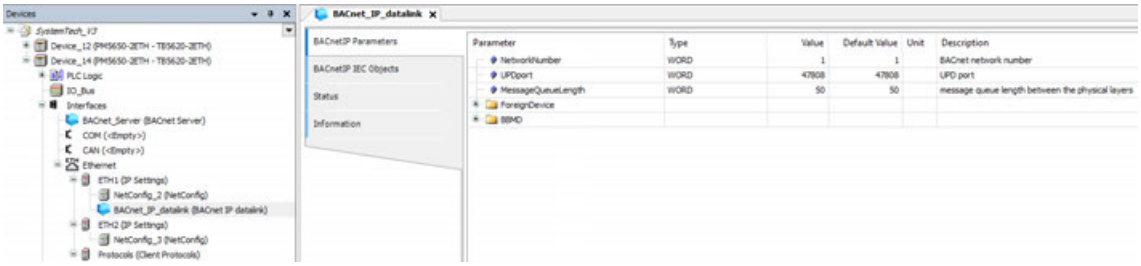
- Below the *“BACnet MS/TP COM”* port object an *“BACnet MS/TP datalink”* is inserted automatically which can be configured according to the requirements.



- `NetworkNumber`: Use the default value `1` if no routing is required. For routing, use a unique network number in one controller.
- `ConnectionType`: Use the default value `Master` if no routing is required. For routing, use *"Master – answering always postponed"*.
- `Baudrate` can be set according to requirements in the range of from 9600 to 38400 bits/s, higher values (57600 and 115200 bits/s) are not recommended.
- `DatalayerAddress`: This is the MAC address as described in *Chapter 1.5.5.3.1 "Supported BACnet networks " on page 2211*. The MAC address must be unique in the MS/TP network.
- For all other parameters the default values are recommended for typical applications.

Configuration of an IP datalink

- Add a *"BACnet_IP_datalink"* object below the Ethernet port `ETH1` or `ETH2`.



- `NetworkNumber`: Use the default value if no routing is required. For routing, use a unique network number in one controller.
- `UPDport`: Use the default value (`47808` decimal) in the normal case. Range is possible from BAC0 (= 47808 decimal) to BACF. UDPport + IP address form the MAC address of the IP datalink as described in *Chapter 1.5.5.3.1 "Supported BACnet networks " on page 2211*. The IP address cannot be specified here. It is automatically taken from the parent Ethernet node (`ETH1` or `ETH2`); its IP address is set in the communication settings of the CPU node, *"Device_14"* in the example.
- `ForeignDevice` and `BBMD`: Special configuration is only needed if an internet router is located between two BACnet devices.
  Chapter 1.5.5.3.1 "Supported BACnet networks " on page 2211
  AC500 can be configured as `ForeignDevice` or `BBMD`, but not the combination of both. An example for BBDM can be found in the example folder.

Configuration of Routing

Routing enables the combination of different BACnet networks to one common *"BACnet internetwork"*.

Chapter 1.5.5.3.1 "Supported BACnet networks " on page 2211

BACnet devices from different BACnet networks can communicate with each other.

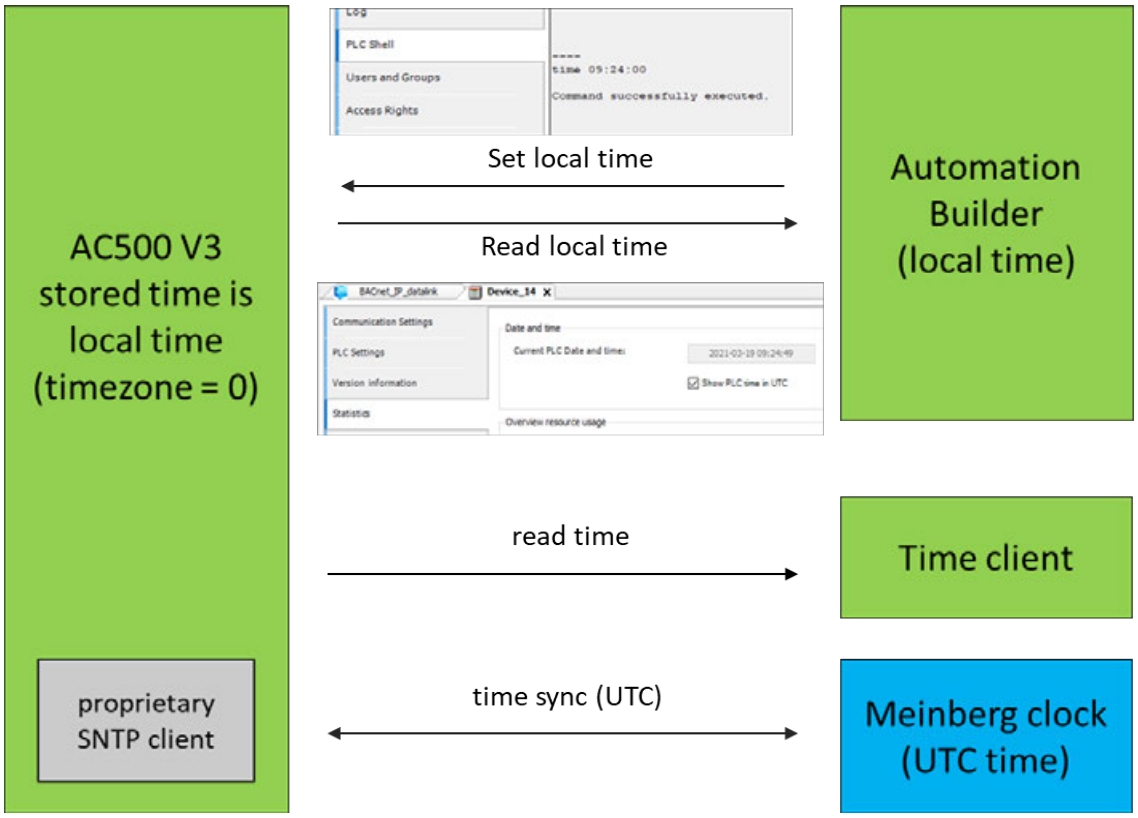
If more than one datalink is configured in one CPU, routing between the different networks is automatically enabled. It must only be ensured that the network number is unique in one controller.

Chapter 1.5.5.3.1 "Supported BACnet networks " on page 2211

For MS/TP the `ConnectionType` must be set to *"Master – answering always postponed"*. An example for routing can be found in the example folder.

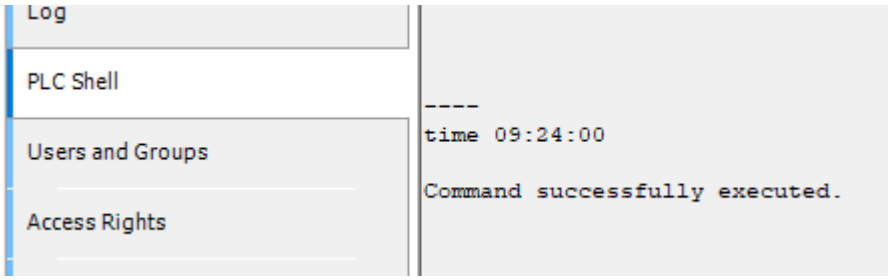
Time syncronisation

The BACnet clients expect to receive the local time. Currently the AC500 V3 does not distinguish between UTC time and local time and its time zone is set to 0. This will be improved in the near future. In the meantime, it is recommended to store the local time (green color in the following figure) in the AC500 as a workaround.

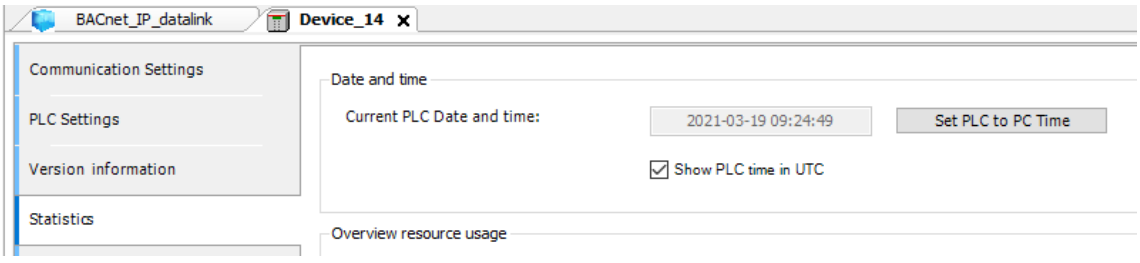


Using this workaround, the following time sync mechanisms can be used:

- Set local time from Automation Builder Tab *"PLC Shell"*:
  Set the time by the command "time hh:mm:ss"



- Read the local time from the Automation Builder Tab *"Statistics"*:
  "Current PLC Date and time" shows the PLC time as local time without conversion, if the tab *"Show PLC time in UTC"* is enabled.



For storing the local time in AC500, do not use the button [Set PLC to PC Time] (Tab "Statistics"), since this is always converting from local time to UTC time.

- BACnet clients can read local or UTC time, both requests will deliver the same (local) time information, since the timezone is 0.
- If an SNTP time sync is required (for example with a Meinberg clock), UTC times are exchanged. For conversion of UTC to local time in AC500 a proprietary STNP client must be programmed.
  Please contact the PLC support for more information.

1.5.5.3.5 Package content

The BACnet package PS5607-BACnet-BC can be installed with the Installation Manager and contains the following components:

- BACnet runtime component, part of AC500 firmware.
- Automation Builder package: CODESYS BACnet
  - BACnet plug-in component
  - Device descriptions for *"BACnet Server"*, BACnet objects, BACnet client and datalinks
  - Libraries: `BACnet`, `BACnetDefaultImpl` and `CmpBACnet`.
    Chapter 1.5.5.3.5.1 "BACnet libraries" on page 2223

Example folder

- Example folder
  - Examples and example documentation
    Chapter 1.5.5.3.5.2 "Application examples" on page 2224
  - Datasheet and FAQ
    BACnet Protocol Implementation Conformance Statement (PICS), acting as a datasheet, describing all BACnet objects, services and communication capabilities.
    BACnet Conformance Certificate
    FAQ – Frequently Asked Questions, including AC500 specific information, performance and limit

BACnet libraries

The IEC library `CmpBACnet` represents the integration of the BACnet stack into a CODESYS IEC environment and provides the BACnet data types as well as the `BACstack` methods. The sole use of the IEC library `CmpBACnet` (without the `BACnet` and `BACnetDefaultImpl` libraries) would result in complex and lengthy IEC application code.

The `BACnet` library simplifies BACnet application development considerably as compared to the sole use of `CmpBACnet`, especially in the following areas:

- Starting and stopping the BACnet stack
- Using BACnet server objects and their properties
- Triggering asynchronous requests (mainly client service requests) and processing the request transaction
- Processing of callbacks from the BACnet stack (see `IBACnetEventConsumer`) and distributing the callbacks to multiple receivers in the application

Furthermore, the `BACnet` library provides a plug-in mechanism (`BACnetServerPlugin`) for extending certain aspects of the `BACnet` library. `BACnetServerPlugin` is the basis for the `BACnetDefaultImpl` library.

The `BACnetDefaultImpl` library is used for the additional simplification of BACnet application development. The BACnet standard ASHRAE 135 leaves some aspects of the practical use of BACnet open. The most notable examples include the following:

- Persistence of server objects
- Storage and persistence of `Trend Log`, `Trend Log Multiple`, and `Event Log` entries
- Update of the date/time information of the device object

The IEC library BACnet is intended as a layer over the IEC library CmpBACnet. However, the layer does not hide the library because this would require the BACnet library to have "facade" functions for CmpBACnet functions. These facade functions would result in larger application code and increased runtime requirements. This is difficult for the PLC to accept. For this reason, it is necessary to know when elements from the BACnet library or CmpBACnet library are to be used.

General rules:

- Starting and stopping the BACnet stack
  Always use BACnetServer.StartBACnetStack and BACnetServer.StopBACnetStack or AutoStart. Never directly use the corresponding functions of the CmpBACnet library, such as CmpBACnet.BACnetServerInit.
- Using BACnet server objects and their properties
  Always use the specified function blocks in IEC-lib-BACnet, such as BACnetAnalogValue. Never directly use the corresponding functions of the BACnet library, such as CmpBACnet.BACnetStorePropertyInstance.
- Triggering of asynchronous requests
  Always use the specified client function blocks of the BACnet library, such as BACnetClientReadProperty. Never directly use the corresponding functions of the CmpBACnet library, such as CmpBACnet.BACnetReadProperty. All functions of the CmpBACnet library that require a BACnetAsyncTransactionToken belong to this category and should never be used directly.
- Processing of callbacks from the BACnet stack and distributing the callbacks to multiple receivers in the application
  Always use IBACnetEventConsumer and BACnetServer.RegisterHook/UnregisterHook/RegisterCallback/UnregisterCallback. Never directly use the corresponding functions of the CmpBACnet library, such as CmpBACnet.BACnetSetHook or CmpBACnet.BACnetSetCallback.

When is it appropriate and safe to directly call the functions of the CmpBACnet library?

Basically, it is only necessary to call functions of CmpBACnet directly when a corresponding functionality is not provided in the BACnet library. Check the BACnet library first before trying to use CmpBACnet directly. It is possible to use blocking functions in CmpBACnet, such as BACnet*CbCompletion, BACnetIam(Ex), or BACnetIHave(Ex), BACnetUnconf*.

Most often, you will use BACnet*CbCompletion to implement your specific IBACnetEventConsumer.BACnetEventCallbacks. But first check whether or not the BACnetDefaultImpl library already contains an appropriate standard implementation.

Application examples

- AC500_V3_BACnet_B-BC_Example_ABxxx.project including simple read and write operations between client and server.
  - Use case 1: AC500 as BACnet client, read and write (with priority)
  - Use case 2: AC500 as *“BACnet Server”*, publish the analog value
- AC500_V3_BACnet_B-BC_Example_Routing_ABxxx.project
- Examples from 3S, including
  - Read and write operations with more options, notification class, calendar, scheduler, etc.
  - Device discovery
  - BBMD
  - Persistence
  - Logging
  - Routing

1.5.6 CAA library guidelines

Function block descriptions for the CAA library can be found in the Library Manager.

The guidelines for the CAA libraries correspond to the general guidelines for library development. For a detailed description see help chapter *Guidelines for Library Development*.

With the help of the CAA library, different use cases for dealing with AC500 PLCs can be programmed. A possible example is the use of the so-called CSV read function, with which information from CSV or other formatted DAT files can be read into Structured Text. This use case is described in the *application example*.

Another application example describes the file handling in order to *write, read and append files*.

1.5.7 Datalogging library

1.5.7.1 Overview

The Datalogging function block library (PS5609-Log) contains five function blocks for the purpose of advanced time-stamped data logging for different use cases.

In the most challenging use case it also can be called buffering: The AC500 application program generates data which are normally transmitted to a telecontrol system for storage and further processing or displaying to the end user. Typically, these may be remote applications like water- or oil-pumping or electrification stations or solar power plants. The connection between these remote stations with an AC500 and a central SCADA/telecontrol station is not always stable or only sporadically connected. Sporadically connected can be by intention, e.g. to save communication costs or open ports/connections to be used with a control station only in a limited way.

- Then the Datalogging function blocks buffer or store data in case of a broken or intentionally interrupted connection between AC500 and the telecontrol system.

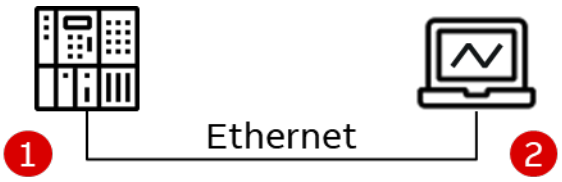


Fig. 26: Overview

1 AC500 application (remote substation)
2 Telecontrol (control station)

- The Datalogging library can be also used as an event recorder. In this special mode data is continuously recorded in a ring buffer which can be read out after a certain event x (e.g. outage) in order to analyze the values especially before but also after the event x.
  OR
- Data can be logged only and on command transferred to the ftp area to be analyzed offline or taken out via the memory card.

The following figure gives an overview of the described interaction of the data logging function blocks. There is always an input function block (*"LogInput"*) needed which transfers the input data into data sets with timestamp for use by the datalogger (*"LogHandling"*). An output function block (LogOutput) receives the current or retrieved data from the datalogger in case of communication or further processing. The input function blocks *"Logxx_Input"*, the function block

"LogHandling" and the output function blocks *"Logxx_Output"* communicate via SRAM FIFOin and FIFOout areas in the memory. The SRAM FIFOin is power-fail-safe intermediate buffer and help in decoupling time wise and speeding up the necessary write/read operations on the logging file structures significantly. These read/write operations on the files are done in blocks of Data sets, enabling a comparably fast interaction with the otherwise slow file system.

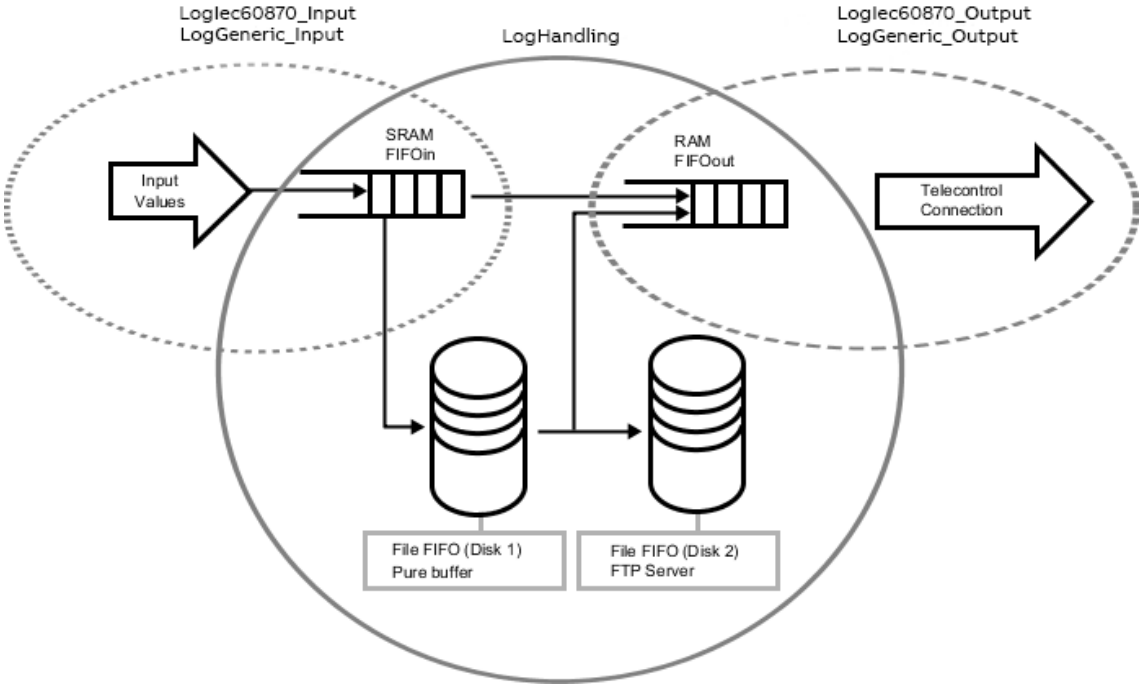


Fig. 27: Overview function blocks

Each Datalogging application requires the main function block *"LogHandling"*, one of the input function blocks to provide data to be logged and one output function block to retrieve the data and send the data to telecontrol (scada).

As input and output function blocks two different types exist:

- For logging data of an interrupted IEC60870 communication, the function blocks *"Loglec60870_Input"* and *"Loglec60870_Output"* are provided. The IEC60870 Datalogging function blocks support the IEC data types and work internally with the standard AC500 IEC60870 library. The IEC Datalogger output function block does not need special handling or control/inputs.
- For other types of general data *"LogGeneric_Input"* and *"LogGeneric_Output"* are provided. The generic Datalogging function blocks support an even larger variety of data types. The generic output function block needs to be hand-shake with for each data set, in order to retrieve the data from the Datalogging files. Therefore the generic function blocks can also be used to integrate the data logging into any other protocol, e.g. Modbus.

The function block *"LogHandling"* ensures that also several consecutive and fast interruptions can be handled without losing data. While the log file is replayed, arriving new data is stored in the SRAM FIFOin and added to the Datalogging files (File FIFO) if the SRAM FIFOin becomes full (during that short time the log file replay is paused). Nevertheless any data send to a control station via a communication is always with the oldest data first (FIFO = "First In First Out").

As it takes up to 30 seconds before a communication break is detected (e.g. with TCP/IP protocols by the AC500 hardware/firmware), the data rate at which data should be logged in case of a communication break has to be calculated and limited.

As an improvement a ping mechanism should be implemented in the substation. This was done in the example program for the IEC logger. With this ping the interruption is already detected after 1-2 seconds (can be configured in the example program - the configurable *"SecureReadTime"* must be considered in this context. This ensures that the time delay - before a loss of connection is detected and is compensated).

As the SRAM FIFO has to store data during this time its size limits the data rate. The SRAM FIFO size is 160 Data sets. If data rate is too high, FIFO will overflow. The maximum data rate is depending on the CPU type, storage media (memory card / flash disk) and cycle time configured, must be determined by try and error.

The data rates for storing only without this detection can be much higher and depends on the CPU and memory type chosen. The data is always logged in directly readable csv format *Chapter 1.5.7.1.5 "CSV file formats" on page 2232*. Depending on the input function block and data type, the log file contains only one or up to 32 data variables per timestamped data set. The Datalogging files can be configured (up to 65k Data sets per file, up to 999 consecutive log files, name format).

1.5.7.1.1 Operating modes

This chapter describes the different operating modes of the Datalogging and their behavior.

- Mode 0/1: Buffer and disposal in chronologic order
  - Mode 0: Limited storage (keeps oldest, but stops if full)
  - Mode 1: Endless (ring buffer) operation modes (deletes oldest)
- Mode 2: Buffer and disposal via FTP, Log file(s) copied to ftp server area for further use
- Mode 3: Events Recorder, logs data before and after an event.

Mode 0/1: Buffer and disposal in chronologic order

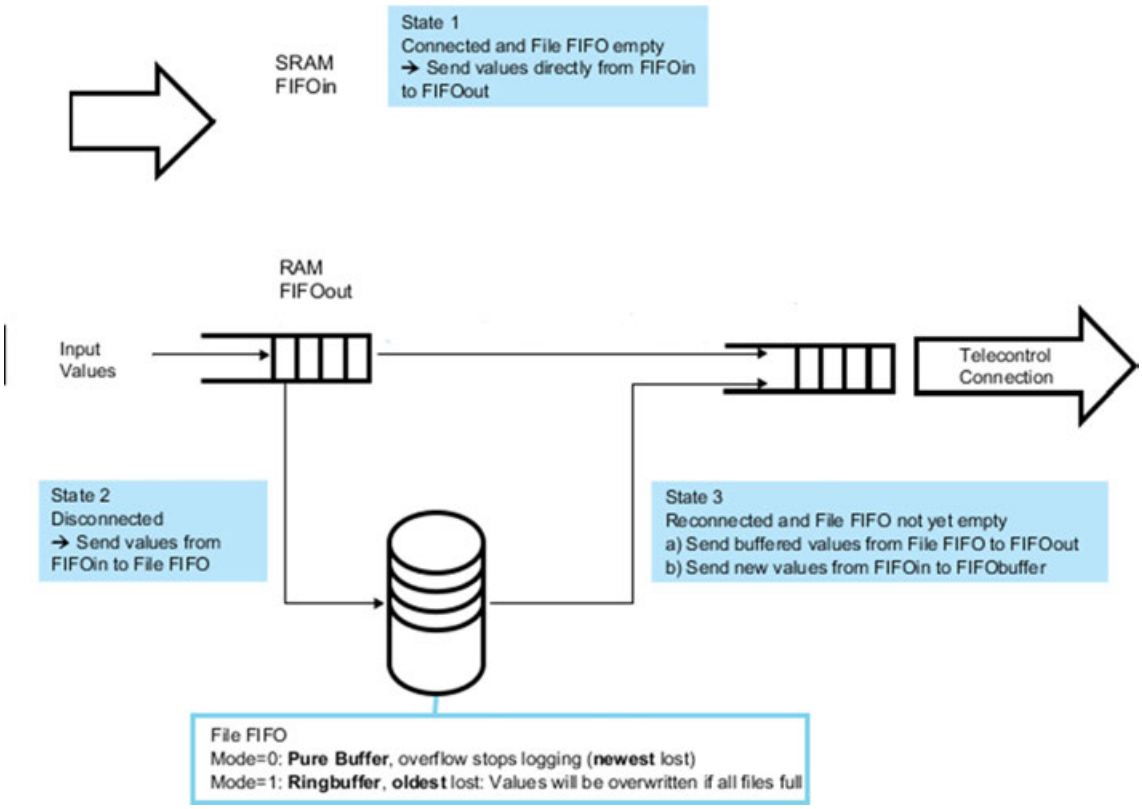


Fig. 28: Overview Mode 0/1

Mode 0/1 is for buffering the values from the AC500 application in case of a broken or intentionally interrupted connection between AC500 and telecontrol. In the normal state 1 the values are directly sent from the FIFOin (input values from application) to FIFOout (telecontrol connection). As soon as the connection is interrupted, the Datalogger changes to working state 2. The values are sent to the file FIFO instead. When the file FIFO is full, the Datalogging is stopped (Mode 0) or the oldest data will be overwritten (Mode 1 = ringbuffer). When the connection is established again and the *"ReleaseHistory"* pin is triggered, the datalogger changes to working state 3. It cares for disposal of the values in chronological order. The buffered values are

written to FIFOout (working state 3a). This may take some time during which new values are coming from the application and stored into FIFOin. Before the FIFOin overflows the datalogger switches to working state 3b and buffers the new values. After that it can continue with working state 3a. Only if the File FIFO is empty (all files deleted) the datalogger changes back to normal state 1.

The advantage of Mode 0/1 is that all values (directly and buffered) are sent to telecontrol in strictly chronological order which is expected by most control stations (SCADA systems/historians).

If a historical value is sent to the SCADA after a current value has already been sent, the historical value is normally rejected by the SCADA.

As it takes up to 30 seconds before a communication break is detected (e.g. with TCP/IP protocols by the AC500 hardware/firmware), the data rate at which data should be logged in case of a communication break has to be calculated and limited. It therefore makes sense to use PING to detect a possible interruption in the connection. This enables an earlier detection of the connectionless state.

Mode 2: Buffer and disposal via FTP

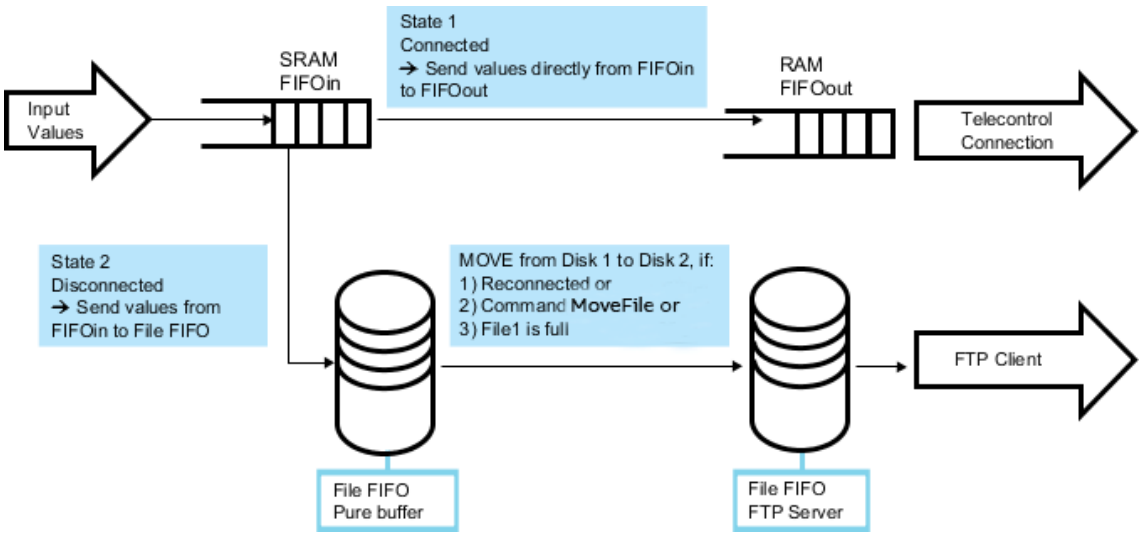


Fig. 29: Overview Mode 2

Mode 2 is also used for buffering the values from AC500 application in case of a broken connection between AC500 and telecontrol. State 1 and state 2 are similar to Mode 0/1. The difference is the disposal. When the connection is established again the Datalogger changes directly back to state 1 and the input values in FIFOin are directly sent to FIFOout (telecontrol connection). The buffered values in File FIFO are internally moved from disk 1 to disk 2 which can then be accessed or used by FTP (client or server). This move action can also be triggered by the command *"MoveFile"*, or when file 1 is full. The advantage of Mode 2 is the immediate availability of the latest and all current values after an outage.

Mode 3: Events recorder

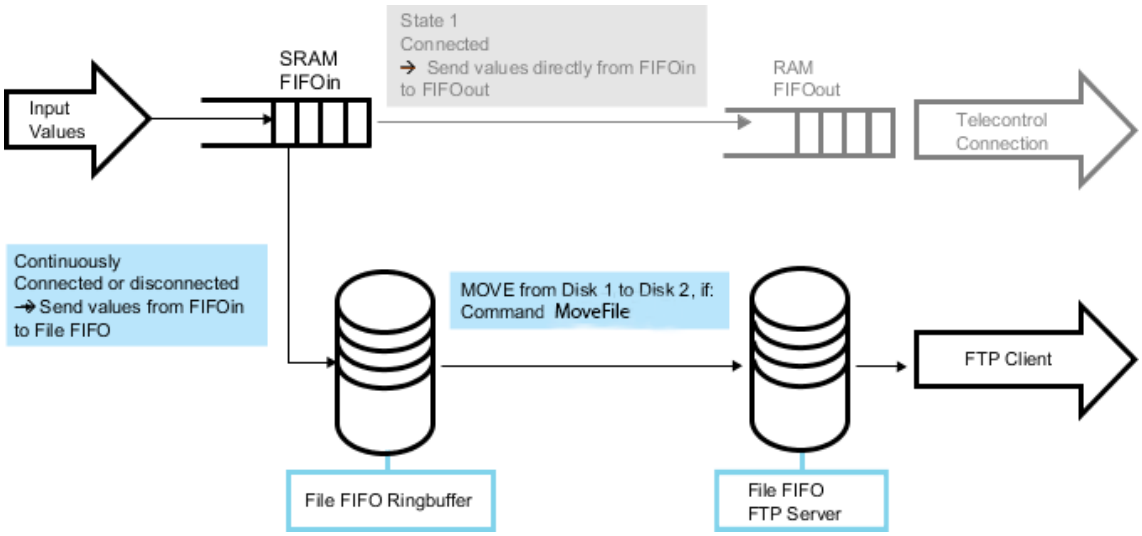


Fig. 30: Overview Mode 3

Mode 3 is used to record data values around an event, before and after the event X, e.g. outage of a part of the plant. The values are continuously recorded into the File FIFO file system independent of the connection status to telecontrol. If the File FIFO is full the oldest values are overwritten (ring buffer). Thus the file FIFO always contains the values from the past period n, which is depending on the number of values per second and on the size of the File FIFO. When a certain event x occurs, the command *"MoveFile"* can be given directly or after the period m. With the command *"MoveFile"* the values in File FIFO are internally moved from disk1 to disk 2 and can be read out by an FTP action (client or server) when required.

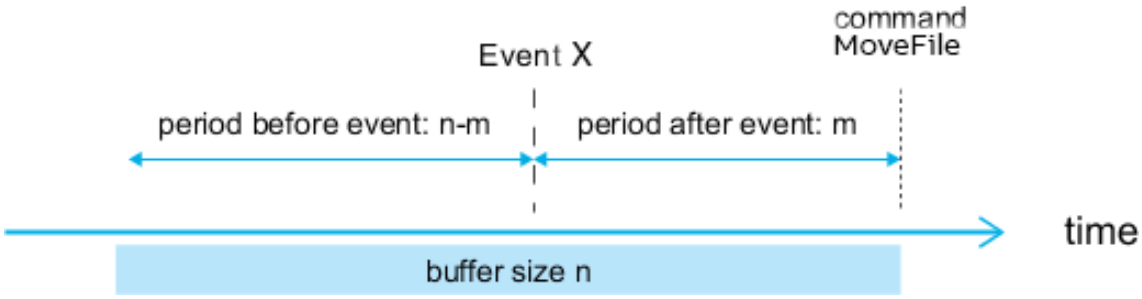


Fig. 31: The buffered values represent the time before the event (n-m) and after the event (m).

The advantage of Mode 3 is that the values from the time period before the event (n-m) and after the event (m) are recorded and can help to reconstruct the cause and effect of the event.

1.5.7.1.2 Technical details

Parameter	Value
IEC 60870-5-104 protocol integrated in Datalogging, IEC data types	SinglePoint SP1/16, DoublePoint DP, IntegratedTotal IT1/16, MeasurementValue ME1/16
Generic logging to file(s); AC500 data types	BIN, BYTE, INT, UINT, DINT, UDINT, REAL
Trigger	Cyclic, event, tolerance

Parameter	Value
File format	CSV, including local timestamp. Generic Logging with separate ID (max. 8 characters), IEC Logging with IEC addresses. Timestamped Data sets contain 1-16 values (IEC) depending on type logged. Generic contain different number of values, depending on type logged. BINARY: max. 58 BYTE: max. 88 INT: max. 50 UINT: max. 58 DINT: max. 29 UDINT: max. 31 REAL: max. 27
Datalogging target	Flash disk or memory card, power fail input for memory card (from UPS)
Datalogging file sizes and storage depth	FIFO storage in file system, Datalogging depth only limited by memory size
Configurable file FIFO	Number of files (max. 999); number of Data sets per file (max. 65535)
Internal SRAM FIFO size	160 Data sets
Block write mode into files	Up to 50 Data sets/second per max. 88 values
Operation modes	Mode 0: Buffer and disposal in chronologic order Limited storage (keeps oldest, but stops if full) Mode 1: Buffer and disposal in chronologic order Endless (ring buffer) operation modes (deletes oldest) Mode 2: Buffer and disposal via FTP, Log file(s) copied to ftp server area for further use Mode 3: Events Recorder, logs data before and after an event.
Supported software/firmware	V3.4 or higher
Current restrictions	One logger per PLC One IEC 60870 connection only: While log file is replayed, no other current information via IEC available Usable solutions: ● Delay replay of log file after connection returned to allow a “general inquiry” ● Use of Mode 2

Logging capacity:

- Data set: 1 data set always has 400 byte
- FIFOin: Has a maximum capacity of 161 data sets (a 400 byte = 64400 byte)
- FIFOout: Has a maximum capacity of 161 data sets (a 400 byte = 64400 byte)
- File: 1 file stores up to 65535 data records, which are copied block by block from the FIFOin in case of a communication lost
- A maximum of 999 files can be saved.

In purely mathematical terms, that would be 999 files * 65535 Data sets * 400 byte / Data set = 26,187,786,000 bytes = 26 GB

Since neither the flash disk nor the usable memory card have such a capacity, the user has to find a sensible compromise. The flash disk as a storage medium is fail-safe, i.e. in the event of a sudden power failure, data in the possibly currently open file is reconstructed when the power is restored. This is not the case with a memory card. There the file is destroyed. It is therefore advantageous if such a variant is operated with a power supply that keeps the PLC alive for at least a few seconds after the supply voltage failure (see *"Input ExternalPower"* on the *"LogHandling"* function block).

Time synchronization:

Currently the AC500 V3 does not distinguish between UTC time and local time and its time zone is set to 0. This will be improved in near future. In the meantime, it is recommended to store the local time in the AC500 as a workaround.

Chapter 1.5.5.3.4.5 "Time syncronisation" on page 2221

1.5.7.1.3 File names

File names are renamed according to storing time with an accuracy of 100 ms. The files are renamed from *"filename.csv"* to a file name with timestamp and with or without file extension, according to input *"Disk2Extension"* is applicable only in Mode 2.

File name with timestamp	02281448.593 = February 28th, 2:48pm (14:48), 59s, 300ms
File name with timestamp and file extension	02281448.csv = February 28th, 2:48pm (14:48)

1.5.7.1.4 Preconditions

The Datalogging library supports CPU PM5650 or higher.

The Datalogging library does not support CPUs of the AC500-eCo series.

CAUTION!

Failure in Processing of the function blocks.

– The function blocks LOGxxxxxx_Input, LogHandling and LOGxxxxxx_Output must be put in the same task.

CPU firmware must be V 3.4.0 or higher.

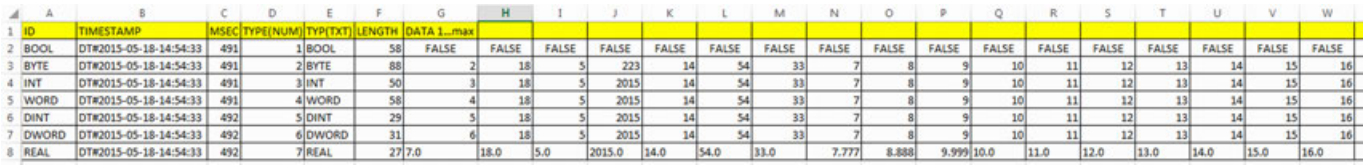
Use memory card from ABB with sufficient free space, at least 1.5 x file size as configured (file size is depending on input *"MaxNumDatasetFile"* from *"LogHandling"* function block).

Maximum number of files (input of *"LogHandling"*) is limited to 999. ABB memory card is formatted with FAT by default.

Chapter 1.6.7.1 "Introduction of AC500 storage devices for AC500 Products" on page 3994

1.5.7.1.5 CSV file formats

Generic data logger



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ID	TIMESTAMP	MSEC	TYPE(NUM)	TYP(TXT)	LENGTH	DATA 1...max																
2	BOOL	DT#2015-05-18-14:54:33	491	1	BOOL	58	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
3	BYTE	DT#2015-05-18-14:54:33	491	2	BYTE	88	2	18	5	223	14	54	33	7	8	9	10	11	12	13	14	15	16
4	INT	DT#2015-05-18-14:54:33	491	3	INT	50	3	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
5	WORD	DT#2015-05-18-14:54:33	491	4	WORD	58	4	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
6	DINT	DT#2015-05-18-14:54:33	492	5	DINT	29	5	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
7	DWORD	DT#2015-05-18-14:54:33	492	6	DWORD	31	6	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
8	REAL	DT#2015-05-18-14:54:33	492	7	REAL	27	7.0	18.0	5.0	2015.0	14.0	54.0	33.0	7.777	8.888	9.999	10.0	11.0	12.0	13.0	14.0	15.0	16.0

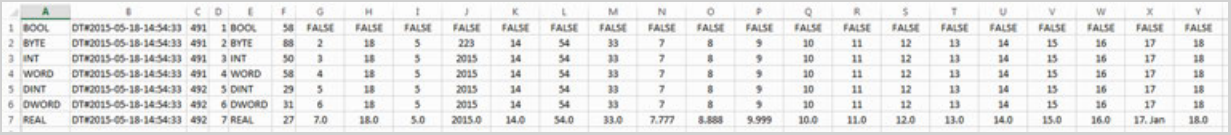
Fig. 32: Explanation of the csv file structure

1 Data set consists of:

ID (8 any char) + TimeStamp + msec + Datatype(num) + Datatype(txt) + Length(following data) + max 32 data

Parameter	Value
ID =	ID of LogGeneric_Input (max 8 any characters)
Datatype =	DataType of LogGeneric_Input (1…7)
Length =	Length of LogGeneric_Input (max 88) BINARY (58); BYTE (88); INT (50); UINT (58); DINT (29); UDINT (31); REAL (27)

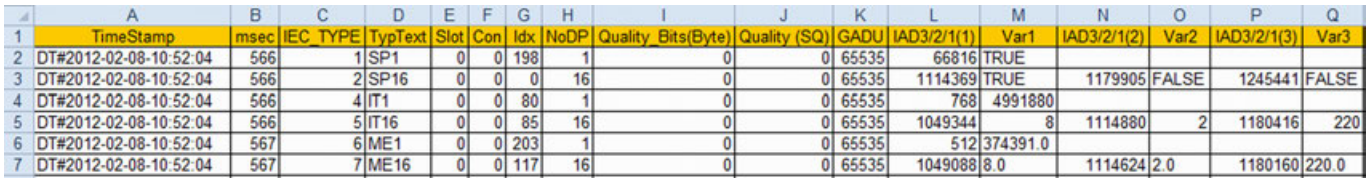
Example



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	BOOL	DT#2015-05-18-14:54:33	491	1	BOOL	58	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	BYTE	DT#2015-05-18-14:54:33	491	2	BYTE	88	2	18	5	223	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
3	INT	DT#2015-05-18-14:54:33	491	3	INT	50	3	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
4	WORD	DT#2015-05-18-14:54:33	491	4	WORD	58	4	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
5	DINT	DT#2015-05-18-14:54:33	492	5	DINT	29	5	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
6	DWORD	DT#2015-05-18-14:54:33	492	6	DWORD	31	6	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
7	REAL	DT#2015-05-18-14:54:33	492	7	REAL	27	7.0	18.0	5.0	2015.0	14.0	54.0	33.0	7.777	8.888	9.999	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17. Jan	18.0

Fig. 33: File opened directly with Excel

IEC60870 Data-logger



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	TimeStamp	msec	IEC_TYPE	TypText	Slot	Con	Idx	NoDP	Quality_Bits(Byte)	Quality (SQ)	GADU	IAD3/2/1(1)	Var1	IAD3/2/1(2)	Var2	IAD3/2/1(3)	Var3
2	DT#2012-02-08-10:52:04	566	1	SP1	0	0	198	1	0	0	65535	66816	TRUE				
3	DT#2012-02-08-10:52:04	566	2	SP16	0	0	0	16	0	0	65535	1114369	TRUE	1179905	FALSE	1245441	FALSE
4	DT#2012-02-08-10:52:04	566	4	IT1	0	0	80	1	0	0	65535	768	4991880				
5	DT#2012-02-08-10:52:04	566	5	IT16	0	0	85	16	0	0	65535	1049344	8	1114880	2	1180416	220
6	DT#2012-02-08-10:52:04	567	6	ME1	0	0	203	1	0	0	65535	512	374391.0				
7	DT#2012-02-08-10:52:04	567	7	ME16	0	0	117	16	0	0	65535	1049088	8.0	1114624	2.0	1180160	220.0

Fig. 34: Explanation of the csv file structure

Table 378: 1 Data set consists of the following parameters

Parameter	Explanation
IecType =	IecType of LogIec60870_Input (1…7)
Slot/Con/Idx/NoDP =	Pin group of LogIec60870_Input (1…7)
Quality_Bits(Byte) =	IV/NT/SB/BL/CA/CY/QOV (packed in 1 byte) of LogIec60870_Input (1…7)
Quality (SQ) =	SQ of LOG_IEC60870_INPUT (1…7)
GADU =	calculated internally, from Automation Builder Configurator (Gadu1+Gadu2)

Parameter	Explanation
IAD3/2/1(n) =	calculated internally, for every datapoint separately, from Automation Builder Configurator (IAD1+IAD2+IAD3)
n =	1 or 16, in case of DP is n=2
VAR(n) =	variable

IEC type	Values	Meaning
SP1	-	SinglePoint 1
SP16	-	SinglePoint 16
DP	-	DoublePoint
IT1	-	IntegratedTotal 1
IT16	-	IntegratedTotal 16
ME1	-	MeasurementValue 1
ME16	-	MeasurementValue 16
Quality_Bits(Byte):	Quality.0 := IV;	Quality with quality invalid
	Quality.1 := NT;	Quality not topical
	Quality.2 := SB;	Quality substituted
	Quality.3 := BL;	Quality blocked
	Quality.4 := CA;	Quality with quality carry
	Quality.5 := CY;	Quality with quality counter adjusted
	Quality.6 := QOV;	Quality Overflow Quality
	Quality.7 := Reserve;	(*Reserve - Quality*)
Quality SQ(Byte)	SQ	Quality sequence number (Range: 0 to 31)

Example

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	DT#2012-02-08-10:52:04	566	1	SP1	0	0	198	1	0	0	65535	66816	TRUE												
2	DT#2012-02-08-10:52:04	566	2	SP16	0	0	0	16	0	0	65535	1114369	TRUE	1179905	FALSE	1245441	FALSE	1310977	FALSE	1376513	TRUE	1442049	FALSE	1507585	FALSE
3	DT#2012-02-08-10:52:04	566	4	IT1	0	0	80	1	0	0	65535	768	4991880												
4	DT#2012-02-08-10:52:04	566	5	IT16	0	0	85	16	0	0	65535	1049344	8	1114880	2	1180416	220	1245952	10	1311488	52	1377024	4	1442560	249594
5	DT#2012-02-08-10:52:04	567	6	ME1	0	0	203	1	0	0	65535	512	374391.0												
6	DT#2012-02-08-10:52:04	567	7	ME16	0	0	117	16	0	0	65535	1049088	8.0	1114624	2.0	1180160	220.0	1245696	10.0	1311232	52.0	1376768	4.0	1442304	249594.0

Fig. 35: File opened directly with Excel

1.5.7.2 Examples

Example projects for the libraries can be found in the folder:
\Users\Public\Documents\AutomationBuilder\Examples\PS5609-Log

1.5.8 High Availability Modbus TCP

1.5.8.1 HA-Modbus TCP - System technology

1.5.8.1.1 The AC500 High Availability system

The AC500 High Availability system is designed for the demand of automation systems that require a higher availability, which is realized by redundant devices and communications. The redundancy concept reduces the risk of losing production due to failure of parts of the automation system and thereby minimizes scheduled idle times.

For instance, control can be taken over by the secondary station automatically if the primary station fails.

AC500 High Availability system implements redundancy based on standard AC500 PLCs:

- PLC
- Field communication
- SCADA communication

General differences in high availability / redundancy systems are in which way and how fast the switchover between redundancies happens.

- Cold standby: A replacement system is there but not up and running - Process has (to allow) to completely stop for switchover – e.g. outputs may go to zero.
- Warm standby: Both CPU may be running (= warm) but e.g. communication need to be started/stopped for switch-over - Process needs to tolerate longer freeze times e.g. on outputs - e.g. several seconds.
- AC500 High Availability systems are "hot-standby":
  - Redundant CPUs and all communications are always up and running (hot)
  - Continuous failure detection in both CPU´s and mutual exchange of status
  - Continuous synchronization of critical/historical data from primary to secondary
  - Automatic switch-over in very short time in case of any failure in primary CPU

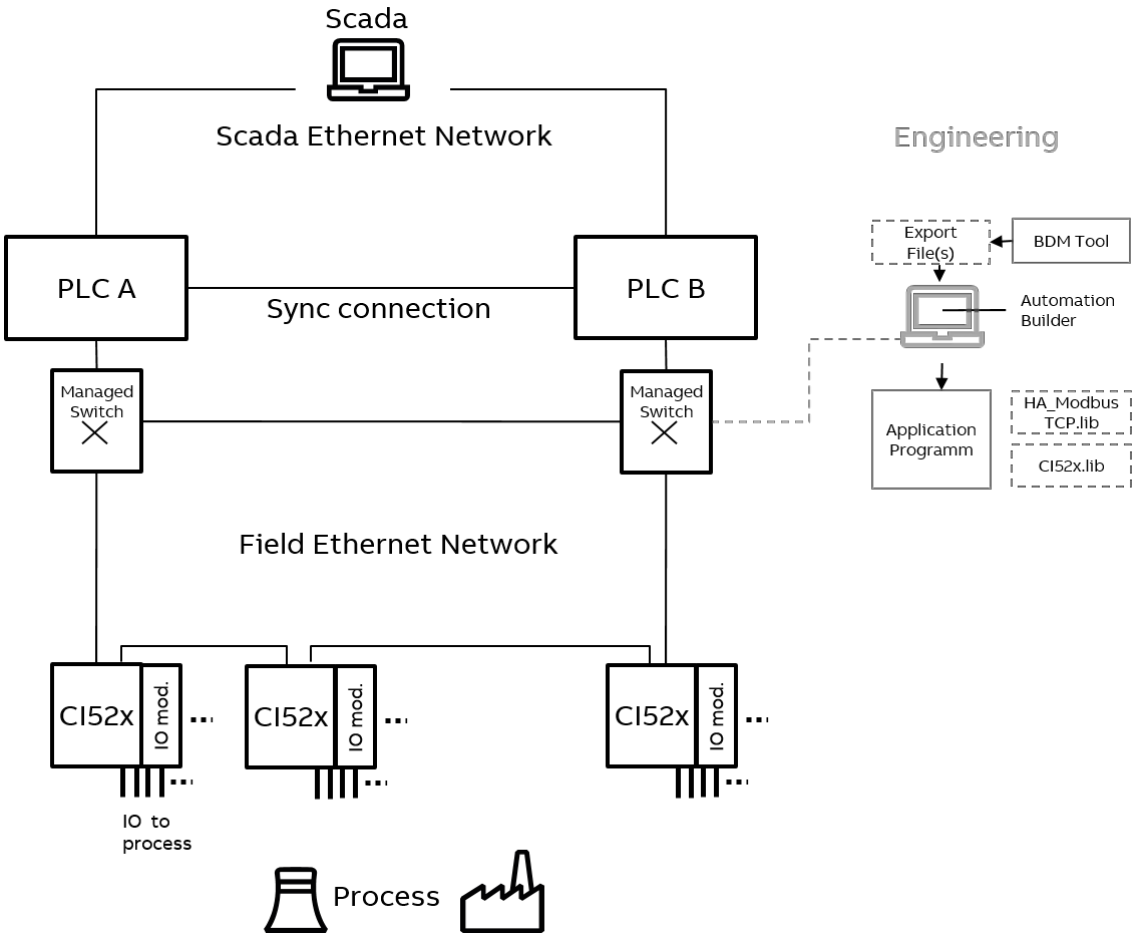


Fig. 36: Principle AC500 HA-Modbus TCP architecture example based on Ethernet redundancy

Details of AC500 HA operation along the figure above:

- PLC redundancy: The two PLCs (A and B) are running in parallel and calculating and reading.
  One is *"primary"* = active, which means also writing data to field devices.
  The other one is *"secondary"* (= stand-by), also calculating but only reading data from field and receiving synchronization (or short = sync) data from the primary.
- Synchronisation data are critical internal variables with e.g. historical content, which will be transmitted from primary to secondary CPU over the sync connection, so that secondary always has the latest data and can take over immediately. Automatically synced are the historic data of the special HA library function blocks (like counters, timers, integral controllers, …), additional Data e.g. of events and diagnosis can be synced by the user with sync blocks. The sync connection also transmits a *"lifecom1"* signal (back and forth) containing diagnosis data of each CPU, so that both CPU know the status of the other CPU. If secondary CPU receives no *"lifecom1"* anymore it assumes that primary CPU has a failure and takes over primary status. If the sync connection is broken both CPUs would try to adopt primary status, therefore, a separate connection *"lifecom2"* is used to differential a "sync link" failure from an "other PLC" failure. The *"lifecom2"* should be routed via a different physical communication path than the data sync/lifecom1, e.g. the Field or SCADA network.
- The field I/O connection is performed via the Ethernet protocol *ModbusTCP* - connecting the CI52x devices (*Chapter 1.6.3.7.4.1 "CI521-MODTCP" on page 3156* or *Chapter 1.6.3.7.4.2 "CI522-MODTCP" on page 3196*).

For high availability/redundancy of the field or SCADA network, proven Ethernet network redundancy mechanisms are used. (In AC500 this is assumed to be realized by at least 2 (to avoid a single point of failure) external, managed switches), which has the advantage to be able to use AC500 HA with any faster redundancy mechanism / protocol.

- For the I/O communication with CI52x modules two variants exist (see online help: PLC Automation with V2 CPUs ➔ PLC integration ➔ Device specifications ➔ Communication interface modules (S500) ➔ Modbus XY)
  For smaller systems, the CI52x modules can be directly daisy chained (as in previous figure above) if MRP (Media Redundancy Protocol) or DLR (Device Level Ring) is used. Ci52x are not actively participating in ring recovery however, a special FW allows fast ring detection and very short freeze times. Larger systems with e.g. many IO and clusters typically anyway connect to the network via a dedicated managed switch.
- SCADA connection is redundant by nature of the two Ethernet ports and can be extended with further redundancy level as well by managed switches. SCADA itself can also switch the primary PLC to ensure communication to the active PLC in case of a simple connection and a connection failure. If the redundancy mechanism of the OPC DA server is not used, SCADA level itself must be able to handle and differentiate primary and secondary PLC and IP addresses based on the HA-status bits. For CP600 a script exists to do the same for Modbus or AC500 communication protocol.

In most PLC applications the critical components to fail are, beneath PLC, typically the power supply or communication components such as wires or switches. Therefore a *SPOF (Single Point Of Failure)* has to be avoided by adding redundant devices or redundancy functions wherever a failure likelihood is high and failures are not tolerable.

HA core functionality typically can tolerate only a single failure in the different levels. Then, a repair of the failed part is highly advised to achieve and ensure redundancy again. As shown in the above figure, the I/O-network cabling already provides a second independent redundancy layer e.g. for cable failure by its redundancy mechanism (e.g. ring), which can keep up communication without switching the PLCs: There a second failure in the PLC level could be tolerated as long as both connecting, managed switched still work, but it is highly advised to repair immediately anyway.

The AC500 High Availability system itself only takes care of the first fault. For example, in case of a second fault the primary PLC remains primary PLC until the second fault occurs. This results in no further switchovers (manual switchovers included).

Due to the efficient data sync mechanism, which allows data sync over normal and shared ethernet networks, with a well-planned communication network, the PLCs can operate geographically separated (by many 10^{th} of kilometers). So even in catastrophic events with full mechanical destruction still one PLC will be available to control the process or infrastructure.

The secondary PLC or single CI52x modules can be exchanged in a running system without interruption of the primary PLC or the process. (Check document in “Examples” directory of Automation Builder if HA package was installed.)

Libraries

In order to achieve high availability, the CODESYS application must be enhanced with HA function blocks, from the HA-Modbus TCP library and the CI52x library. If the bulk data manager tool (BDM) is used for configuring the System and I/O modules - this is done automatically for the basic initial configuration step by code creation resulting in a prepared user specific “template” application (see below).

- HA-Modbus TCP library contains *HA control* and *HA utility* function blocks
  - *HA control* function blocks manage the core HA functionality by collecting diagnosis and switching if necessary.
  - *HA utility* function blocks provide standard functions in the application program with internal *sync* for integral data e.g. timers, counters, PI control.
- *CI52x library* contains a function block to configure and communicate to the communication interface modules and ensures that only the primary PLC writes to the outputs. The inputs are read by both PLCs.
- For both PLCs the same application must be used/downloaded.

Bulk data manager tool (BDM)

For configuration of the CI52x Modbus TCPs, a separate Bulk Data Manager tool (BDM) is provided. Especially in larger systems usage of BDM is recommended to comfortably engineer HA and create CI52x related configuration and variable data in one place:

- Configuration and parameters of the used I/O modules
- Program code creationfor variable naming, configuration, communication and all basic HA functionality

The BDM tool can serve SCADA programming and documentation as well in an efficient manner.

1.5.8.1.2 Hardware, requirements and options overview

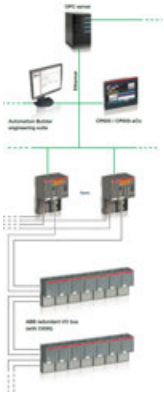
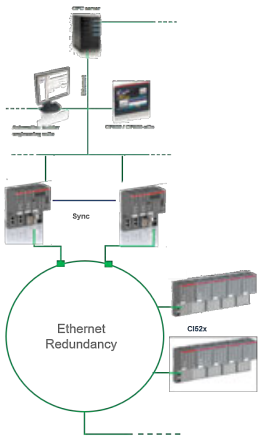
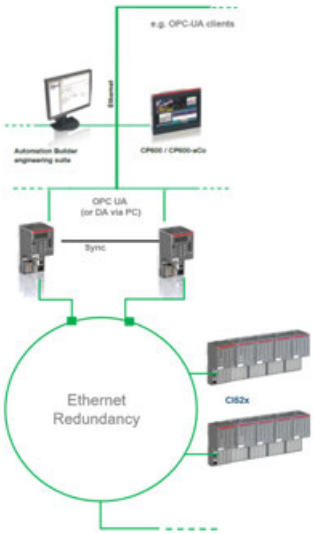
Two same type AC500 PLCs are required as central hardware components. Each PLC is equipped with at least two Ethernet ports at a processor module or at a communication module. The two PLCs, called PLC A and PLC B, are linked by Ethernet to exchange and synchronize information (*Sync*). Connections to the AC500 peripheral field devices (I/O) are performed via Ethernet as well.

For further information on which CPU type and library to be used refer to *Table 379 “Overview of AC500 HA systems and options” on page 2238*.

The following table gives an overview of the different High Availability variants possible with AC500.

The figures are indicative, depend on chosen architectures, system size, network and CPU/CM modules used.

Table 379: Overview of AC500 HA systems and options

Library version		HA-CS31	HA-Modbus TCP					
CPU version		V2 CPUs	V2 CPUs			V3 CPUs		
I/O communication		Parallel serial	Ethernet			Ethernet		
CPUs		PM573 - 595	PM573	PM591	PM595	PM5630	PM5650	PM5670
		Parallel serial	I/O network based on Ethernet and ext.edundancy mechanism					
Max. system size CI52x [1, 6]		3 - 50 [2]	3	< 25 / 50	< 60 / 92	< 30	< 50	< 120
I/O modules		CI590: S500	CI52x: S500 and S500-eCo usable [4]					
Switch-over times	CPU	25 -120 ms [3]	Typically < 50 ms ~[6]					
	Field	15 - 120 ms [3]	Depends mainly on network size, redundancy mechanism of external switches [7]					
SCADA connectivity		OPC DA, IEC60870, …	OPC DA, IEC60870, …			OPC DA, OPC UA, IEC60870, …		
Interfaces		Several CS31 and Ethernet	Several ETH ports, via CM597			2 ETH ports [5] + 1 CAN Interface		
Sync		UDP	UDP			UDP		
Lifecom1		-	UDP			UDP		
Lifecom2		-	Modbus TCP			Modbus TCP / CAN		
Overview of AC500 HA system								

[1] Number of CI52x recommendation based on performance or max. number of sockets (CPU and CM modules).

For more details of Modbus clients supported in AC500 V3 PLCs refer to *Chapter 1.6.1.3.3 “Limitation of connections per protocol” on page 2392*.

[2] Limited by CPU performance, number of CM574 modules number of CS31 clients and process data limits.

[3] Depends on system size and CPU type.

[4] For details on certain S500-eCo modules not supported, see the Automation Builder release notes, Appendix 1.

[5] CM597 not available for V3 CPUs.

[6] Based on HA bits switchover, depending on failure case *Chapter 1.5.8.1.3.2 “Use case descriptions” on page 2245* *Chapter 1.5.8.1.5.2 “Task configuration recommendations for HA system” on page 2258*.

[7] Field network: If CI52x are used with their 2 ports as part of a ring: In the moment of a network switchover single telegrams may be destroyed: - for V2 ETH onboard: Standard TCP delays repeats by 500 ms - for V3 CPUs onboard or V2 CPU using CM597: A special HA-FW ensures fast repeats of typ. ~50ms (settable).

CPU choice, system size and performance indications

The diagrams below indicate the example choices of AC500 CPU's (horizontal axes) based on the number of communication interface CI-remote I/O clusters (Communication Interface modules; numbers see legend) used in a system and resulting application cycle times (vertical axes).

Further details can be found in *Chapter 1.5.8.1.5.2 “Task configuration recommendations for HA system” on page 2258*. The values in below graphs base on the assumption to use max. 50-60% as CPU loading by the bare fast IO communication and HA functionality. So the application load would come on top and cycle times (especially HA, Modbus) need to be relaxed (made higher) compared to below indication.

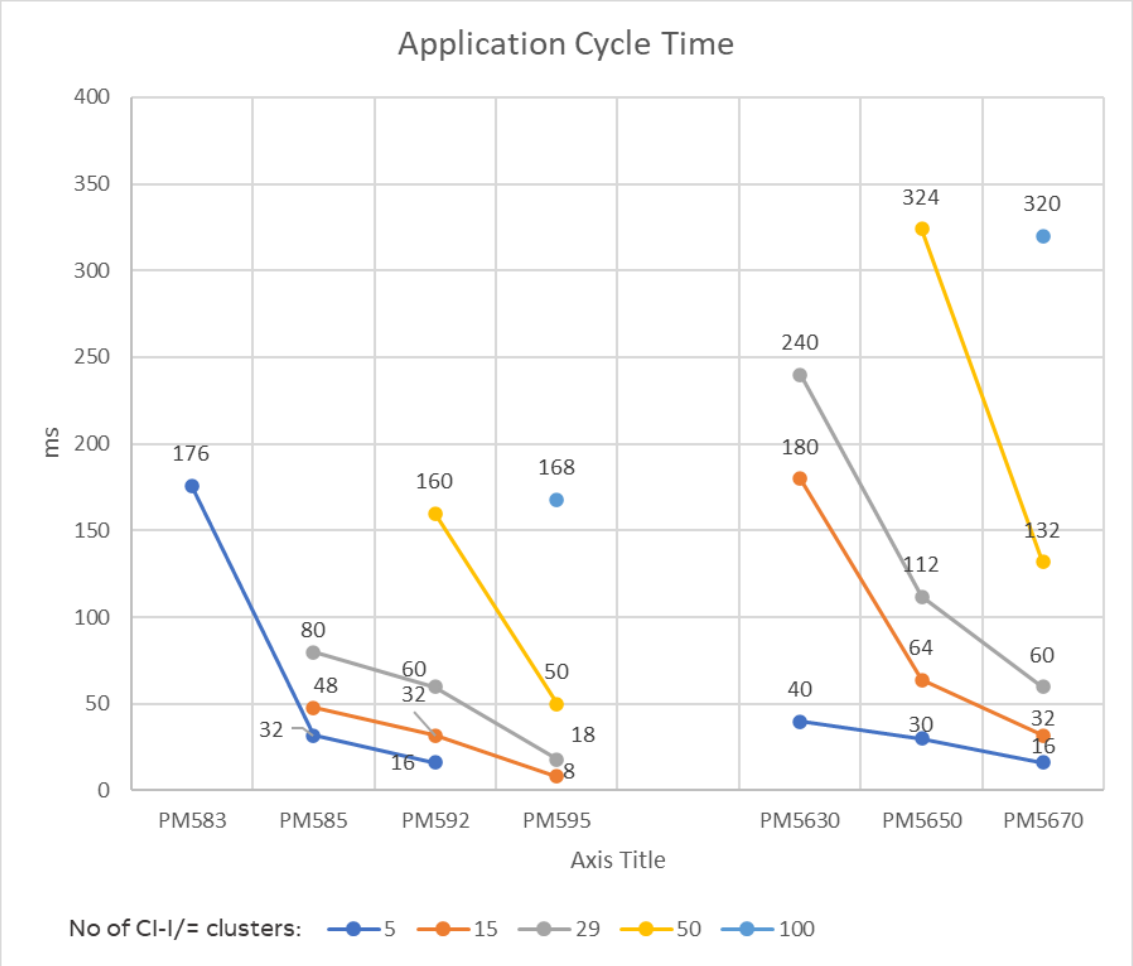


Fig. 37: Indication of AC500 CPU's performance (horizontal axes) based on the number of communication interface CI-remote I/O clusters (Communication Interface modules; numbers see legend) used in a system and resulting application cycle times (vertical axes).

Example: If you need a system supporting min. 25 CI at application cycle time around 120 ms, suitable options based on above graph would be V2 PLCs - PM592 or PM595 and V3 PLCs – PM5650 or PM5670. The main parameter in the application cycle determination is the amount of overall Sync data, which is assumed 160 bytes per CI for the smaller systems, up to 250 bytes per CI for the larger ones. Sync data of the project of in total more than ~1200 byte necessitates several HA cycles to transfer within one application cycle.

The V2 or V3 PLCs types, also differ in available interfaces, protocols supported and memory size.

CI521-MODTCP or CI522-MODTCP can be used as peripheral devices which communicate via the Modbus TCP protocol with the PLCs. The HA-Modbus TCP library supports currently up to 120 CI52x, depending on the CPU type as listed in *Further information on page 2238*. Each CI52x supports up to a maximum of 10 S500-I/O modules. Nevertheless the standard Modbus TCP communication of the HA library transfers only 120 words per cycle: Therefore please check if for your module configuration matches: In case of many analog IO modules with high-density - like 16 channel AI523/AO523 or modules with fast counters - this limit might be surpassed by roughly 5-6 such modules (to help calculate exactly, there is an Excel sheet provided in the HA “Examples” subfolder of Automation Builder once installed).

For more details of Modbus clients supported in AC500 V3 PLCs refer to *Chapter 1.6.1.3.3 “Limitation of connections per protocol” on page 2392*.

Local I/O on a CPU can signal / interact for diagnosis or service with / from this CPU. This local I/O is not redundant and won't be available to communicate to in case of a CPU failure.

Hardware connections

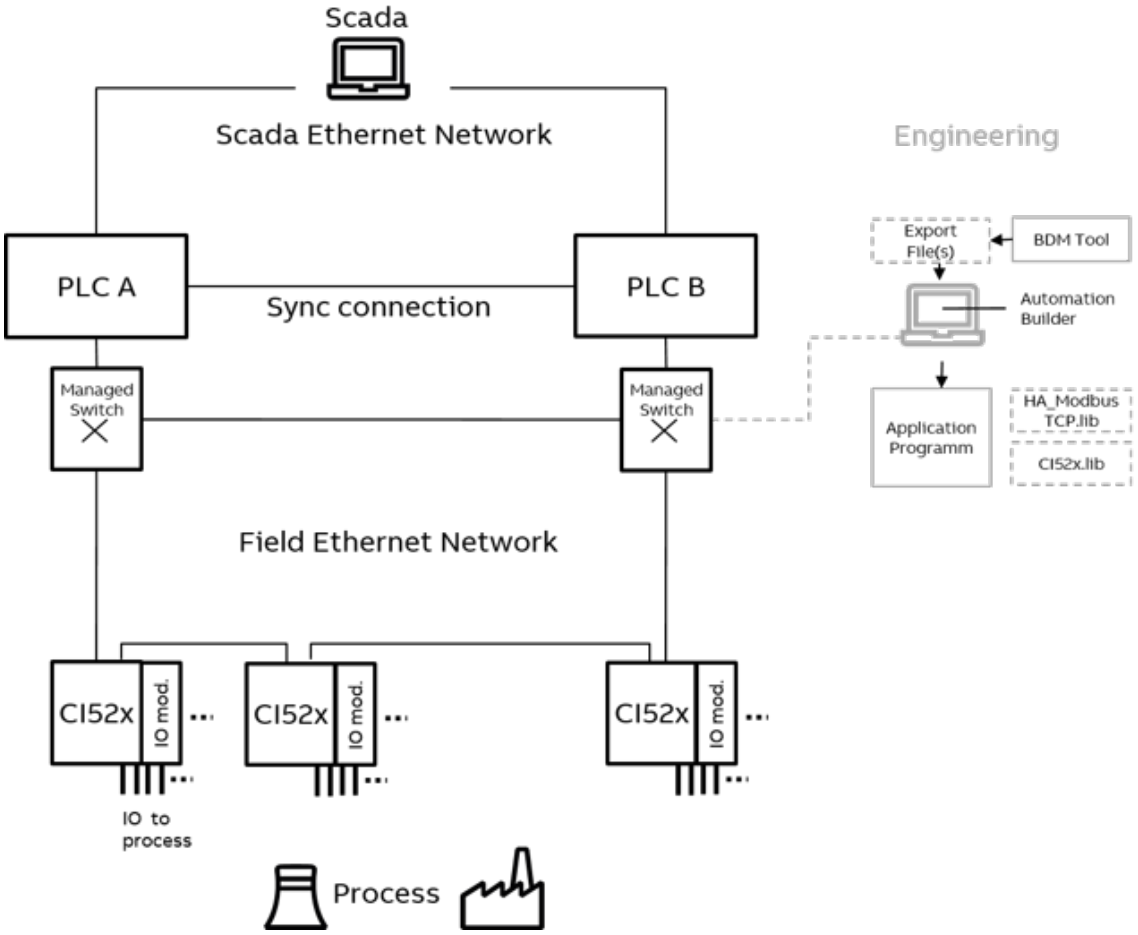


Fig. 38: AC500 HA and SCADA connection

SCADA/ Engineering connection is done using ETH ports of both PLCs and one or several managed Ethernet switches depending on the redundancy requirements in the Ethernet levels.

- HA communication between PLC A and PLC B must be done via two physical connections between PLC A and PLC B in order to distinguish a “sync link” failure from another PLC failure:
  - Sync (including *“lifecom1”*) over Ethernet
  - *“Lifecom2”*
    over Ethernet (Modbus TCP): Can be combined with Field or SCADA network or a separate Ethernet network
    over CAN (only possible with AC500 V3 CPU)
- Field devices (CI52x modules) will be connected via Ethernet switches, forming a redundant network (if requested). For details on network configuration see ↳ *Chapter 1.5.8.1.5.3 “Field I/O network topologies” on page 2260*.

The following table shows possible combinations of connections for different CPU types. There must be at least two physical connections. The availability can be increased with a third physical connection, e.g. CM597 for AC500 V2 CPUs or CAN for AC500 V3 CPUs.

	V2 CPU with 1 ETH onboard + CM597		V2 CPU with 2 ETH onboard (+CM597)			V3 CPU with 2 ETH onboard (+ CAN)			
Scada / Engineering	1	1	1	1	1	1	1	1	1
Sync / lifecom1	1	11	1	2	2	1	2	1	2
lifecom2	11	1	2	1	11	2	1	3	3
Field (Modbus TCP)	11	11	2	2	2	2	2	2	2
# of physical connections	2	2	2	2	3	2	2	3	3

1 ETH1 (orange)
2 ETH2 (green)
3 CAN (blue, applicable only in V3)
11 CM597 communication module at slot1 (grey)
The blue box indicates the example which is used in the next chapters.

The numbers in the figure above define the slot on which the connection is made. Last line # of physical connections define how many physical interfaces are used or connected between the PLCs.

It is also possible to realize an HA system without a communication interface CI module see chapter ↳ *Chapter 1.5.8.1.6.1.1 “Configuration without communication interface modules to establish redundancy” on page 2264*.

Hardware Example

HA hardware configuration based on V3 PLC to explain the minimal recommended Ethernet port configuration.

- The *Sync* connection is performed via SCADA network, the
- *“lifecom2”* is performed via field network (or the other way around).

The following figure represents the connection example with the details from the highlighted box (see previous figure).

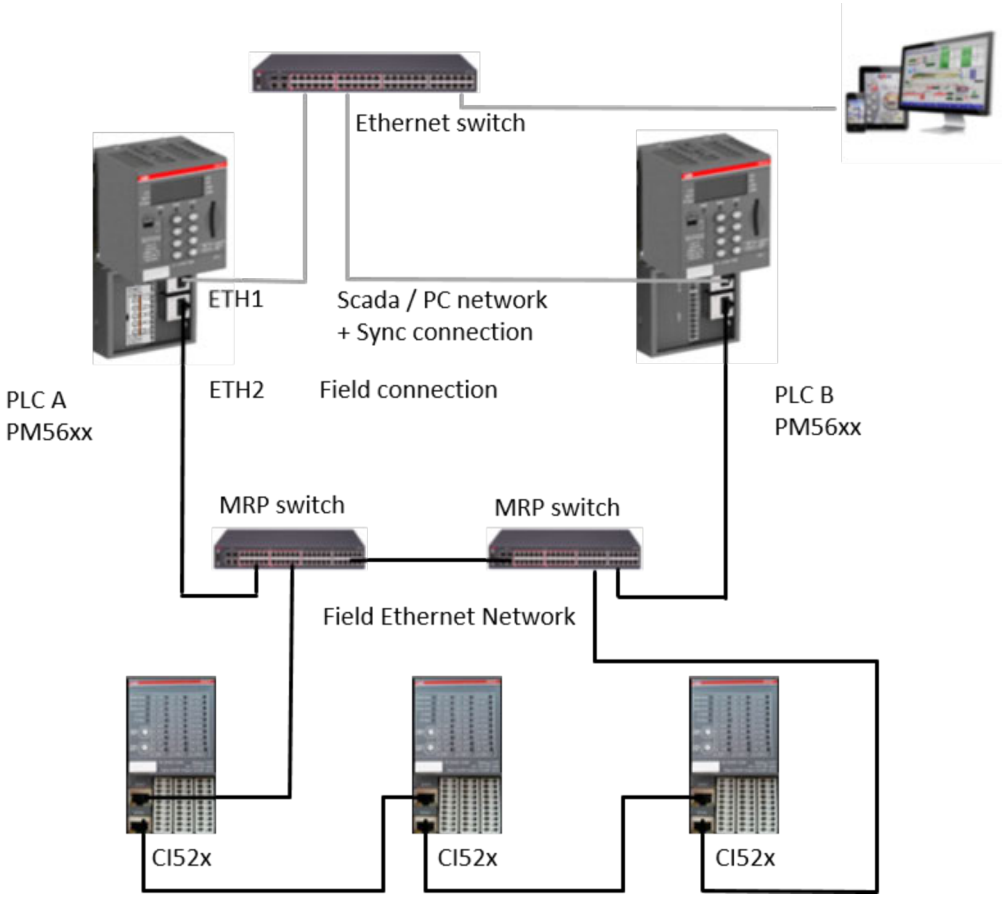


Fig. 39: Physical connection example: 2 ETH CPUs combining sync data / “lifefcom2” with SCADA / Field I/O network

Support of I/O modules (S500/S500-eCo) depends on the version of the library package. See the version details of the library in the Automation Builder release notes.

1.5.8.1.3 Functionality

Failures and use cases

The AC500 High Availability system performs a switch-over whenever the primary PLC is powered off, crashed or stopped or if the primary PLC loses fieldbus communication (cut of ETH or defect MRP switch) while the secondary PLC still has connection.

In the following the different use cases and reaction times are outlined.

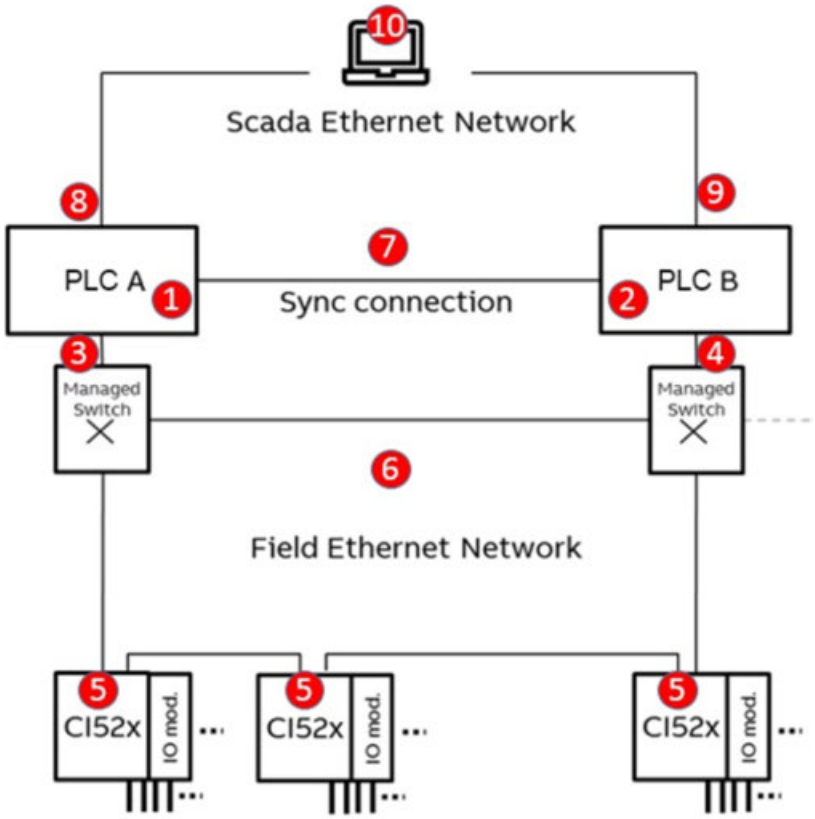


Fig. 40: HA use cases – failures, assuming PLC A is primary and "lifecom2" over field network

The below use case table with reaction and diagnosis messages are based on the setup where Sync is via SCADA network, *"lifecom2"* over field network and PLC A is primary.

Case	Use case	Reaction	Diagnosis message on *)
1	Primary PLC is powered off, crashed or stopped.	Switchover to secondary PLC. CI52x outputs are frozen during switchover period.	Secondary
2	Secondary PLC is powered off, crashed or stopped.	No switchover, process continues.	Primary
3	Primary PLC loses connection to fieldbus CI52x modules while secondary PLC still has a connection.	Switchover to the secondary PLC. CI52x outputs are frozen during switchover period.	Primary
4	Secondary PLC loses connection to one or more CI52x modules.	No switchover, process continues.	Secondary
5	CI52x module is stopped/ powered off.	No switchover, process continues.	Primary and secondary
6	Connection lost in Field Ethernet network.	Depending on Ethernet network structure, and redundancy mechanisms used a reconfiguration time exists.	Lifecom2 lost and CI module lost errors will be generated in primary and secondary.
7	*Sync* and/ or *"lifecom2"* are broken.	No switchover, process continues.	Primary and secondary

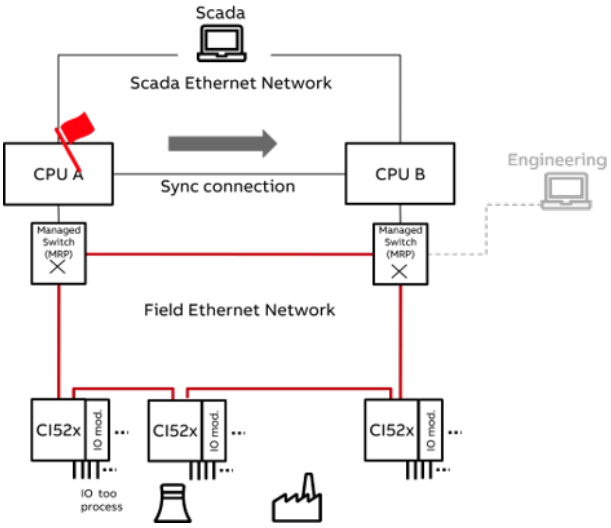
Case	Use case	Reaction	Diagnosis message on *)
8	Primary PLC loses connection to SCADA.	SCADA is responsible to detect and to switch over.	-
9	Secondary PLC loses connection to SCADA.	SCADA is responsible to detect and to switch over.	-
10	SCADA is broken	SCADA is responsible to detect and to switch over.	-
11	Manual switchover by the user.	Switchover to the secondary PLC. CI52x outputs are frozen during switchover period.	-
*) Diagnosis description, see function block description.			

Use case descriptions

The below cases explain the behavior of the system during different use cases.

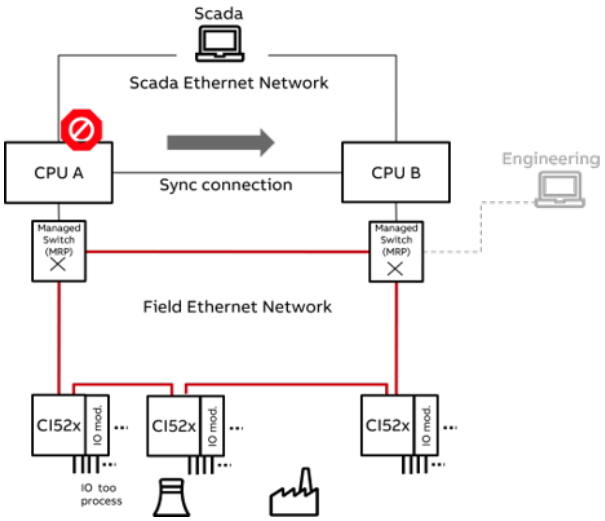
Basic diagnosis information is provided for each case. For diagnosis description refer to *Chapter 1.5.8.1.8 “Diagnosis” on page 2269*.

Case 1 a): Primary PLC is powered off or crashes



Reaction	Switchover to secondary PLC. The communication interface modules are updated by the new primary PLC.
Comment	CI52x outputs are frozen during switchover period.
Diagnosis message on function block	Primary PLC is powered off. Secondary PLC: control block output Runtime Error = 16#001E and xHaModPrimary = TRUE

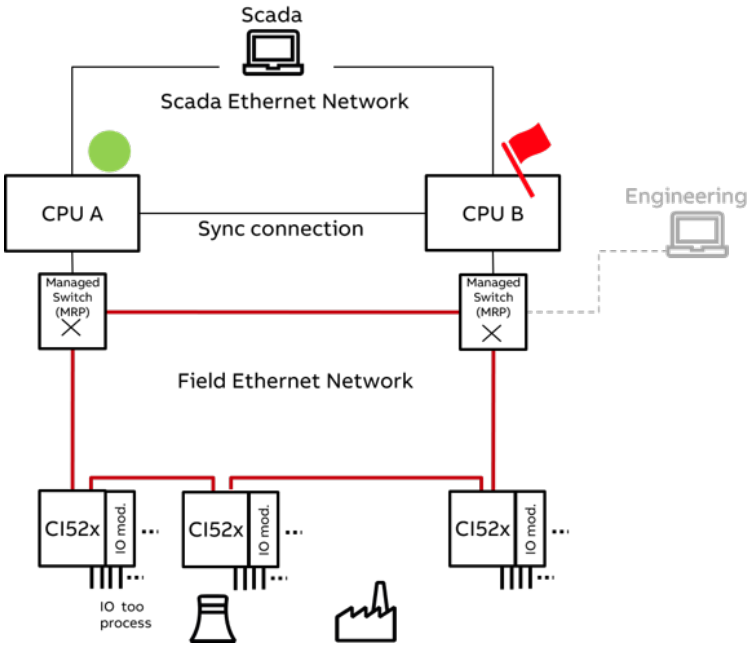
Case 1 b): Primary PLC is stopped



Reaction	Switchover to secondary PLC. The communication interface modules are updated by the new primary PLC.
Comment	CI52x outputs are frozen during switchover period.
Diagnosis message on function block	Primary PLC is stopped. Secondary PLC: control block output Runtime Error = 16#0016 and xHaModPrimary = TRUE

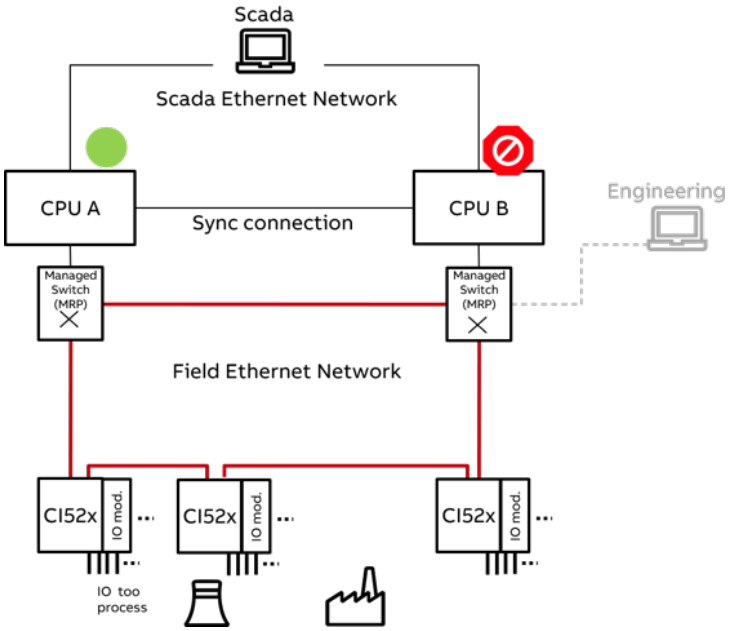
If “lifecom2” is lost and the PLC is in STOP mode RUNTIME ERROR will not be TRUE. This is because Modbus is still responding even if PLC is in STOP mode.

Case 2 a): Secondary PLC is powered off or crashes



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#001E and xHaModPrimary = TRUE Secondary PLC is stopped.

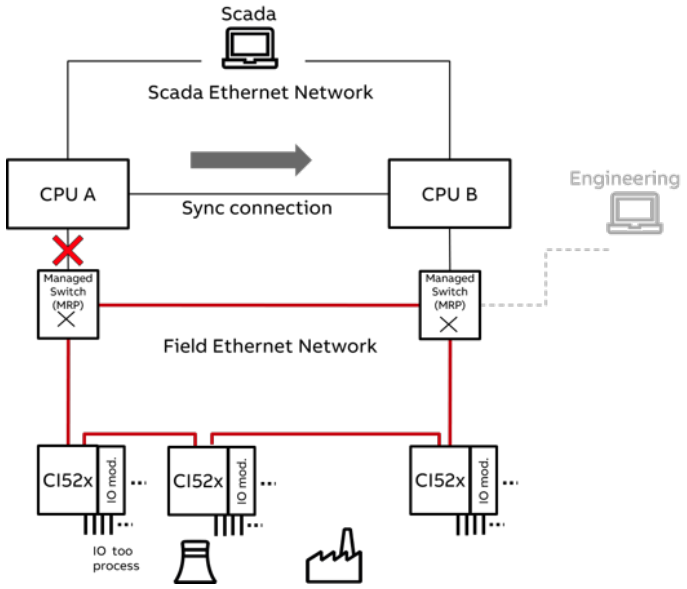
Case 2 b): Secondary PLC stop



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0016 and xHaModPrimary = TRUE Secondary PLC is stopped.

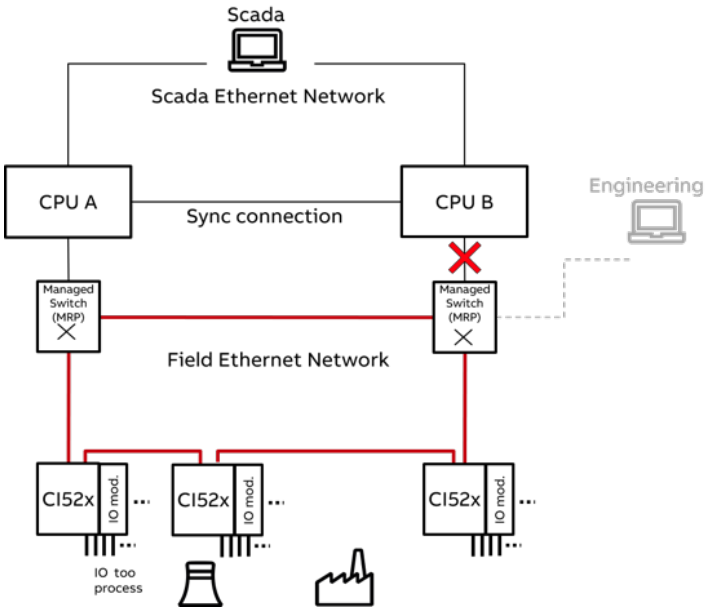
If “lifecom2” is lost and the PLC is in STOP mode RUNTIME ERROR will not be TRUE. This is because Modbus is still responding even if PLC is in STOP mode.

Case 3: Primary PLC loses connection to fieldbus CI52x modules



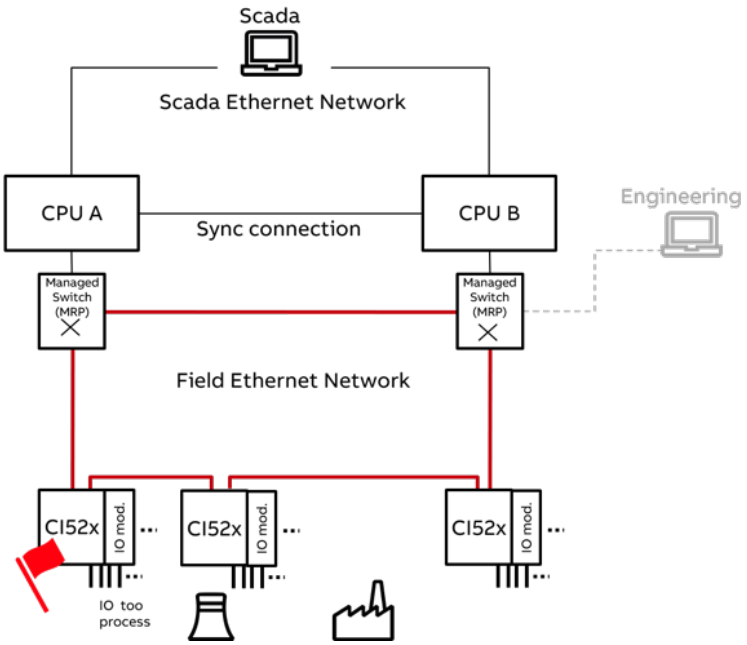
Reaction	Switchover to secondary PLC. The communication interface modules are updated by the new primary.
Comment	CI52x outputs are frozen during the switchover period.
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0094 and xHaModPrimary = FALSE Secondary PLC: control block output Runtime Error = 16#0015 and xHaModPrimary = TRUE

Case 4: Secondary PLC loses connection to fieldbus CI52x modules



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0015 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0094 and xHaModPrimary = FALSE

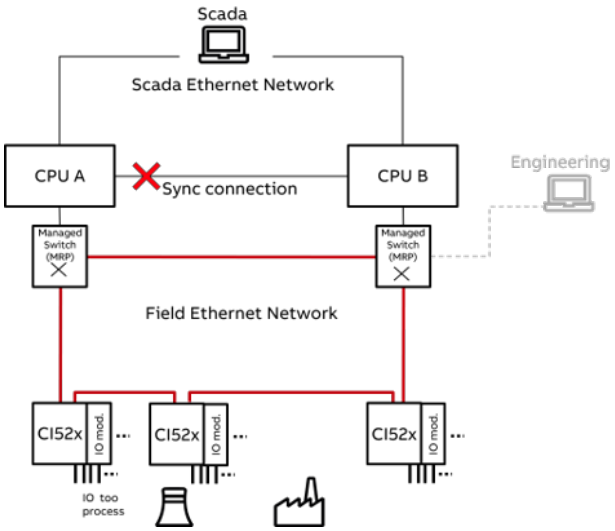
Case 5: CI52x is powered off or stopped



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0081 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0081 and xHaModPrimary = FALSE

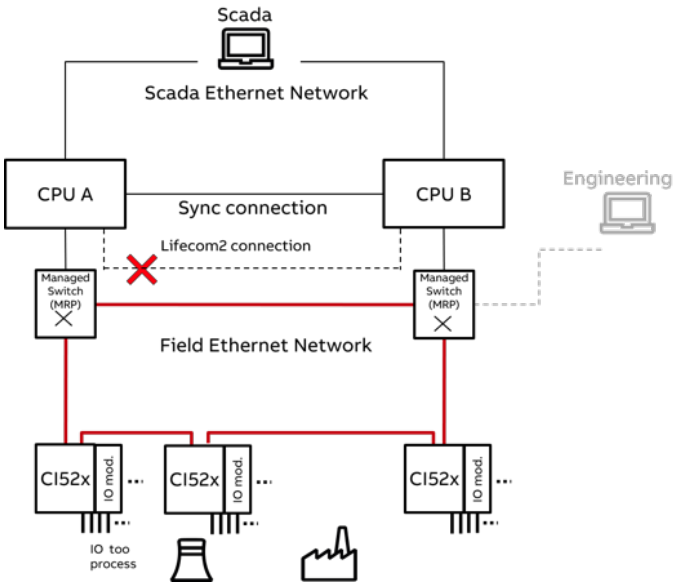
If any CI52x-MODTCP module is powered off and on, there is no need to power restart the complete system. The module will be recognized once the communication is reestablished.

Case 7 a): Sync connection is broken between the PLCs



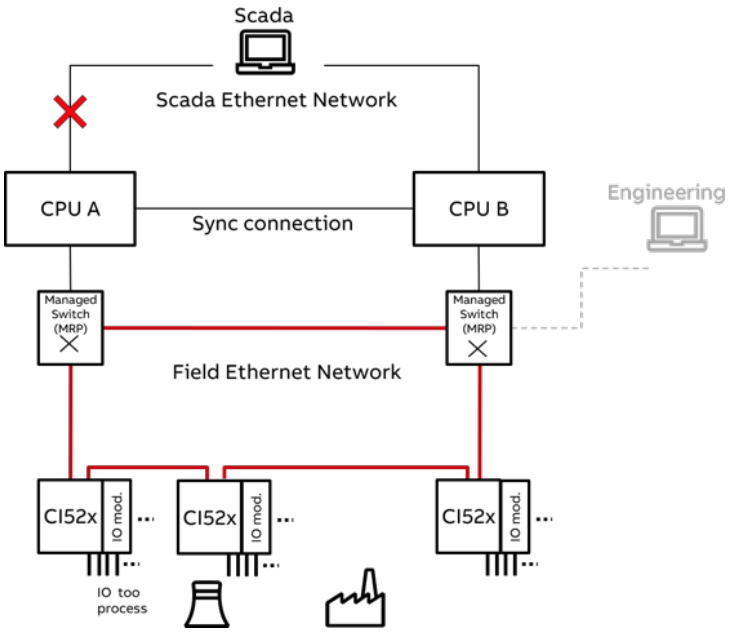
Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0014 / 16#0094 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0014 / 16#0094 and xHaModPrimary = FALSE

Case 7 b): Lifecom2 connection is lost between the PLCs



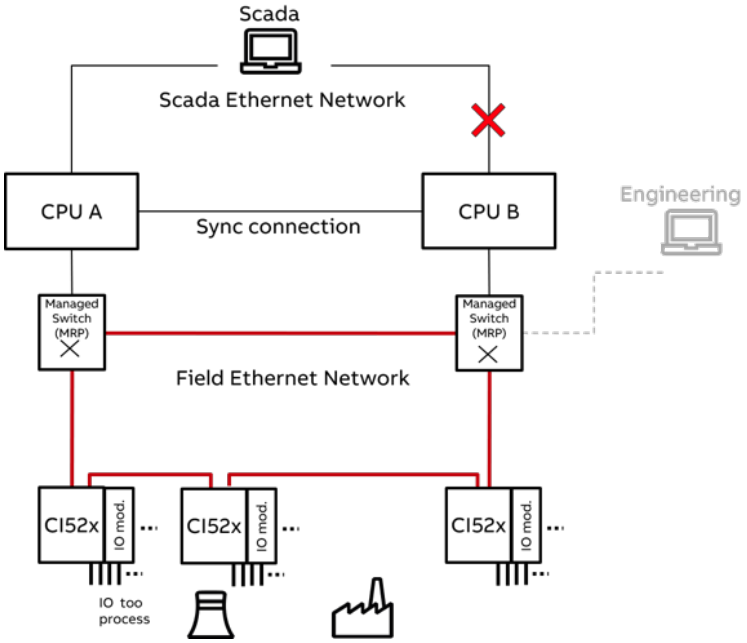
Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0008 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0008 and xHaModPrimary = FALSE

Case 8: Primary PLC loses SCADA connection



Reaction	No switchover
Comment	Process continues, SCADA is responsible to detect and switchover
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = FALSE

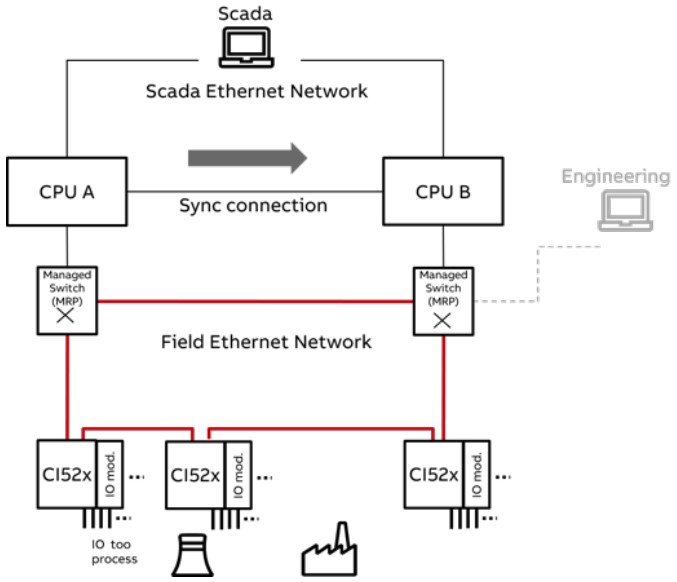
Case 9: Secondary PLC loses SCADA connection



Reaction	No switchover
Comment	Process continues, SCADA is responsible to detect and switchover
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = FALSE

SCADA link may be combined with sync connection or “lifecom2” connection. In that case runtime error and system behavior will be as described in the cases above (Sync connection lost / “lifecom2” connection broken).

Case 11: Manual changeover by user



Reaction	Changeover from primary PLC to secondary PLC.
Comment	CI52x outputs will be frozen during switchover
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = FALSE Secondary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = TRUE

A manual switchover can be triggered from both PLCs. For each trigger a switchover from primary PLC to secondary PLC will take place.

1.5.8.1.4 How to get and install the AC500 High Availability system package

The PS5601- High Availability Modbus library package can be installed from the *Automation Builder Installation Manager* by selecting the component.

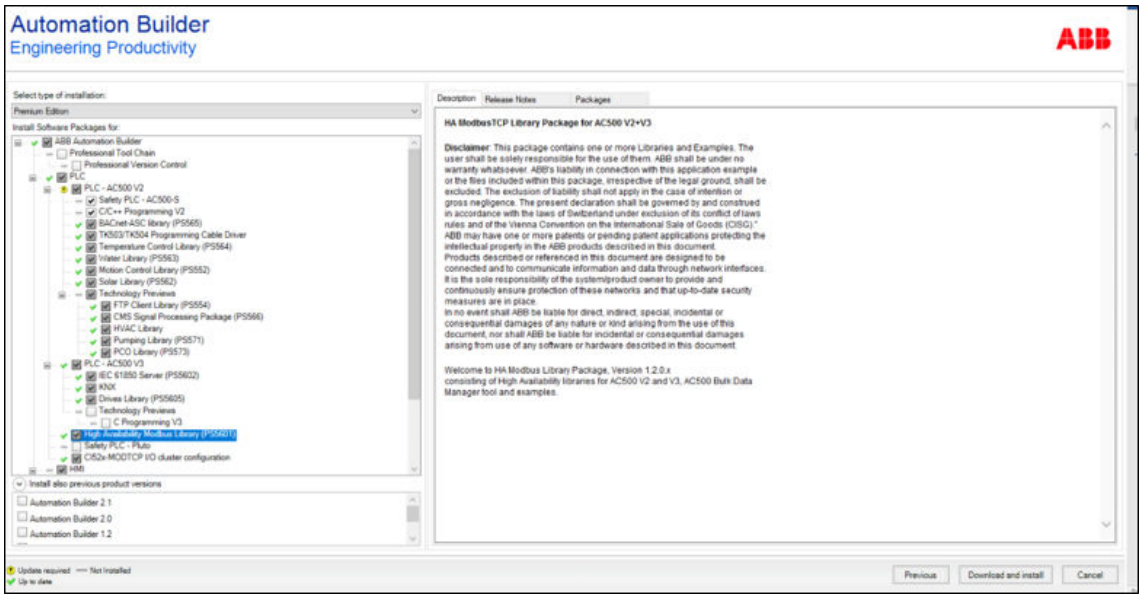


Fig. 41: Automation Builder Installation Manager

The following components are installed:

- Libraries
  - AC500 V2 libraries: *C:\Program Files (x86)\Common Files\CAA-Targets\ABB_AC500\AC500_V12\library\PS5601-HA-MTCP*
    CI52x_AC500_Vxx.lib, HAModbus_AC500_Vxx.lib.
  - AC500 V3 libraries available in library repository:
    ABB_CI52x_AC500.compiled-library, ABB_HaModbus_AC500.compiled-library
- Online help: HA-CS31, HA Modbus V2 function block description
- Automation Builder Example folder: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP*
  - AC500_V2: Examples for AC500 V2 including documentation
  - AC500_V3: Examples for AC500 V3 including documentation
  - BulkDataManager: Bulk Data Manager (BDM) tool which helps efficient engineering in larger projects. This requires a separate installation. Further information can be found in the document: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP\BulkDataManager\Documentation*.
  - HA-Modbus TCP System Technology.pdf (this document)

1.5.8.1.5 System structure

This chapter explains the detailed structure of the HA system in CODESYS. A HA-Modbus TCP system is characterized by two AC500 PLCs with the following features:

- Identical programs (application with additional HA and Modbus function blocks) that are loaded to both PLCs.
- Communication interface modules CI52x-MODTCP that are connected via Modbus TCP.
- Synchronization of both PLCs (*sync/lifecom1* and *lifecom2* logical connections).

Programming

Each PLC contains at least three main tasks/ programs:

- HA program
- Application program
- Modbus program

The programs in one PLC communicate via internal structures of the libraries and dedicated internal memory areas for HA-Sync array and the Modbus CI52x memory(ies) CiModDataxx.

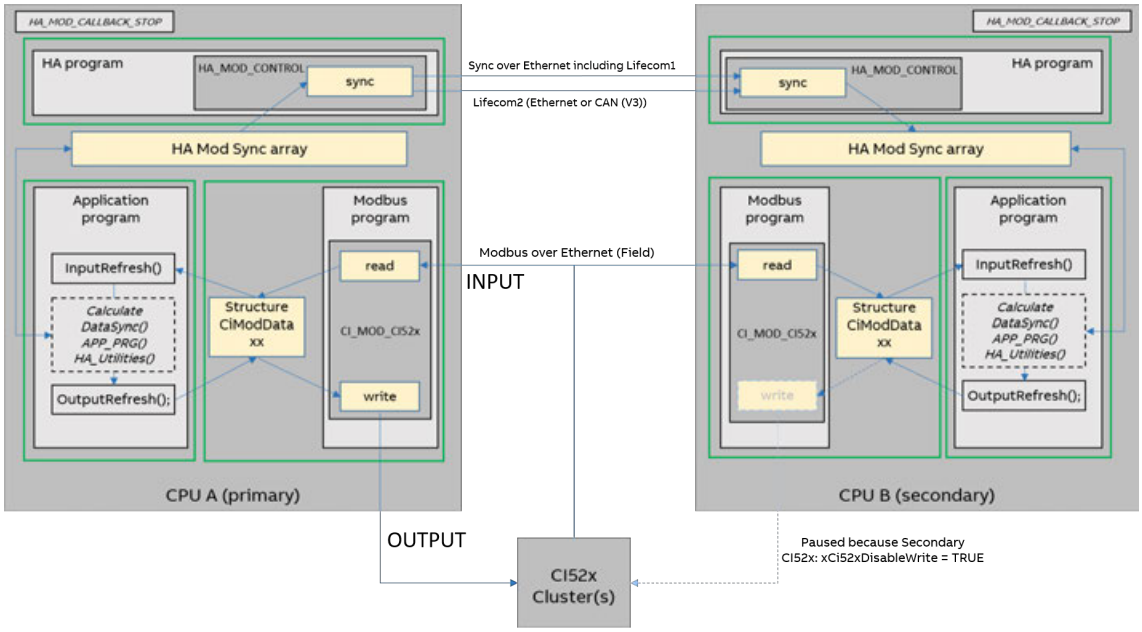


Fig. 42: Principle structure of the HA system and recommended tasks: HA, Modbus, Application

Table 380: Image description

Layout element	Meaning
Dotted outline box	Indicates optional function block or programs.
Solid outline box	Indicates the mandatory function blocks or programs. All mandatory blocks are called when an export is created from Bulk data manager.
Italic font	Indicates the program or functions user should call in his project and not created by Bulk data manager.
Light yellow background block / blue arrow	Indicates the operations which are handled internally in the library.
Green solid box	Indicates the three different tasks which user has to configure.

Modbus program

The function block CIModCI52x (V3) / CI_MOD_CI52x (V2) reads the input values from the CI52x modules and stores them in the structure CiModDataxx. If the CPU is primary it also writes the outputs to the CI52x modules. The Function block also parametrizes the CI Module as configured in e.g. Bulk data manager tool during the first startup or when a CI module is exchanged.

Normally the HA-Modbus TCP library takes care of communication monitoring. Nevertheless if com- munication is cut completely, the CI52x communication interfaces and its I/O modules have to react on their own to achieve a bumpless or desired behavior: The following parameters for the CI52x com- munication interfaces and I/O modules need to be considered:

- *– CI52x: parameter "Timeout" for Bus supervision: [2)]*
  Allows to detect errors from communication interface side as well and take action to ensure a fail- safe behavior if communication is cut. It can be set in 10 ms steps. If set to 0 no bus supervision is active. Proposed value: 50 = 500 ms = default in Bulk data manager; this value should be increased, e.g. to value 65 if AC500 V2 CPU ports are used for field communication to take care of the larger TCP retransmit time.
- *"Behaviour Outputs" at "Timeout for Bus supervision" [1), 2)]. This fail-safe parameter has to be con- sciously set: separate settings are possible for each module (and communication interface): "off"; "last" or "substitute": 5 s, 10 s, ∞ s [1)].*

Remarks:

[1)] The parameters "Behaviour Outputs at comm. Error" is only analyzed if the Failsafe-mode is [ON].

[2)] Both are CI52x parameters set e.g. via Bulk data manager tool in the program.

Application program

- At the start of the application task the InputRefresh program has to be called. It copies data from Modbus via the structure CiModDataxx to the user variables, which were defined in BDM as signals. For further information refer to BDM documentation, chapter 7 which is available in the path: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP\BulkDataManager\Documentation*.
- Only the main application programs should be in this task and use these variables for the user defined functions. E.g here the user programs and logic should be called and use the HA libraries utility blocks (which *sync* their historic data automatically) and HA_MOD_DATA_SYNC blocks for further user data which should be synchronized.
- Data of utility blocks and HA_MOD_DATA_SYNC blocks are copied to the HA Sync array of the primary CPU (which is sent to the secondary CPU by the HA program).
- OutputRefresh program is called as a last step. It copies data from the user variables via structure CiModDataxx to Modbus.

Example of a utility function block (with integrated sync data)

Consider the on-delay timer HA_MOD_TON (V2)/ HaModTon (V3).

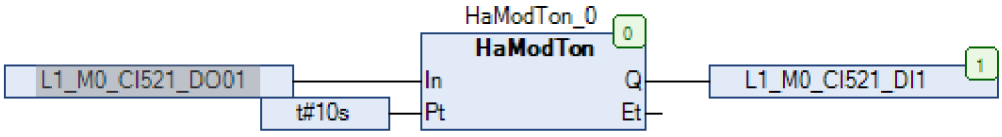


Fig. 43: HaModTon utility function block with internal synchronization

Both PLCs require the same function block called in the program. Under normal operating conditions the elapsed time ET and output Q of the timer is synchronized internally from primary to the secondary CPU. ET and Q data are available and can be attached to local or global variables in the program as per application requirements. If PLC A shuts down due to a fault, the primary status switches over to PLC B.

In the event of a switchover, the moment PLC B becomes the primary, the timer on this PLC will keep running. Until the time of PLC A failure, the timer on PLC B was synchronized. This is most important in cases when one CPU was not in run or off and needs to “catch up” such integral or historic system values (timers, counters, operator settings, …). The actual process remains then unaffected by the switchover.

HA program

HA_MOD_CONTROL has two functions:

- Exchange status data (lifecom1 and lifecom2) and switch from secondary to primary PLC (or vice versa) based on the status according to the use cases described in *Chapter 1.5.8.1.3.1 “Failures and use cases” on page 2243*.
- Send sync “HA SYNC” array from primary to secondary PLC to ensure that the secondary PLC is always in hot-stand-by and can take over immediately. UDP protocol is used for data synchronization between the CPUs.

Data synchronization via UDP

This chapter explains how the data synchronization happens between primary and secondary PLC via UDP.

All prepared sync data is synchronized with the secondary PLC. Typically only integral values (timers, counters, PID, …) or settings which might have been received have to be synchronized. For example for fast start-up cases when a secondary CPU was restarted, as both PLCs are running and calculating closely in parallel and based on the same input values, synchronization will make the secondary start with current value instead of default value. For details on how to configure or use the data sync function block refer example projects.

Following steps are performed:

- HA SYNC array is transferred via UDP to the secondary CPU. This includes the exchange of lifecom1 status between primary and secondary CPU.
- In the HA program the HA_MOD_CONTROL function block collects all diagnosis, sync and lifecom2 data from the field and/ or the other PLC. Whether a switchover is necessary is decided based on a simple decision matrix.
- Lifecom2 is exchanged between CPUs over Modbus TCP every cycle.
- One task per program, see figure above.
- Status of the inputs connected to CI52x decentralized I/O stations is transferred to both PLCs simultaneously in every PLC cycle. They are received by the CI52x function block.
- At the end of the program, the generated output values are sent, by transferring from the primary PLC respective buffers to the CI52x-MODTCP module(s) via CI52x function block and Modbus TCP. The secondary PLC is prepared to send but stays “silent” (not sending output values).

PLC needs one HA cycle to send one ETH frame data from primary to secondary CPU and receive acknowledge from secondary CPU. Similarly V3 PLC needs two HA cycles.

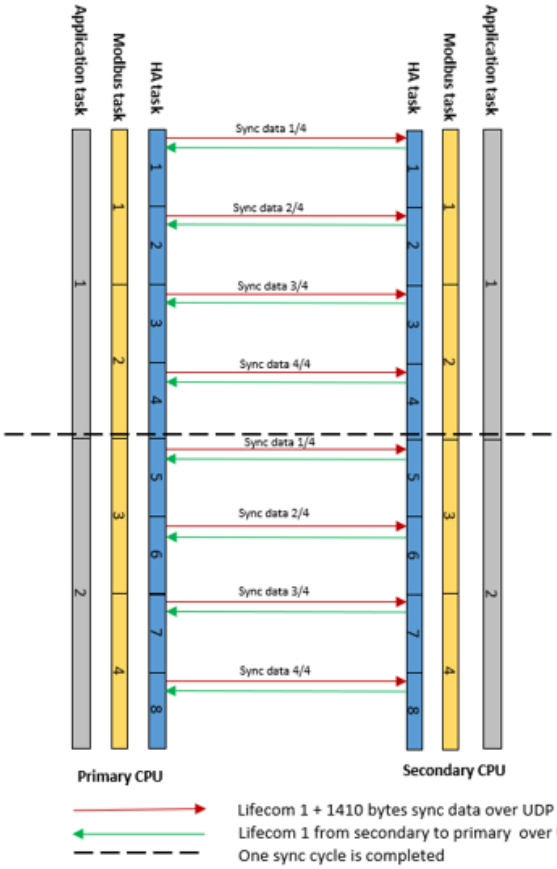
One ETH frame copies approx. 1412 data bytes. The number of ETH frames needed to synchronize HA Sync Array completely depends on the number of data sync bytes. Global variable *iNoOfEthFrames* gives the user this information, which should be used to calculate the cycle time for the application task.

Chapter 1.5.8.1.5.2 “Task configuration recommendations for HA system” on page 2258

Up to max. 60 kB of Sync data can be synchronized.

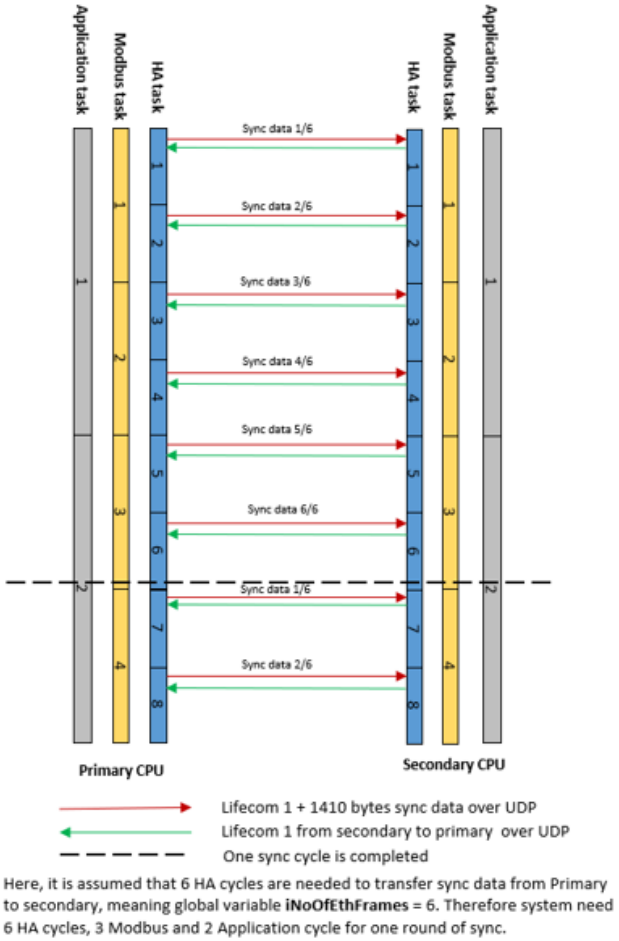
Synchronization between the primary and the secondary PLC happens over a few cycles of HA task time depending on the total sync data bytes configured in the system. Lifecom1 is also exchanged between the primary and the secondary PLC. The primary PLC sends lifecom1 to the secondary PLC along with sync data. Backwards the secondary PLC sends lifecom1 to the primary PLC every cycle.

The following figures shows an example for V2 PLC. When in the project the sync data is equal to 4 *iNoOfEthFrames* then it takes 4 HA cycles to synchronize the data between the PLCs.



Here, it is assumed that 4 HA cycles are needed to transfer sync data from Primary to secondary, meaning global variable **iNoOfEthFrames** = 4. Therefore system need 4 HA cycles, 2 Modbus and 1 Application cycle for one round of sync.

When sync data in the project is equal to 6 *iNoOfEthFrames* then it takes 6 HA cycles to synchronize the data between the PLCs.



Here, it is assumed that 6 HA cycles are needed to transfer sync data from Primary to secondary, meaning global variable **iNoOfEthFrames** = 6. Therefore system need 6 HA cycles, 3 Modbus and 2 Application cycle for one round of sync.

Task configuration recommendations for HA system

For a balanced performance of the HA system consider the following recommendations in your project task configuration:

General

- Use the real time priorities for all HA related tasks. The HA program/ task should be called at highest priority as it is responsible for the core HA functionality and should be the fastest task.
- The Modbus task contains the Modbus communication function blocks at lower priority and (depending on CPU performance) also a faster cycle time to ensure sufficient update rates on Modbus without over- loading the CPU with communication.
- The application program parts should be called in the application task with even lower priority and a larger cycle time than above tasks.
- Configuration to improve standard Modbus TCP for a fast switch over between PLCs.
- AC500 V2
  - CM597ETH_SET_TCP_RTO function block from CM597_ETH_AC500_V28.lib needs to be called inside HA task. User needs to call this function block for each CM597 module connected. For recommended values see example description.
- AC500 V3
  - RTO retransmission time function block “EthSetRtoMin” for the ETH port where fieldbus communication is configured. By default, minimum retransmission time configured is 15 ms.

Task	Priority	PM57x, PM58x, PM59x	PM595-4ETH	V3 PLCs
HA	10 (high)	4 ms or higher	2 ms or higher	4 ms or higher
Modbus	11 (medium)	Maximum of (HA cycle time *2), (3 ms + roundup (#CI/2))	Maximum of (HA cycle time *2), (3 ms + roundup (#CI/2))	Maximum of (HA cycle time *2), (3 ms + roundup (#CI/2))
Application	12 (low)	Maximum of (Modbus cycle time *2), (iNoOfEthFrames * HA cycle time)	(iNoOfEthFrames * HA cycle time)	Maximum of (Modbus cycle time *2), (iNoOfEthFrames * HA cycle time *2)

Procedure for task configuration

1. Choose suitable CPU type according to chapter CPU choice, system size, performance indications
2. Configure task priorities according to the table
3. Set HA task to minimum according to above table
4. Calculate Modbus cycle time according formulas in the table, based on HA cycle and number of CI modules “#CI”
5. Calculate Application cycle time according to formulas in the table, based on Modbus cycle time and variable iNoOfEthFrames, which is defined in the global variables of HA-Modbus TCP library.
6. Measure PLC and CPU load during trial operation.

   V3: PLC Utilisation: *Chapter 1.6.5.1.2.1.4 “PLC utilization” on page 3470*

   If the PLC load is higher than 40 % or CPU load higher than 60 % then increase HA cycle time (e.g. to 8 ms / 12 ms / 24 ms, …) and go to step 4, repeat the steps until loading is within defined range.

> *A new **V3 CPU configuration option** is introduced from Automation Builder 2.4.1 and onwards which allows to change the priority for Ethernet communication in PLCs.*
>
> *Set this configuration in the device tree of the CPU in Automation Builder double click on PLC “CPU_Parameters Parameters ➔ Communication Schema ➔ Select “Onboard Ethernet””.*
>
> *The above parameter should be set to “Onboard” Ethernet for HA systems and it will consequently increase the loading due to the higher priority. PLC Load < 50 % and CPU load < 70 % should be considered as guidelines here instead, while setting the task times while setting the task times.*

7. Following timeout values has to be defined in the user project according to the relation defined.

Timeout variables (see definitions in box below table)	HA in V2	HA in V3
timCI52xTimeOut	1 * Modbus Task time	50 ms or Modbus Task time, whichever is higher
timHaModSyncTimeOut	1* HA Task time	2 * HA Task time
timResponseTimeout	Not applicable	50ms or (2 * Modbus Task time), whichever is higher
timCanTimeOut	Not applicable	100 ms or (2 * Application Task time) whichever is higher

8. Add additional applications and SCADA communication: Check PLC and CPU load again vs. your requirements.

In the HA Modbus system different timeouts must be configured for the fine operation of the system as described above in the task configuration for V2 and V3 PLCs. These different timeouts meaning, and relation is explained below:

timHaModSyncTimeOut:

Time limit to check if the new sync data is received or not in the secondary PLC. If this timeout is not defined properly, Sync lost error/ “lifecom1” lost error will be generated.

timCanTimeOut:

Time used for the check whether “lifecom2” is received when configured via CAN. This value is applicable only in AC500 V3. Lifecom2 via CAN won’t be stable between the PLCs and runtime error "lifecom2 lost" will be flickering if not the right value is configured.

timCI52xTimeOut:

Time limit to check whether new data is received in the Modbus field modules. It is also used to check whether “lifecom2” is received when configured via Modbus TCP. If 'timCI52xTimeOut' is not defined as described, “lifecom2” error / communication interface diagnosis error will not be generated as expected.

timResponseTimeOut:

Timeout value to check whether CPU has lost the communication interface modules connected in the network. If this value is not defined as described, communication interface module lost detection will not be indicated properly.

Field I/O network topologies

Modbus TCP communication between PLC and communication interface modules CI521-MODTCP or CI522-MODTCP can be done using different network topologies. In the following subchapters different simple combinations with their pros and cons are explained.

If a CI52x module of a daisy chain is powered off, next following modules will lose connection/ data provided there is no redundancy in the Ethernet network (e.g. ring and managed switch).

Simple ring topology (smaller systems)

In a simple configuration, CI52x modules can be part of a ring if MRP (or DLR) protocol is used in the managed switches. Then the CI52x are connected from one to another device (“daisy chained") through e.g. two network switches. The redundancy protocol detects a closed ring and opens one port of a managed switch to avoid the ring. The user has to configure the necessary ring configurations and enable the ring manager for the used ring ports in one switch.

It is recommended that time interval between ETH cable disconnection and re-connection should be greater than 2-3 seconds.

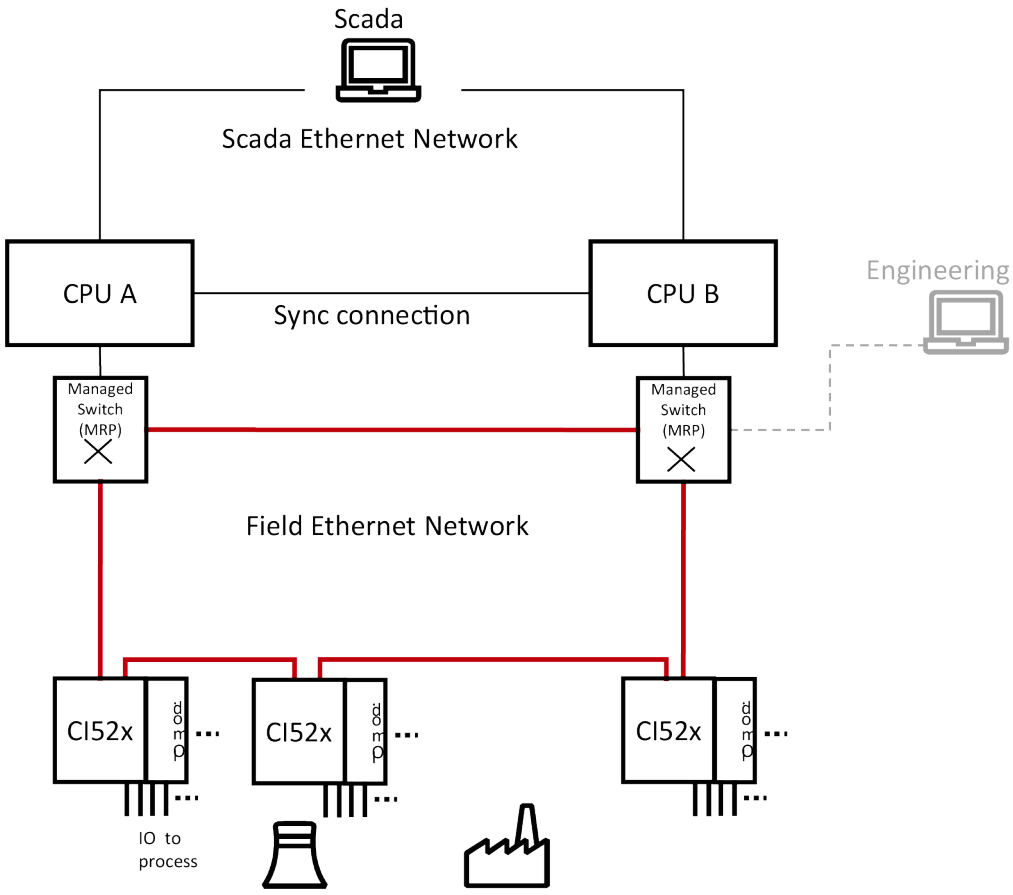


Fig. 44: Redundant ring topology with 2 MRP switches (avoids a SPOF (Single Point Of Failure))

Standard network topology (large systems)

In the standard redundant network, which is often done by third party dedicated telecommunication companies, managed switches are used for every connection point to this network. It’s the network's (and operator's) responsibility to repair any failure fast enough so that no influence on the HA system or its outputs occur.

The network can use other fast redundancy algorithms, also having other than ring structures, if redundancy links are activated fast enough.

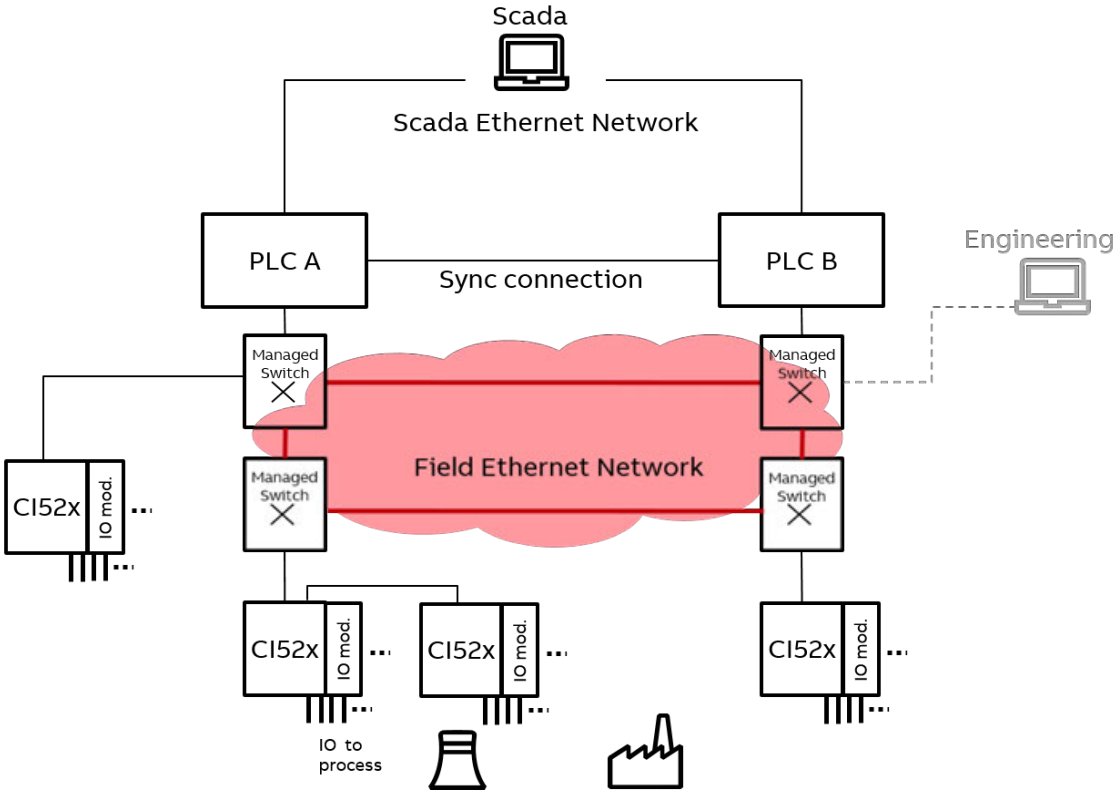


Fig. 45: Redundant ring topology with independent network (using any fast redundancy mechanism internally in a ring or meshed network)

Parallel network topology (using PRP)

Each CI52x module and PLCs as single ended devices are connected by PRP switches to both networks. Here the failure of the switch which connects the primary CPU will also lead to a switchover.

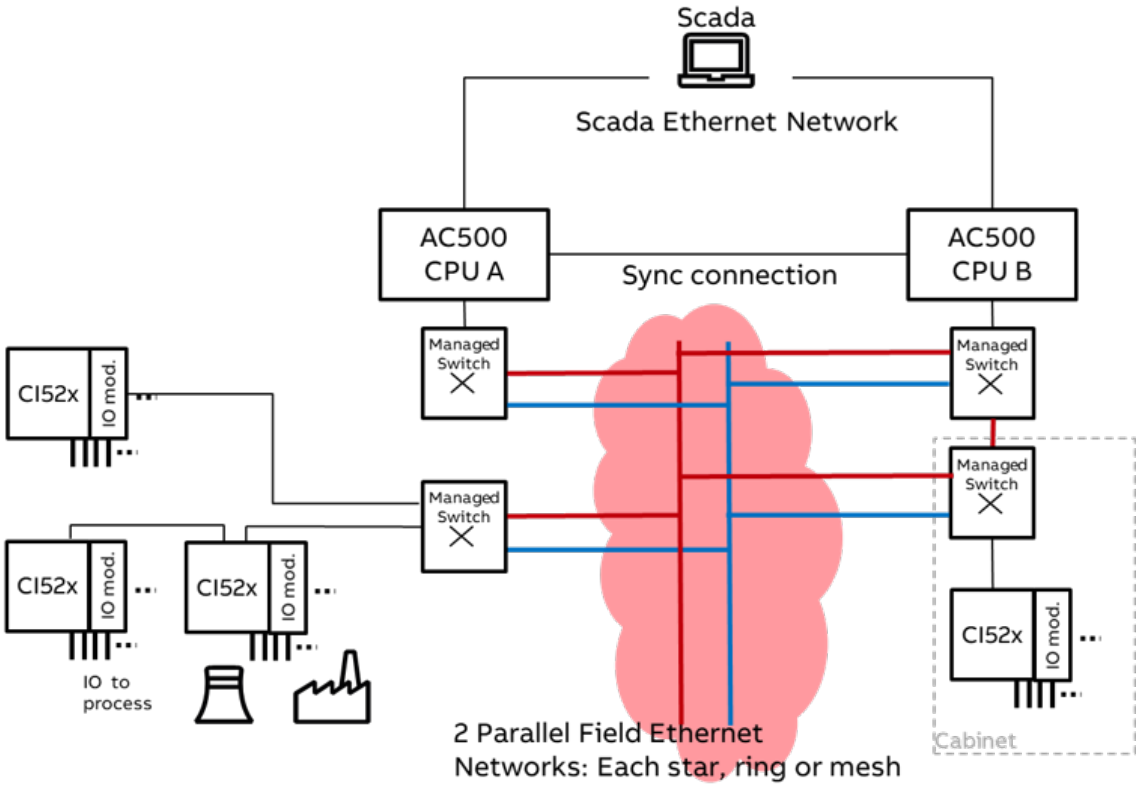


Fig. 46: Parallel-redundant network, PRP switches to connect each device, CPU and communi-cation inter- face modules

Redundancy switchover timing should match the settings in the program and communication interface modules for time-out and freeze periods. The networks for larger systems are often seen as a separate entity and done by a separate company. Make sure to have the redundancy status information of the network at least in SCADA, to repair in time. If the I/O field network responsibility is with the automation/ PLC part, the redundancy status should be also monitored by the PLC. A warning to initiate repair may be created from the managed switches in the I/O field network.

Examples

- Alarm output(s) wired (e.g. to a CI52x input and related settings of the switch(es)).
- Settings of the switch(es) to send (e.g. SNMP traps, which can be received in PLC (AC500 SNMP library)).
- Use of “automation switches” which can also communicate their status directly via Modbus.

It is also possible to connect switches in ring combination with CI modules connected to them in daisy chain. User needs to do the relevant setting based on type of switch and protocol (Ex: MRP, RSTP).

If RSTP ring configuration is used in the system, ring reconfiguration time is slower than other ring protocols. During this reconfiguration, connection to the CI modules will be lost.

HA Modbus system without communication interface modules in the network

It is also possible to have a HA Modbus system without connecting any field devices, CI521-MODTCP / CI522-MODTCP in the network. This system can be used for establishing a redundant PLC system with data synchronization between two AC500 controllers, either without field IO or with user integration of other protocols to field-IO or “intelligent” IO: CPUs as field devices.

Secondary will be on hot standby with primary PLC, during a power off/ stop of the primary PLC. Secondary will take over the control and continue the process. Any user integrated field-IO or CPUs can establish communication mapped with the primary bit: parallel reading but prevent parallel writing.

HA without CI modules can be also used during commissioning to check the data sync, OPC and SCADA related communications without any field devices configured. The user has to set the global variable 'xNoCiBus' to TRUE defined in the HA_GLOBAL_VARIABLES. This variable has to be set to TRUE in both PLCs. Note: It is not advised to update this variable during run time.

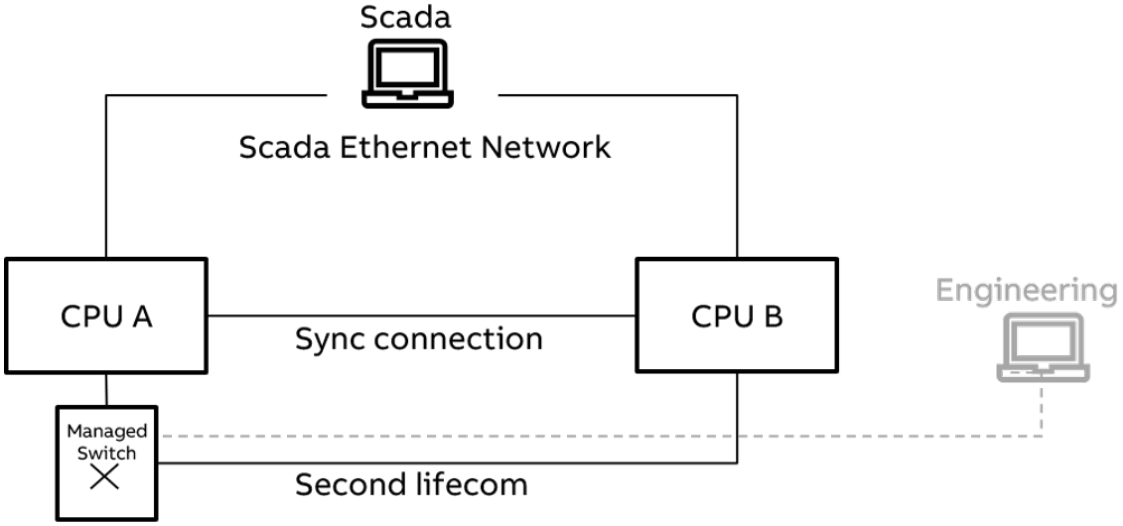


Fig. 47: Simple SCADA connection

1.5.8.1.6 Getting started

Quick start list and guidelines

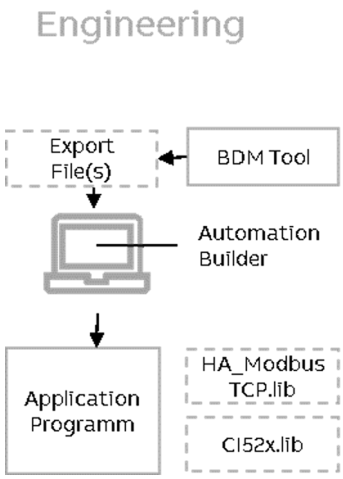


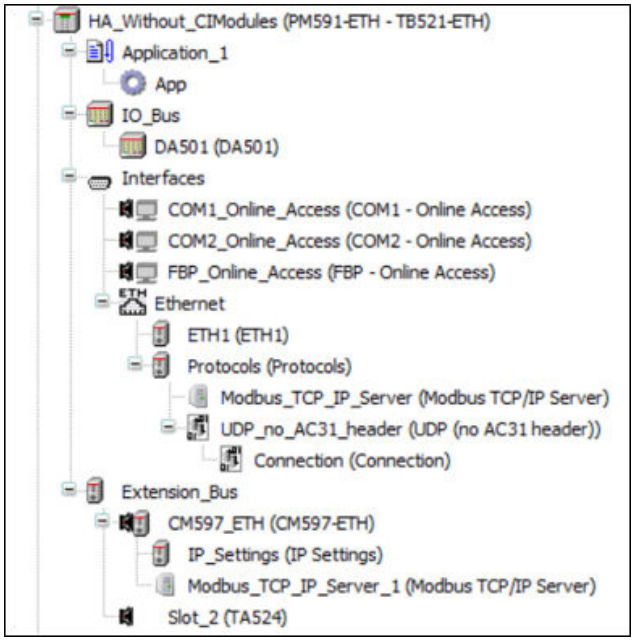
Fig. 48: Engineering workflow using Automation Builder

Simple steps to engineer the HA Modbus system is explained in the following chapters.

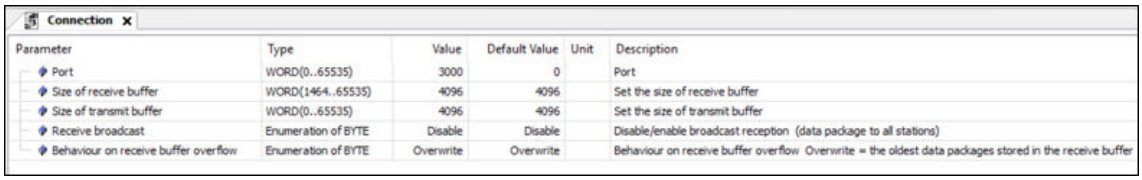
Configuration without communication interface modules to establish redundancy

Configuration of the HA system without communication interface modules to establish redundancy is done by the following steps (for details see the example documentations):

1. Install the hardware ↳ *Chapter 1.5.8.1.2 “Hardware, requirements and options overview” on page 2237*.
2. Select the CPUs based on the requirements ↳ *Chapter 1.5.8.1.2.1 “CPU choice, system size and performance indications” on page 2239*.
3. Install Automation Builder including the latest libraries ↳ *Chapter 1.5.8.1.4 “How to get and install the AC500 High Availability system package” on page 2252*.
4. Create a new project in Automation Builder for the chosen CPUs.
5. Configure the required Modbus and UDP configuration in the Automation Builder device tree of the CPU.

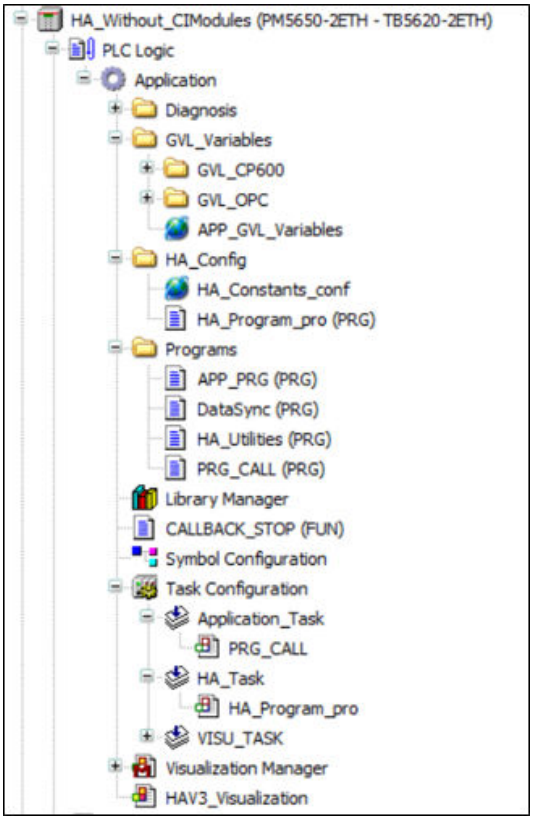


6. For UDP in AC500 V2 PLC, configure *“UDP_no_AC31_header”* and set the port number to value '3000'.



7. Assign the IP addresses in ≥ 2 different Ethernet networks:
   - SCADA network: SCADA, connected PLC A and PLC B
   - Field network: connected CI52x module(s)
8. Configure the mandatory HA_MOD_CONTROL function block for the HA task ↳ *“HA program” on page 2256*.
9. Add Callback stop function HA_MOD_CALLBACK_STOP and call it in the system event “stop”.
10. Add optional HA utility function blocks or function block HA_MOD_DATASYNC.

11. Make the global variable xNoCiBus = TRUE to run the system without communication interface module configured in the system. Refer to *Chapter 1.5.8.1.5.3.4 “HA Modbus system without communication interface modules in the network” on page 2263*.



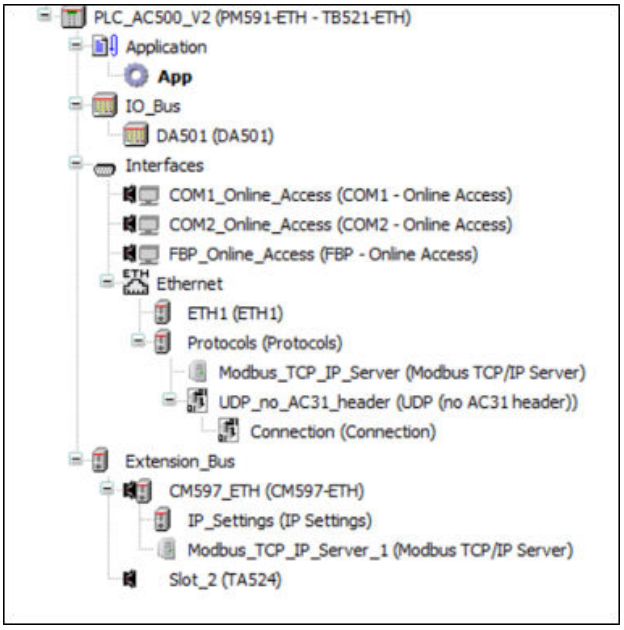
12. Add the task configuration *Chapter 1.5.8.1.5.2 “Task configuration recommendations for HA system” on page 2258*.
13. Activate the runtime license if it is a V3 PLC to enable HA system. Refer to *Chapter 1.6.6.2.2.2 “PLC runtime licensing” on page 3665*.
14. Compile and download to both PLCs (simplified in V3 via integrated download manager).
15. Create a boot project, restart the complete system and RUN.
16. Operation: Test use cases (e.g. by putting the primary PLC to STOP mode and observe the switchover). For different use cases and behavior refer to .
17. Runtime error and diagnosis function block can be used to monitor the system . For details refer to chapter Diagnosis *Chapter 1.5.8.1.8 “Diagnosis” on page 2269*.

Configuration with communication interface modules and redundancy

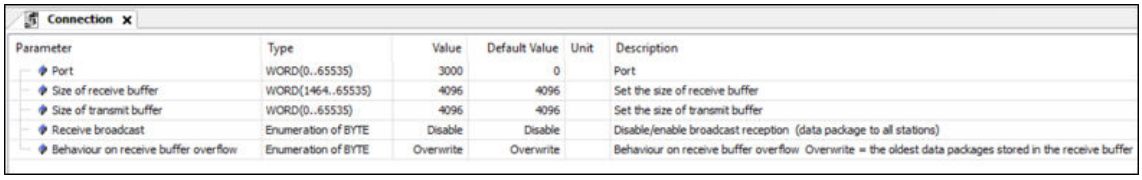
For medium or large HA systems the configuration with communication interface modules and redundancy is done by the following steps. For details see the example documentations:

1. Install the hardware *Chapter 1.5.8.1.2 “Hardware, requirements and options overview” on page 2237*.
2. Select the CPUs based on the requirements *Chapter 1.5.8.1.2.1 “CPU choice, system size and performance indications” on page 2239*.
3. Install Automation Builder including the latest libraries *Chapter 1.5.8.1.2 “Hardware, requirements and options overview” on page 2237*.
4. Install the Bulk Data Manager tool (BDM) *Chapter 1.5.8.1.4 “How to get and install the AC500 High Availability system package” on page 2252*.
5. Create a new project in Automation Builder for the chosen CPUs.

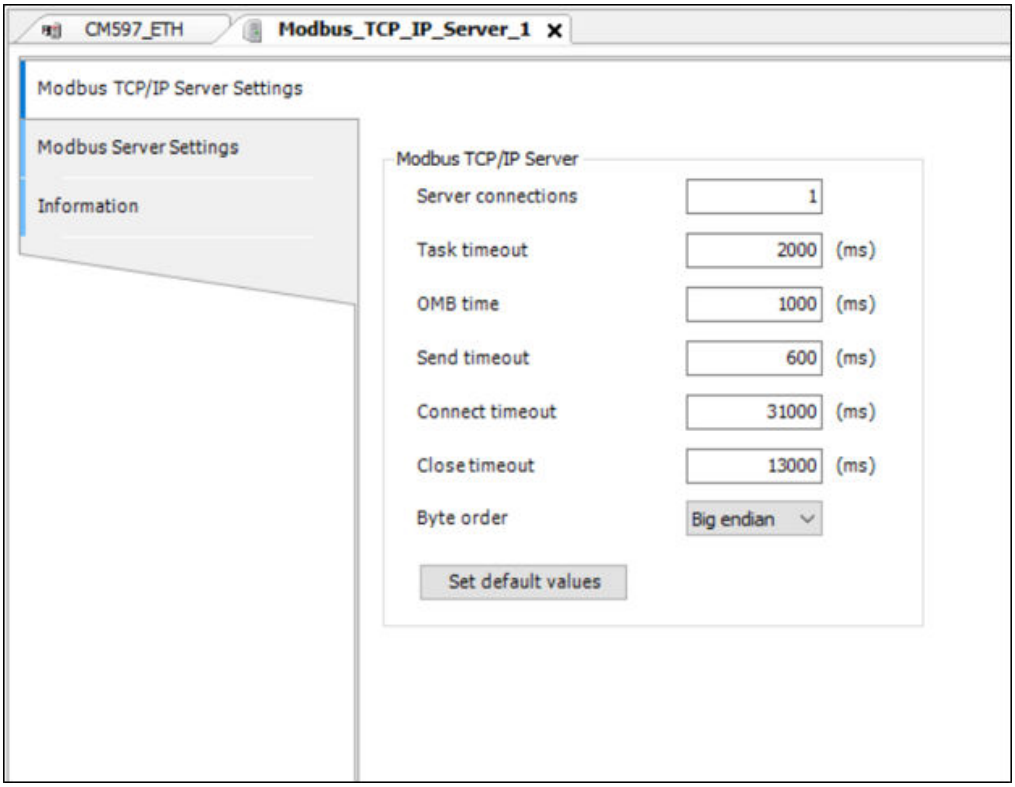
6. Configure the required Modbus and UDP configuration in the Automation Builderdevice tree of the CPU. UDP settings are only required in AC500 V2 PLCs.



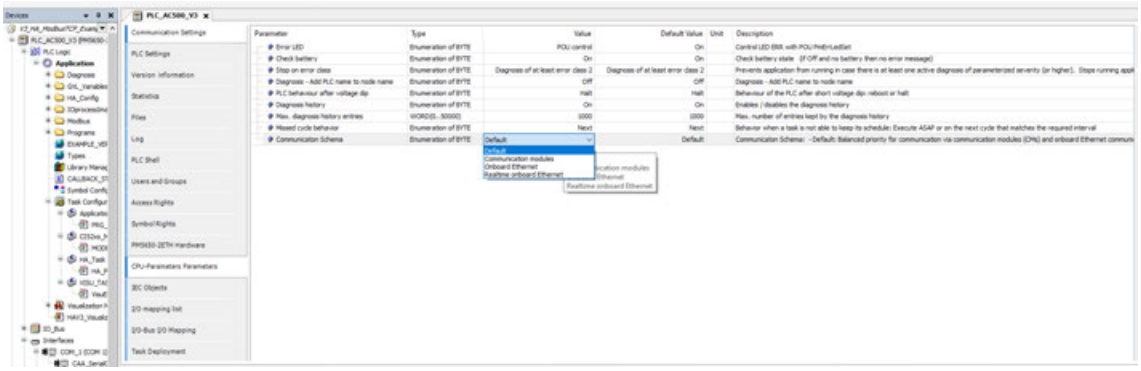
7. For UDP in AC500 V2 PLC, configure *"UDP_no_AC31_header"* and define the port number as '3000'.



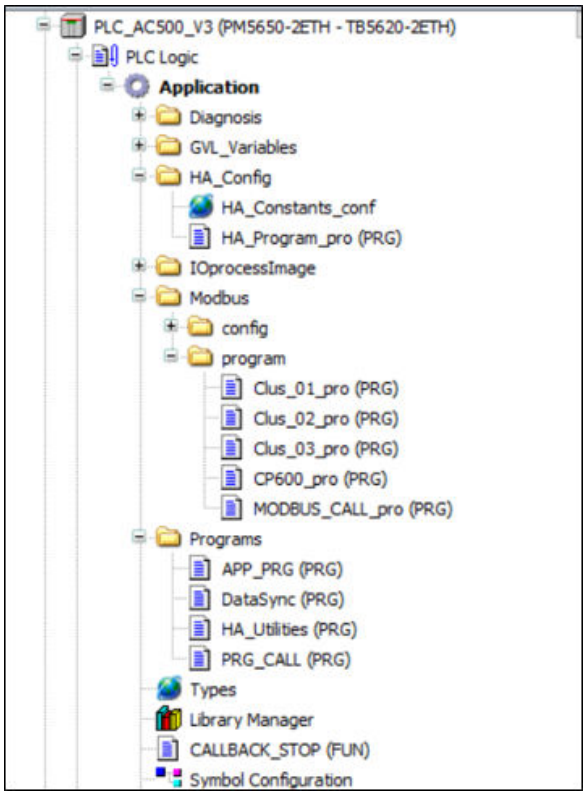
8. In AC500 V2 PLCs for each CM597-ETH communication module added the *"Send timeout"* value has to be changed to 600 ms for the Modbus TCP server.



9. Assign the IP addresses in ≥ 2 different Ethernet networks:
   - SCADA network: SCADA, connected PLC A and PLC B.
   - Field network: connected CI52x module(s).
10. Configure a network switch in the field network (if managed /redundant) based on network redundancy required ↳ *Chapter 1.5.8.1.5.3 “Field I/O network topologies” on page 2260*.
11. Run BDM tool to configure CI52x network.
12. Export the files. Refer for details in the document: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP\BulkDataManager\Documentation*.
13. Import the Bulk data export files to the Automation Builder project.
14. Add Modbus TCP configuration for the ETH ports.
15. For the system with V3 PLCs, set the Communication Schema to “Onboard Ethernet” *“CPU-Parameters Parameters”* for better performance.



16. Add Callback stop function HA_MOD_CALLBACK_STOP and call it in the system event *“stop”*.
17. Add optional HA utility function blocks or function block HA_MOD_DATASYNC.



18. Add the task configuration ↳ *Chapter 1.5.8.1.5.2 “Task configuration recommendations for HA system” on page 2258*.

19. Activate the runtime license if it is a V3 PLC to enable HA system. Refer to *Chapter 1.6.6.2.2 “PLC runtime licensing” on page 3665*.
20. Compile and download to both PLCs (simplified in V3 via integrated download manager).
21. Create a boot project, restart the complete system and RUN.
22. Operation: Test use cases (e.g. by putting the primary PLC to STOP mode and observe the switchover).
23. For different use cases and behavior refer to *Chapter 1.5.8.1.3.1 “Failures and use cases” on page 2243*.
24. Runtime error and diagnosis function block can be used to monitor the system. For details refer to *Chapter 1.5.8.1.8 “Diagnosis” on page 2269*.

1.5.8.1.7 HA-Modbus TCP Limits

HA-Modbus TCP is supported as of Automation Builder 2.0 or higher and the corresponding AC500 CPUs mentioned previously. AC500 V3 PLC is currently not supporting external ETH communication modules. Therefore, onboard ETH1, ETH2 (and eventually CAN) ports are to be used for communication.

3000 sync instances can be used: Either 3000 HA_MOD_DATA_SYNC function block instances alone or together 3000 instances of HA_MOD_DATA_SYNC inclouding + HA utility function block can be used. If more than 3000 instances are configured user can see the error at xHaModDataErr = True and wHaModDataErNo = 16#2022 in HA_GLOBAL_VARIABLES.

The maximum length of sync data at an instance of HA_MOD_DATA_SYNC function block would be 1412 bytes. The maximum size of sync data which can be synced between PLCA and PLCB in total can be max. 60 000 bytes.

The HA-Modbus TCP system takes care of the first fault only. This fault must be visualized by the programmer and overall system (e.g. HMI, SCADA) to the operator, to plan and repair as soon as possible as redundancy might be lost. If more than one error occurs, system may not react to second or following faults.

SCADA/ HMI has to be configured/programmed to:

- Only read data from the primary PLC.
- Parameters and control data should be always written to both PLCs or has to be synchronized via the function block.

This is given automatically when using OPC DA, where the CODESYS OPC Server does this switching for the connected clients according to the primary status. For CP600 HMI a script is available to switch likewise (connected via the internal AC500 protocol or Modbus). Zenon as a SCADA also uses the AC500 protocol to automatically switchover.

1.5.8.1.8 Diagnosis

This chapter explains the diagnosis information available to the user in the HA Modbus library and CI52x library. Diagnosis information is available at the outputs of HA control function block, HA Diagnosis function block and at the CI52x function block.

Depending on the use case defined in *Chapter 1.5.8.1.3 “Functionality” on page 2243* different diagnosis information can be accessed.

Primary CPU currently can read-out the diagnosis information (CI52x function block outputs) from communication interface module only once, hence secondary PLC will not be able to read the diagnosis information from the CI52x module.

So if any change happens in CI52x diagnosis it is not reflected in the secondary CPU.

This can lead to different diagnosis information of CI52x module in the primary and the secondary CPU. Hence it is recommended to customers that diagnosis information should be handled in the application (e.g. SCADA).

Diagnosis in HA-Modbus TCP library

In the HA Modbus library diagnosis information is available at the control block and diagnosis block.

Output System Configuration

This output at the HA control block gives the information of system configuration. Each bit of the word represents a different configuration.

Bit	Description
0	Sync is configured via CAN
1	Sync is configured via UDP
2	Lifecom2 is configured via CAN
3	Lifecom2 is configured via UDP
4	Lifecom2 is configured via Modbus TCP
5	Initialization for Ethernet configuration

Output System Configuration error

This output at the HA control block gives the details of error in the configuration. Each bit of the word represents different configuration errors. It is valid only when Error = TRUE.

Bit	Description
0	Communication interface module is not configured properly
1	1< SyncSlot >3. Invalid value at input sync slot
2	1<SecSlot>3. Invalid value at input second slot
3	Value at IpAdrCpuASync is invalid
4	Value at IpAdrCpuBSync is invalid
5	Value at IpAdrCpuALifecom2 is invalid
6	Value at IpAdrCpuBLifecom2 is invalid
7	IpAdrCpuASync = IpAdrCpuBSync or IpAdrCpuALifecom2 = IpAdrCpuBLifecom2 The IP addresses assigned at sync or lifecom2 inputs are wrong

Output Runtime error

This output at the HA control block gives the details of the error during run time of the system. Each bit of the word represents different runtime errors. It will not set Error = FALSE.

Bit	Description
0	Communication interface modules are lost
1	Other CPU is not active
2	Lifecom1 is lost (part of sync)
3	Lifecom2 is lost. This error will not be TRUE if the PLC is in STOP status. This is because Modbus is still responding even when PLC is in STOP
4	Synchronization is lost
5	Error in synchronization
6	Ethernet status error
7	Other PLC lost communication to CI52x modules
8	CAN_HEADER function block has error
9	CAN_DATA function block has error
10	fbGetOwnIP function block has error

Diagnosis function block

Outputs at the HA Diagnosis function block, HaModDiag (V3) / HA_MOD_DIAG (V2) provides the following diagnosis information of the HA system.

Output	Description
CpuAPrimary / CPUA_PRIMARY	TRUE indicates CPU A is primary
CpuBPrimary / CPUB_PRIMARY	TRUE indicates CPU B is primary
CpuARun / CPUA_RUN	TRUE means CPU A is in RUN mode
CpuBRun / CPUB_RUN	TRUE means CPU B is in RUN mode
CpuACI52xBusActive / CPUA_CI52x_BUS_ACTIVE	Modbus TCP CI52x bus active on CPU A
CpuBCI52xBusActive / CPUB_CI52x_BUS_ACTIVE	Modbus TCP CI52x bus active on CPU B
CpuACi52xCfg / CPUA_CI52x_CFG	Total number of CI52x configured on CPU A
CpuBCi52xCfg / CPUB_CI52x_CFG	Total number of CI52x configured on CPU B
CpuACi52xAct / CPUA_CI52x_ACT	Total number of CI52x active on CPU A line
CpuBCi52xAct / CPUB_CI52x_ACT	Total number of CI2x active on CPU B line
SyncInstances / SYNC_INSTANCES	Number of data sync and utility blocks initialized in the system
SyncDataCheckSum / SYNC_DATA_SUM	Checksum of all address pointer blocks in bytes, indicates total number of bytes getting synchronized.

Output	Description
StHACpuStatus / stHA_CPU_STATUS	HA own CPU status. It will show the status details of logged in CPU for the following parameters: ● HA1: CPU A is primary ● HA2: CPU B is primary ● bit_CI52x_BUS_active: CI52x bus with one or more communication interface modules active ● bit_CI52x_BUS_err: CI52x bus one or more communication interface modules powered off / connection lost ● RUN: Run status of CI52x ● cnt: Count of data sync communication, indicates data sync between CPUs is okay.
StHAotherCpuStatus / stHA_OTHER_CPU_STATUS	HA other CPU status. It will show the status details of other CPU for the following parameters: ● HA1: CPU A is primary ● HA2: CPU B is primary ● bit_CI52x_BUS_active: CI52x bus with one or more communication interface modules active ● bit_CI52x_BUS_err: CI52x bus one or more communication interface modules powered off / connection lost ● RUN: Run status of CI52x ● byETH_ACT_CI52x_Count: CI52x alive identification count.

Other diagnosis variables

Apart from the errors / diagnosis information available in the control and diagnosis block, few other variables can be monitored too.

Variable	Value	Description
wHA_ER_NO_SYNC_LINK	16#7487	No sync link between the PLCs
HA_MOD_INVALID_LENGTH	16#2017	Invalid length at the input of the data sync block
HA_MOD_ERNO_TBL_OVERFLOW	16#2022	HA data reference table is full
xHaModDataErr	TRUE	IF TRUE – HA data sync is in error state
wHaModDataErNo		HA data sync error code
xHaModErr	TRUE	HA system is in error state
dwHaModServerAlive		Life counter incremented by OPC DA server

Diagnosis in CI52x library

In addition to the diagnosis information in the HA Modbus library, additional diagnosis information for each communication interface module can be obtained from the CI52x library.

System Configuration error

This output at the CI52x function block gives the details of the configuration error in the CI52x module. Each bit of the byte represents different configuration errors:

Bit	Description
0	Reserved
1	Wrong ETH port is configured at input Config ETH
2	Wrong IP address is configured for communication interface module

Runtime error

RuntimeError (v3) / RUNTIME_ERROR (v2) of the function block CiModCi52x (v3) / CI_MOD_CI52x (v2). Runtime error is a combination of error bits that are described in the following:

Runtime Error	Description
Bit 0	Indicates communication error i.e., when CPU is not able to get any response from CI52x module. This error will get reset when communication is reestablished.
Bit 1	Indicates parameter state is not equal to 2 (PARA_STATE_PARA_DONE). If not true, then system gives I/O bus error. System resets this error when parameter state is equal to 2.
Bit 2	Indicates the cluster error [1]) in the system, if there is an error in the diagnosis buffer. ACK input is needed to reset this error.
Bit 3	Indicates the hardware configuration error, mismatch between configuration and actual hardware detected. System automatically resets this error when the hardware matches.
S-ERR (LED on communication interface module)	Indicates that there is some issue with channel configuration in the cluster [1]). It is not linked with Runtime Error. User can read DiagBuffer (v3) / DIAG_BUFFER (v2) from CiModDiag (v3)/ CI_MOD_DIAG (v2) function block to get more information. This error does not get reset using ACK. It will only reset when all channel errors are removed.
[1]) "Cluster" means a combination of one communication interface module and several I/O modules attached to it.	

Runtime error in different scenarios:

Error	Run-time error	PLC A: Primary				PLC B: Secondary			
		Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error	Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error
Wrong IP address configured	16#1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
Wrong slot address configured [1])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Communi-cation cable dis-connected	16#2	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
Wrong I/O module plugged in the CI module	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
Wrong hotswap I/O module plugged at the start	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
Wrong hotswap I/O module swapped online	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Configured I/O module not con-nected at start [2])	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
Configured hotswap I/O module not con-nected at start [2])	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
I/O module powered off in CI module [2])	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Hotswap I/O module powered off in CI module [2])	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Remove hotswap I/O module when online [2])	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE

Error	Run-time error	PLC A: Primary				PLC B: Secondary			
		Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error	Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error
CI module is powered off	16#2	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
Mismatch in Channel configura-tion and wiring [3])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regular I/O module mounted on hotswap terminal unit [4])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
During an error stage if HA system changeover is initiated [5])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

[1]) Slot input in the block can be ignored. Similar to ETH input of the ModMast blocks.

[2]) Error generated only in the primary PLC, to reset ACK input to be used.

[3]) No runtime error in function block. Module generates S-Err and ZP Blinks.

[4]) No runtime error in function block. Module generates S-Err.

[5]) Runtimer Error bit2 gets reset when the PLC is switched over and error won’t be available in any of the PLC regardless of its Primary status.

Communication interface diagnosis

Table 381: Function block CiModDiag (V3) and CI_MOD_DIAG (V2)

Output	Description
DevState / DEV_STATE	CI521 or CI522 device current status is displayed. ● STATE_PREOP: Device is booting ● STATE_OPERATION: Device is operational, no bus supervision is active ● STATE_ERROR: Device detected a bus error, bus supervision is active ● STATE_IP_ERROR: Device has an IP address error ● STATE_CYLIC_OPERATION: Device is operational, bus supervision is active ● STATE_NA: Not available
ParaState / PARA_STATE	CI521 or CI522 device parameter status. ● PARA_STATE_NO_PARA: Device has no parameters ● PARA_STATE_PARA_ACTIVE: Parameterization process is running ● PARA_STATE_PARA_DONE: Device used valid parameters and parameterization is done ● PARA_STATE_ERROR: Device has invalid parameters ● PARA_STATE_NA: Not available
DeviceInfo / DEVICE_INFO	CI521 or CI522 type and extended module types. This will give the details of the module configured in the communication interface module including the I/O modules. If module is with suffix F, then fast counter is enabled for that module.
DiagBuffer / DIAG_BUFFER	CI521 or CI522 module diagnosis buffer. Refer to *Chapter 1.6.5.3.1.2.3.2 “Diagnosis data” on page 3611*.
ErClass / ERR_CLASS	Communication interface error class. Refer to *Chapter 1.7.3 “Diagnosis messages” on page 4062*
ErNo / ERR_NO	Communication interface error number. Refer to *Chapter 1.7.3 “Diagnosis messages” on page 4062*
ModMastErr / MODMAST_ERR	Latest 22 Modbus TCP error message status of the ModMastTcp (V2) / COM_MOD_MAST (V2) function block.
ModMastErNo / MODMAST_ERR_NO	Latest 22 Modbus TCP error numbers. Refer to the error details in Modbus library *Chapter 1.5.8.1.8.1 “Diagnosis in HA-Modbus TCP library” on page 2270*. V3: Refer to the ERROR_ID enumeration in the Modbus TCP library (in the Library Manager)

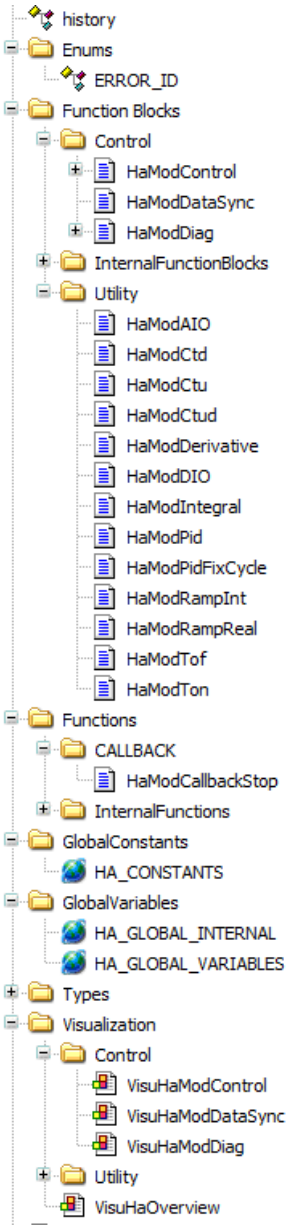
1.5.8.1.9 Library overview

Documentation

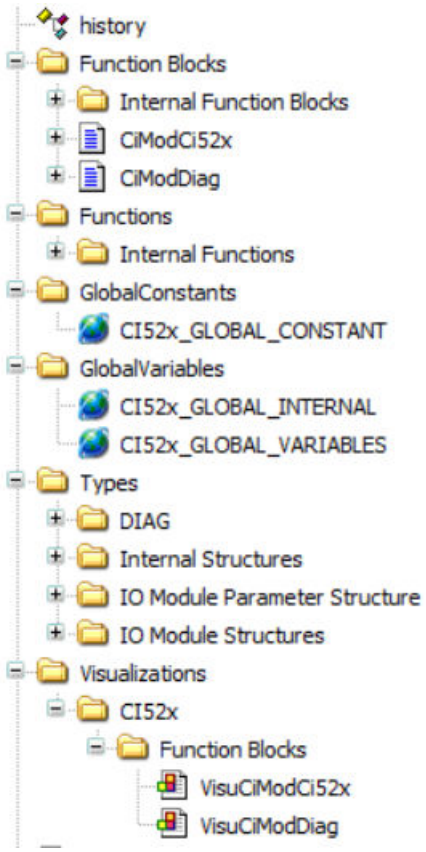
CODESYS V3 libraries are described in the Library Manager as an integrated documentation. Refer to the documentation section within the Library Manager.

The following function blocks are contained in the libraries:

HA-Modbus TCP library



HA_CI52x library

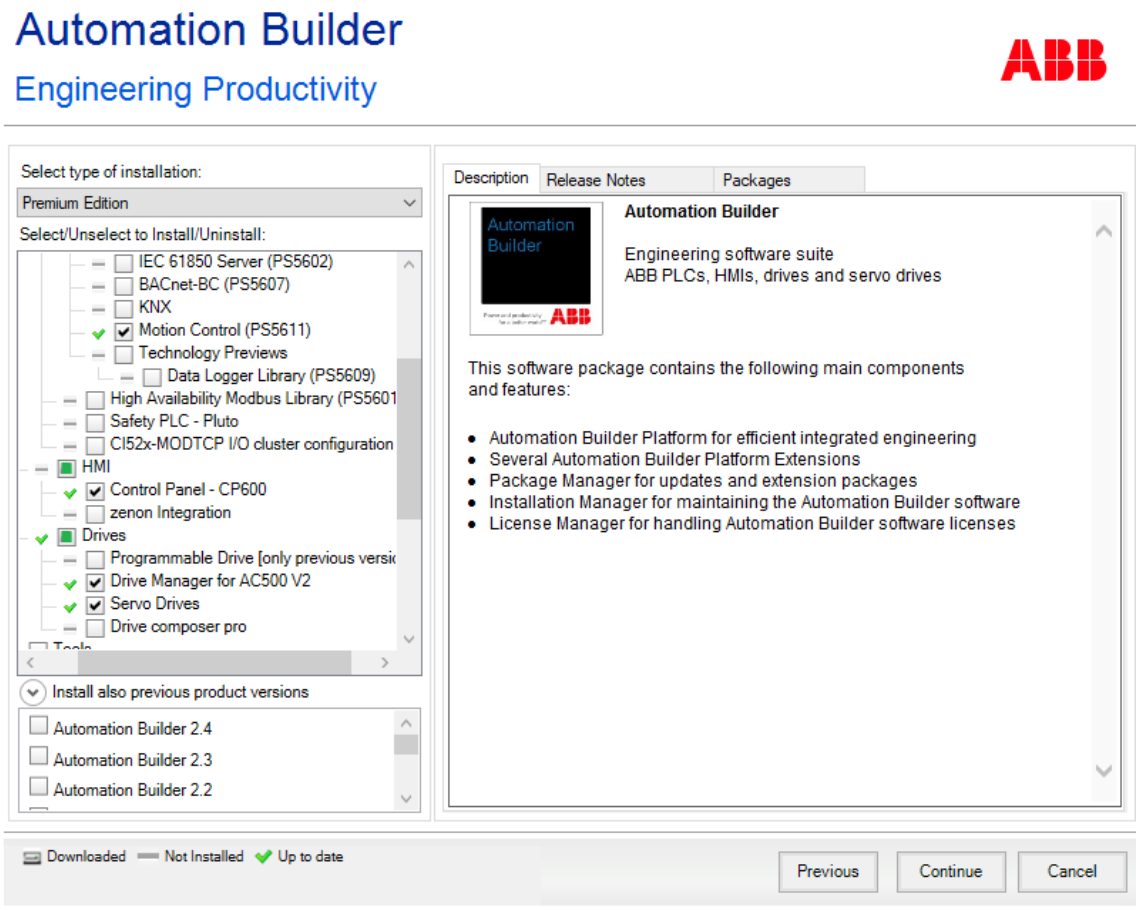


1.5.9 Motion Solution Wizard

Preconditions for the use of the *"Motion Solution Wizard"*

To be able to use the Motion Solution Wizard and the detailed information pdf, the following software packages must be installed:

- *"Motion Control (PS5611)"*
- *"Servo Drives"*



1. Select *“Tools”* and *“Installation Manager”* to open it.
2. Select *“Modify”*.
3. Mark the mentioned software packages.
4. Click the *“Continous”* button.
   ⇨ The various packages are downloaded and installed.

In the installation process, some operations must be confirmed.

Further detailed information about the “Motion Solution Wizard” and the “CAM editor” which are both basing on the “Motion control library” according PLCopen can be found in “AC500_V3_Motion Control_Wizard&Cam_Editor_Quick_Start_Guide_3ADR010899.pdf”:

C:\Users\Public\Documents\AutomationBuilder\Examples\PS5611-Motion

Or via the Automation Builder: “Help ➔ Project examples ➔ PS5611-Motion”

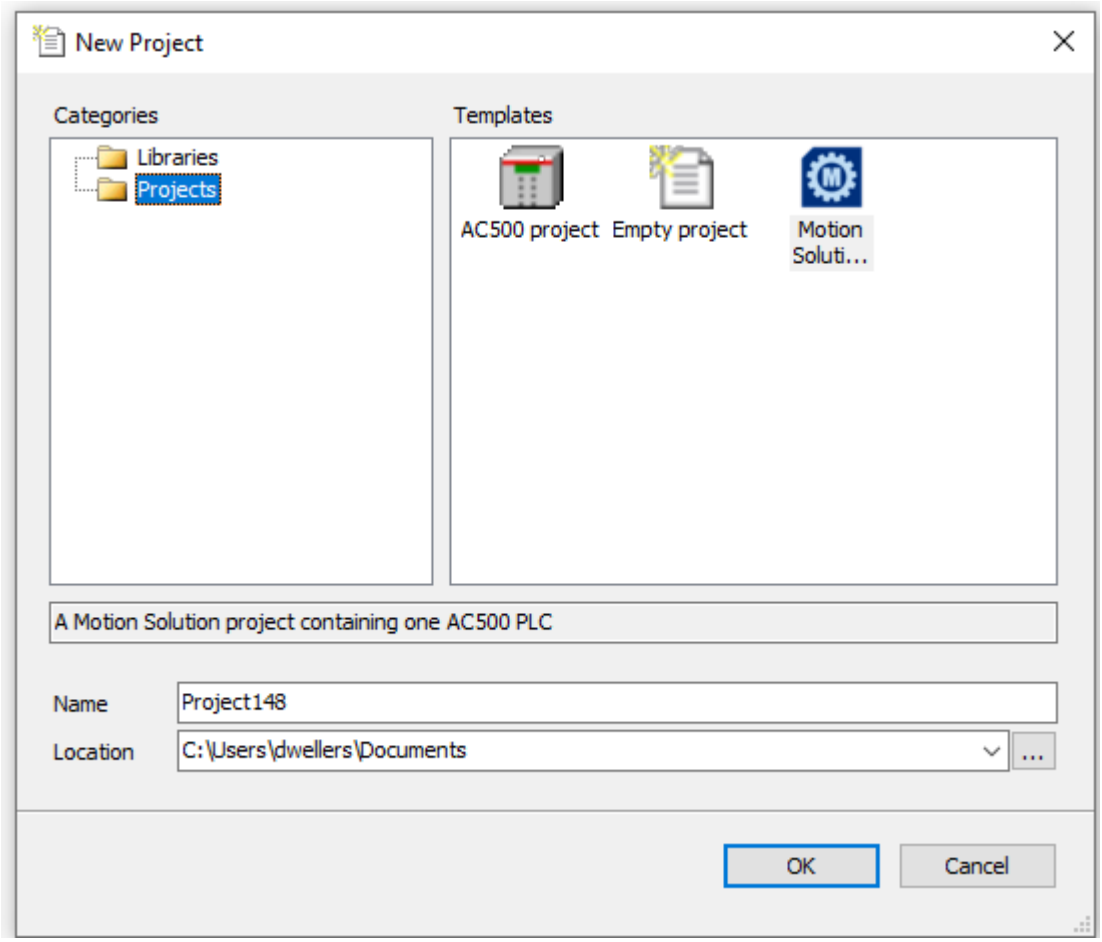
↬ Chapter 1.4.1.8.23.1 “Basic Motion” on page 317

Currently the “Motion Solution Wizard” supported only the ABB EtherCAT servo drives (E180/E190).

More information about additional software packages is available here: ↳ *Chapter 1.3 “Automation Builder installation manager” on page 169*

1.5.9.1 Create new project

Select *“File”* ➔ *“New Project”*.

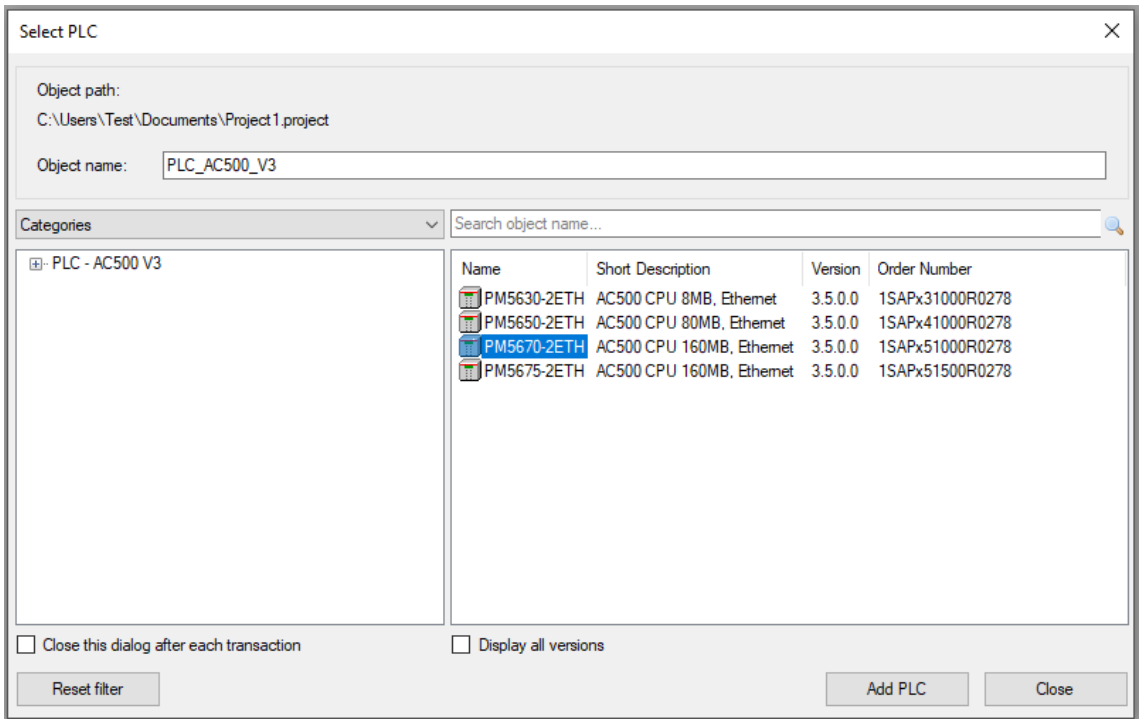


After successful download and installation of the missing software packages, the *“Motion Solution Project”* icon is showed in the *“Templates”*.

1. Select the *“Motion Solution Project”* icon.
2. Enter a project name.
3. Specify the storage location for the new project.
4. Click the *[OK]* button.
   ⇨ A new project is created and can be configured.

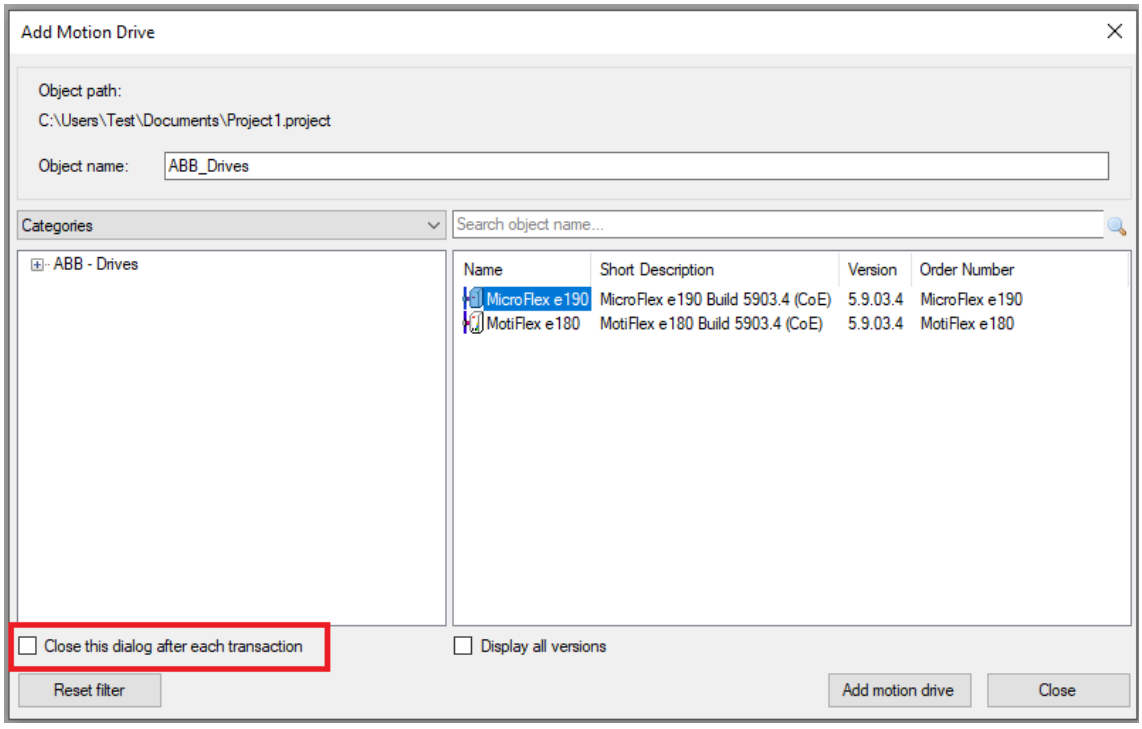
1.5.9.2 Select PLC

The Motion Solution Wizard can only be executed in an EtherCAT environment.



1. Select the desired PLC device.
2. Click the *[Add PLC]* button.
   ⇨ Along with the PLC added, Motion Solution Wizard and CM579-EtherCAT module also get added (Slot 1).

1.5.9.3 Select servo drive (motion axis)



After creating the hardware tree, Automation Builder will now pop up *"Add Motion Drive"* window. Here it shows all the installed EtherCAT supported ABB Servo drives.

Each servo drive added under the EtherCAT master will be counted as a motion axis in Automation Builder.

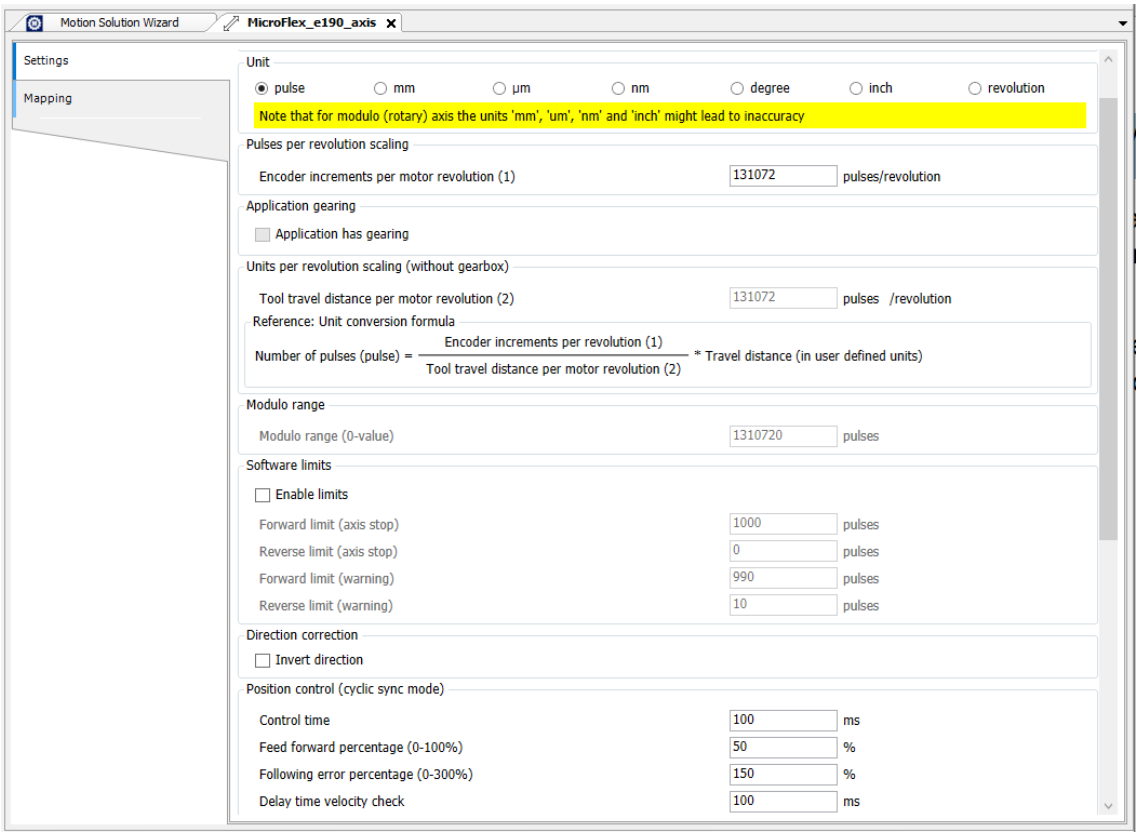
1. Delete the check mark of *“Close the dialog after each transaction”* (Red rectangle in the figure above).
   ⇨ Now several axes can be selected one after the other without closing the window.
2. Click on the motion drive you want to install.
3. Click *“Add motion drive”*.
   ⇨ In the *“Devices”* tree, the selected axis is displayed.
4. For more axes repeat the procedure.
   ⇨ The selected axes are displayed in the *“Devices”* tree.
5. Click the *[Close]* button if the required axes have been selected.

1.5.9.4 Configure servo drive (motion axis)

Users need to configure each axis separately as per the application requirement by opening the motion axis object which is added under the servo drive.

1. Double-click on the axis you want to configure.
   ⇨ The *“Settings”* and the *“Mapping”* tab are shown.
2. Now select the desired settings.

Depending on the selected axis type, the view and selection of setting options may change.



Settings

All settings related to the application and axis specific will be done here and needs to be carefully updated for each axis. Based on the inputs provided here, wizard will compile and generated the code.

The following *"Settings"* are available:

Parameter		Value	Description
Axis type			
	Modulo (rotary)		By selecting the Modulo (rotary) the axis will be configured as a roll-over axis and the desired modulo range can be configured later.
	Finite (rotary)	Default	The axis will be configured as a roll-over axis where in modulo range it is non editable by the user and calculated based on the *"Unit"* selection, *"Inc_Per_R, U_Per_Rev_Nominator and U_Per_Rev_Denominator setting"*.
	Linear (rotary screw)		Rotary motor with linear movements (linear axis).
	Linear (linear motor)		Configure if the axis is a linear motor.
	Virtual axis		This option is an additional check which user can do along with the axis type selection to make the configured axis type (physically configured) as virtual axis.
Units			
	Supported units are: Pulse, mm, µm, nm, degree inch and revolution. For *"Modulo (rotary)"* axis the units mm, µm, nm, and inch might lead to inaccuracy.		
Feedback device scaling			
	Forcer distance between an A and a B pulse(1)	40000 nm	Only available with an *"Axis type" "Linear (linear motor)"*.
Pulse per revolution scaling			
	Encoder increments per motor revolution (1)	131072 pulses / revolution	User can update this parameter with the actual encoder increments per motor revolution. Not available with an *"Axis type" "Linear"*.
Application gearing			
	Application has gearing	☐	Based on the actual application requirement, here user can check / uncheck the *"Application has gearing"* check box. Here user can also update the required tool travel distance per motor revolution. Not available with an *"Axis type" "Linear"*.
Units per revolution scaling (without gearbox)			
	Tool travel distance per motor revolution (2)	131072 pulses / revolution	When the user unchecks the check box, user can update the *"Tool travel distance per motor revolution"* as per the application requirement. Not available with an *"Axis type" "Linear"*.
	Number of pulses (pulse) =	$\frac{(1)}{(2)}$	* Travel distance (in user defined units)

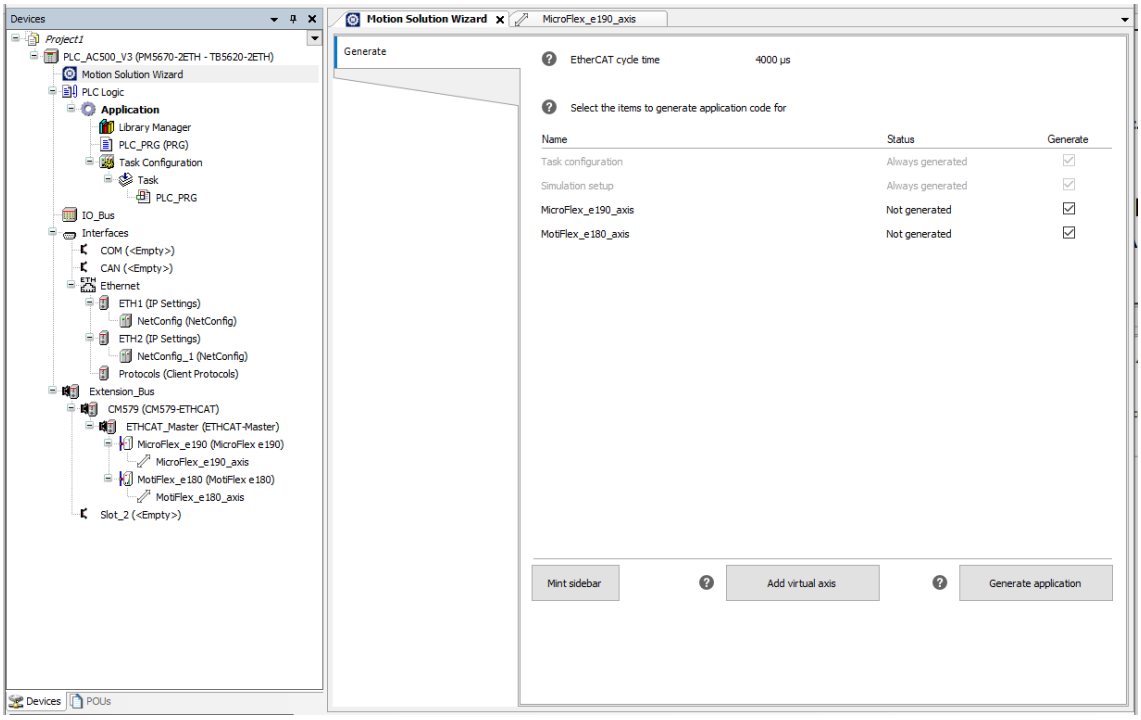
Parameter		Value	Description
Units per revolution scaling (with gearbox)			
	Tool travel distance per Gearbox output side revolution	1 mm /revolution	When the user checks the check box, user will be prompted to provide the gear box details additionally and during the generate application, the wizard will update the same accordingly. Not available with an *“Axis type” “Linear”*.
	Gearbox output turns: Tooling side (Numerator of reducation ratio) (4)	1	
	Gearbox input turns: Motor side (Denuminator of reducation ratio) (5)	1	
	Number of pulses (pulse) =	$\frac{(1)*(4)}{(3)*(5)}$	* Travel distance (in user defined units)
Modulo range			
	Modulo range (0-value)	131072 pulses	User can provide the modulo range here. Active only when the user selects the axis type as any of the “rotary” axis.
Software limits			
	Enable limits	☐	User can configure some of the common *“Software limits”* from the wizard itself. By default, *“Software limits”* in wizard are not enabled and user need to enable the same by enabling the check box.
	Forward limit (axis stop)	1000 pulses	
	Reverse limit (axis stop)	0 pulses	
	Forward limit (warning)	990 pulses	
	Reverse limit (warning)	10 pulses	
Direction correction			
	Invert direction	☐	In some of the application it is neccessary to change the direction for actual and reference positions. By default, the check box will be unchecked, and the direction will be normal. By selecting the check box *“Invert direction”* both actual and reference position will be inverted.
Position control (cyclic sync mode)			
	Control time	100 ms	User can configure the parameters related to position control and supervision. ↳ *Chapter 1.5.10.4.3 “Axis parameters” on page 2353*
	Feed forward percentage (0-100%)	50 %	
	Following error percentage (0-300%)	150 %	
	Delay time velocity check	100 %	
	Position lag supervision	Activated (default)	
		Deactivated	
Dynamic limits			
	Maximum application velocity	36000 pulse /sec	The units defined for each parameter and update the same accordingly since each parameter here is having a separate unit. Some parameters are depending on the drive settings and needs to be set correctly to get the desired result.
	Maximum speed reference value	13107200 Unit dependant on drive settings	
	Maximum speed (user defined)	6000 rpm	

Parameter		Value	Description
	Maximum acceleration	10000 pulses / sec²	It is recommended to keep the same maximum speed at the drive and the PLC parameter as the same and if user can set the maximum application velocity to a desired value to limit the maximum application speed.
	Maximum deceleration	10000 pulses / sec²	
	Maximum jerk	2000 pulses / sec²	
Drive based limits			
	Maximum positiv torque	300 %	Define the torque limits in wizard and the same will be written to the SDO startup parameter if the user select *"Torque limits"* in *"Mapping"* page. There parameters are currently not used in the program by default.
	Maximum negative torque	-300 %	
Results (calculated)			
	Position resolution	1	Based on the inputs provided, wizard will calculate the results and can be viewed immediately at the end of the configuration page.
	Maximum possible velocity	1.31072E+07 pulses /sec	
	Maximum allowed following error	1966080 pulses	

Mapping tab

Parameter		Value	Description
Control type			
	Cyclic synchronous position mode (CSP)	Cyclic synchronous velocity mode (CSV)	By default, wizard is selected for *"Cyclic synchronous position mode (CSP)"*. User can change the same based on his application requirement. CSVL is an ABB specific mode to achieve load control / profiling. By using this mode user can use the *"MotionControlLoad"* library. *Chapter 1.5.10.4.7 "PLC-based motion control — Load control / fluid power extensions" on page 2367*
		Cyclic synchronous velocity mode for load control (CSVL)	
Additional PDO mapping			
	Touch probe 1 pos	☐	Most of the applications needs additional PDO mapping and the wizard helps the user to add most used PDO mapping just by selecting the same here. User can add other PDO mapping which are not listed here manually by enabling the expert settings from the axis configuration page.
	Touch probe 1 neg	☐	
	Touch probe 2 pos	☐	
	Touch probe 2 neg	☐	
	Master encoder	☐	
	Following error	☐	
	Digital input states	☐	
	Digital output force	☐	
SDO startup parameter mapping			
	Give EtherCAT control	☑	By default, two of the SDO startup parameters are always checked and it is recommended not to change them.
	Operating mode	☑	
	Torque limits	☐	

1.5.9.5 Open Motion Solution Wizard editor page and generate application



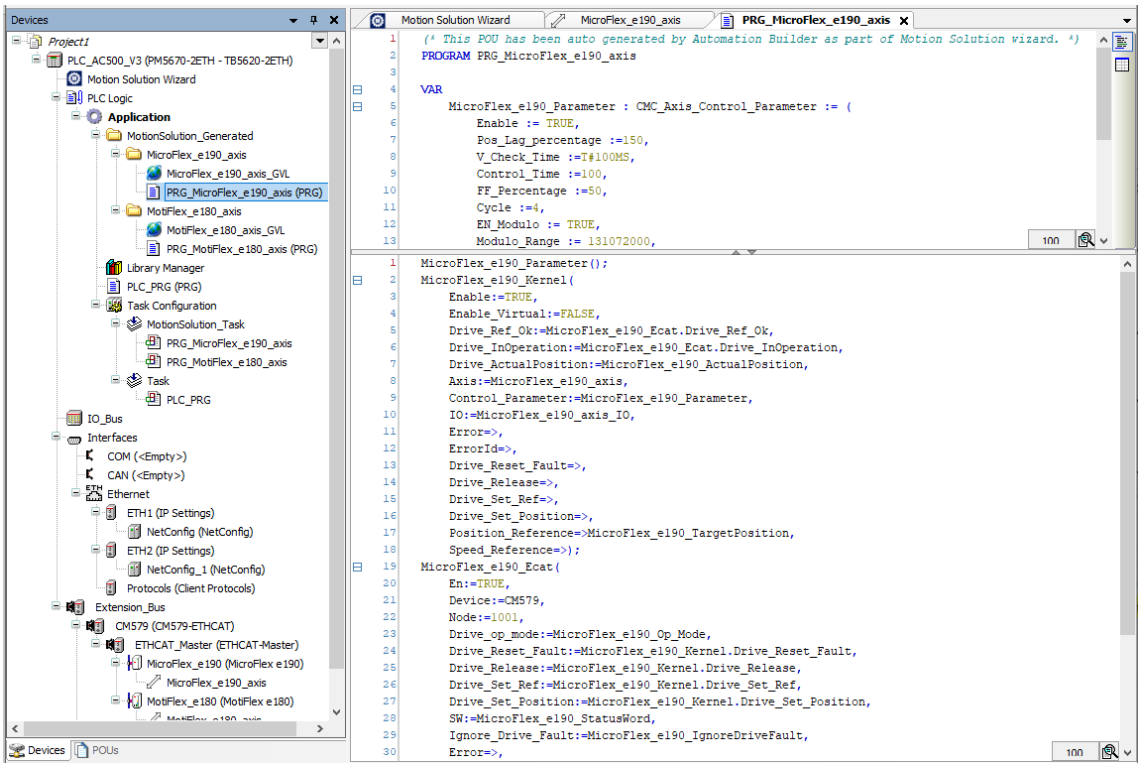
Once all the settings have been made, double-click to switch to the *“Motion Solution Wizard”* icon in the *“Devices”* tree.

All existing axes are listed here with their status. In the screenshot above *“Not generated”*.

1. Set a check mark for the axes that are to be generated under the column *“Generate”*.
2. Click the button *“Generate application”*.
   ⇨ The application code/s will now be generated.
3. Confirm the *“Generation successful”* window by cklicking the *[OK]* button.

1.5.9.6 Check generated application

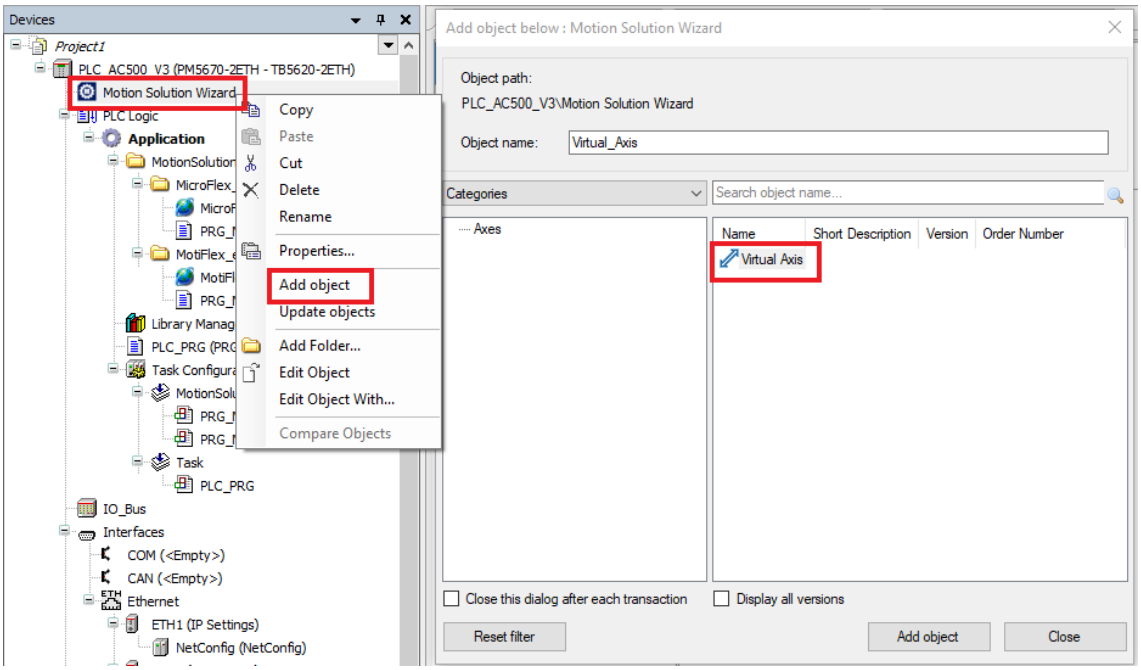
A new folder has been created in the *“Application”* - *“MotionSolution_Generated”*. In this folder are the folders of the generated axis with the global variables and function block calls. The added axes are mapped in the *“Devices”* tree under *“Extension_Bus”*.



These application codes can now be copied as required and integrated into your own PLC programming.

1.5.9.7 Optional: Add and configure virtual axis for simulation without real axis

Some of the applications needs virtual axis to be configured to fulfill the application requirement.



1. Right click on the *“Motion Solution Wizard”*.

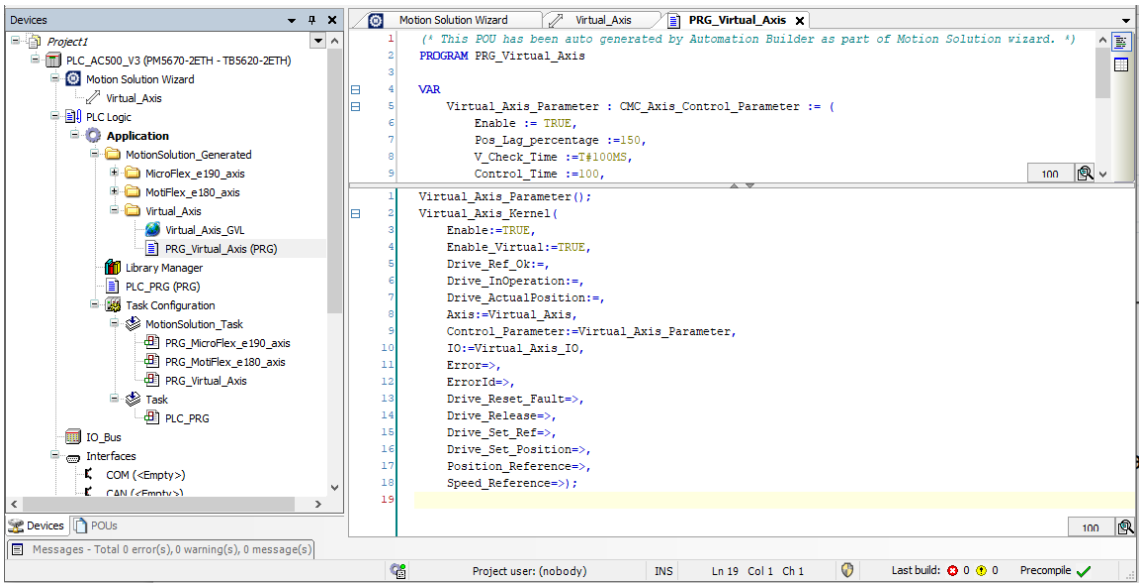
   ⇨ A new window opens.

2. Click *“Add object”*.

   ⇨ A new window opens.

3. Mark the *“Virtual Axis”* object.
4. Click the *[Add object]* button and the *[Close]* button.
   ⇨ A virtual axis was created.

After adding the virtual axis, user can find the same under *“Motion Solution Wizard”* object in Automation Builder. User can double-click on the added virtual axis object to get the settings page and configure it as per the requirement. Settings here is similar to the motion axis.



1.5.10 Motion control library

Safety instructions

- All pertinent state, regional, and local safety regulations must be observed when installing and using this product. When functions or devices are used for applications with technical safety requirements, the relevant instructions must be followed.
- Read the complete safety instructions of the user's manuals for the drives you are using, before installation and commissioning.
- Read all safety instructions of the AC500 PLC. See System description AC500 or chapter *Chapter 1.6.2.4 “Regulations” on page 2406* in the online help.
- Read the Important user Information. See chapter *Chapter 1.6.2.1 “Safety instructions” on page 2395* in the online help.

1.5.10.1 Preconditions for the use of the libraries

The user has to read the following instructions and documents before using the libraries:

The library package has been released for the software and firmware versions listed in the readme file of Automation Builder only (see “Help ➔ Automation Builder Release Notes”) . In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the HA library with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e. g. of example programs.

The first version of Motion Control Library Package PS5611-Motion has been released with Automation Builder 2.4.0.There after the package is updated with several changes. For details on all changes please refer PS5611-Motion release note area from Automation Builder release notes.

The Motion control package contains follows libraries:

Library	Automation Builder	PLC firmware
ABB_MotionControl_AC500	AB 2.4.0 or higher	AC500 V3 firmware version 3.3.1 or higher AC500-eCo V3 firmware version 3.4.0 or higher
ABB_Ecat_CiA402_AC500		
ABB_MathFunctions_AC500		
ABB_MotionContro-lEco_AC500 (kernel blocks for Eco V3 PLCs)		
ABB_MotionControl-Load_AC500	AB 2.5.0 or higher	AC500 V3 firmware version 3.5.0 or higher

The PS5611-Motion libraries have been tested with the following product / firmware / software versions:

- AC500 V3 PLC firmware 3.3.1 or higher
- AC500-eCo V3 (PTO & PWM) firmware 3.4.0 or higher
- CM579-ETHCAT EtherCAT Communication Module firmware 4.4.3.21 or higher
- ABB e190 Drive
- CD522 module

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product / software / firmware versions. The error-free operation of the PS5611 - Motion with other devices / software / firmware versions should be possible but can not be guaranteed and may need adaptations e. g. of example programs.

CAUTION!
Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

Limits on number of synchronized axis

There are limits on the minimum EtherCAT cycle time, user can configure in each PLC type.

Table 382: Details on the limits on the minimum EtherCAT cycle time

PLC type	PM5630	PM5650	PM5670
Min. EtherCAT master cycle time	2 ms	1 ms	0,5 ms

Other than the above limits, there is also limits on configuring the number of synchronized axis in each PLC type. This limits is based on the EtherCAT master cycle time configured under EtherCAT master.

Table 383: Details on the limits for each PLC type

PLC type	PM5630	PM5650	PM5670
Number of synchronized axis in 1 ms	-	8	16
Number of synchronized axis in 2 ms	4	16	32
Number of synchronized axis in 4 ms	8	32	64

"Number of axis" is counted in Automation Builder is based on the number of Kernel function block instance declared in the IEC application. In this way, it is made sure all real and virtual axis are counted.

User can increase the EtherCAT cycle time to accommodate more "Number of axis" in the same PLC type.

User can use the *[Statistics]* tab from Automation Builder to see how many axis are supported for the particular PLC type and for the EtherCAT master cycle time configured. Once the axis is configured user need to update the *[Statistics]* tab by *"Generate Code"* to get the updated information.

Automation Builder allows an additional axis than what is mentioned in the above table to support one virtual axis additionally.

Please remove any Kernel function block instance which is declared but not used in the application to get the correct number of axis calculated by Automation Builder under the [Statistics] tab.

1.5.10.2 Overview

The PS5611-Motion is a Motion Control library for AC500 V3 CPUs, to create Motion Control applications based on function blocks according to the standard of PLCopen Motion Control. These function blocks can be used for PLC-based Motion Control and cover a wide range of possible Motion Control functionalities. Starting from single axis movements to master-follower axes to perform electronic gearing and CAM functions.

This documentation contains the following chapters:

- Overview
  In the subsequent chapters general information are provided for a better understanding of Motion Control with AC500 PLC and PS5611-Motion. There is also a tabular overview of the available PLCopen function blocks and their compatibility with PLC-based Motion Control and the provided drive-based Motion Control axis implementations.
- PLCopen
  The principle of the PLCopen Motion Control standard is explained as well as how PLCopen function blocks can be used to create PLC Motion Control application programs.
- PLC-based Motion Control
  This chapter explains how PLC-based Motion Control with AC500 can be realized and how it can be used in combination with the available PLCopen function blocks.
- PLC-based Motion Control Fluid Power Extension or Load Control
  This chapter explains how the PLCopen part 6 Fluid Power - extension also called "Load Control" can be used to practically realize also a form of Torque control (or -profiling) and how it can be used in combination with the available PLCopen function blocks and switch between Torque/Load control and position control.

The detailed functionality of each function block is defined in the integrated documentation of the library.

1.5.10.2.1 PLC-based motion control

With PS5611-Motion the application program and the profile generator are realized in the PLC.The implementation of the profile generator is based on a set of function blocks which are named Central Motion Control (CMC).

The profile generator of many possible axes is centrally placed inside the AC500 PLC. Therefore multiaxis motion functionalities become easily available and can be accessed by PLCopen function blocks. As a result, Motion Control functionalities are almost drive independent.

The detailed functionality of each function block is defined in the integrated documentation of the library.

Available motion control functionalities:

- Simple axis Movements
- Electronic Gearin
- Electronic CAMs
- Position Profiles
- Velocity Profiles
- Acceleration Profiles
- Load control (Torque profilling)

Then the output is a position reference signal which the drive will follow. A new position reference value will be calculated with every cycle of the PLC and has to be transferred to the drive, which demands real time capabilities to the PLC and to the communication channel. A real time fieldbus like EtherCAT is needed. The feedback of the actual position can be used for supervision purposes during operation and is needed to adjust the value of the position reference before the drive will be enabled.

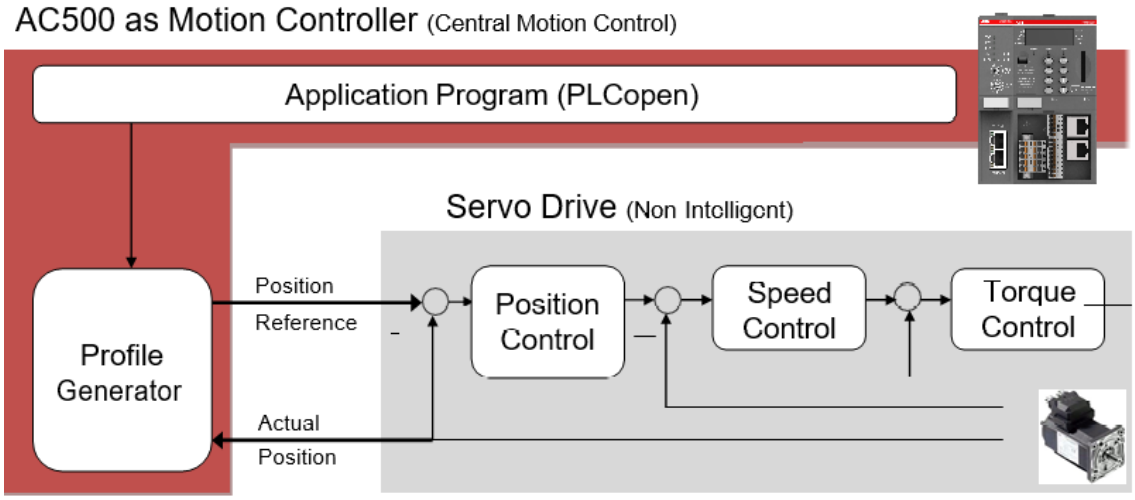


Fig. 49: System structure of PLC-based Motion Control with AC500 PLC and PS5611-Motion

With PLC-based Motion Control it is also possible to include the position control loop to the AC500 PLC. In this case a speed reference signal will be transferred to the drive, which makes it possible to perform the full range of motion functionalities with standard drives. To close the position control loop, feedback of the actual position is mandatory.

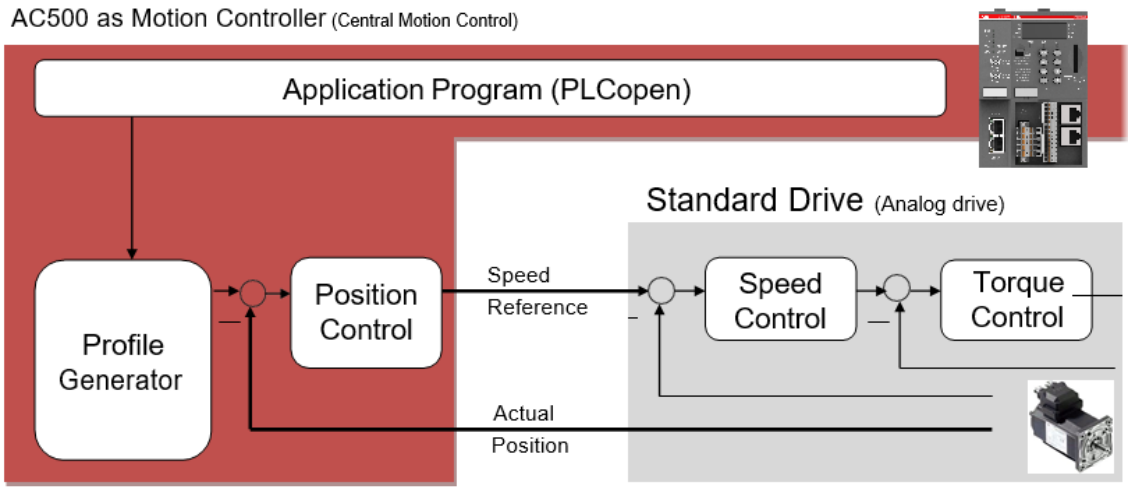


Fig. 50: PLC-based Motion Control with AC500 PLC and PS5611-Motion, closed position control loop

With PLC-based Motion Control it is also possible to include the load control loop to the AC500 PLC. In this case a speed reference signal will be transferred to the drive, which makes it possible to perform the full range of motion functionalities with standard drives. To close the position control loop, feedback of the actual position is mandatory and to close the load control loop, feedback of the actual load / torque is mandatory.

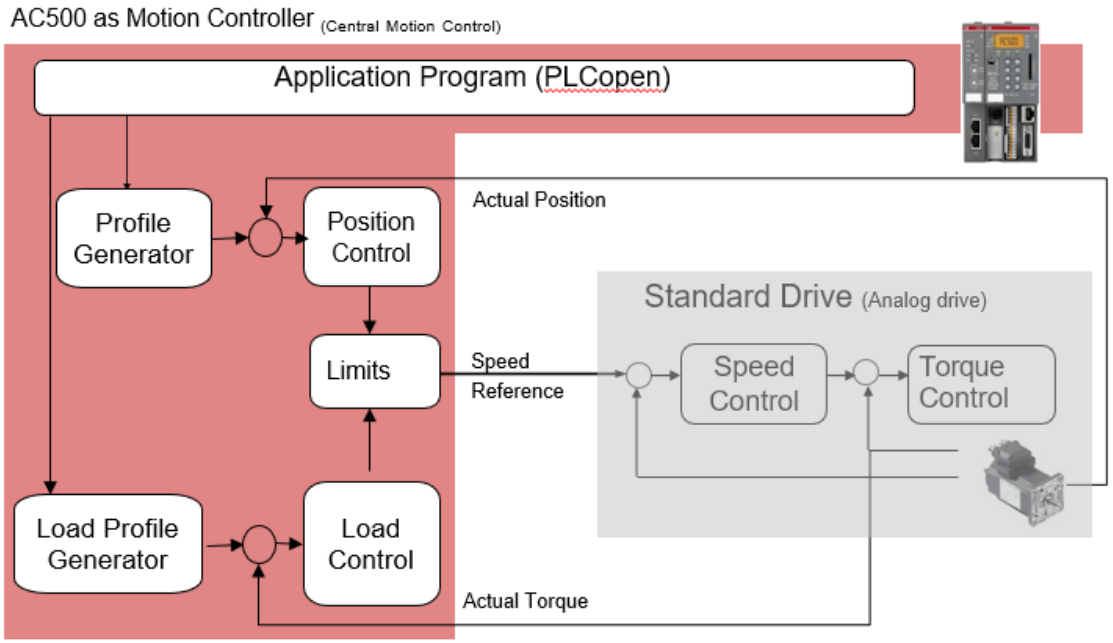


Fig. 51: PLC-based Motion Control with AC500 PLC and PS5611-Motion, closed load control loop

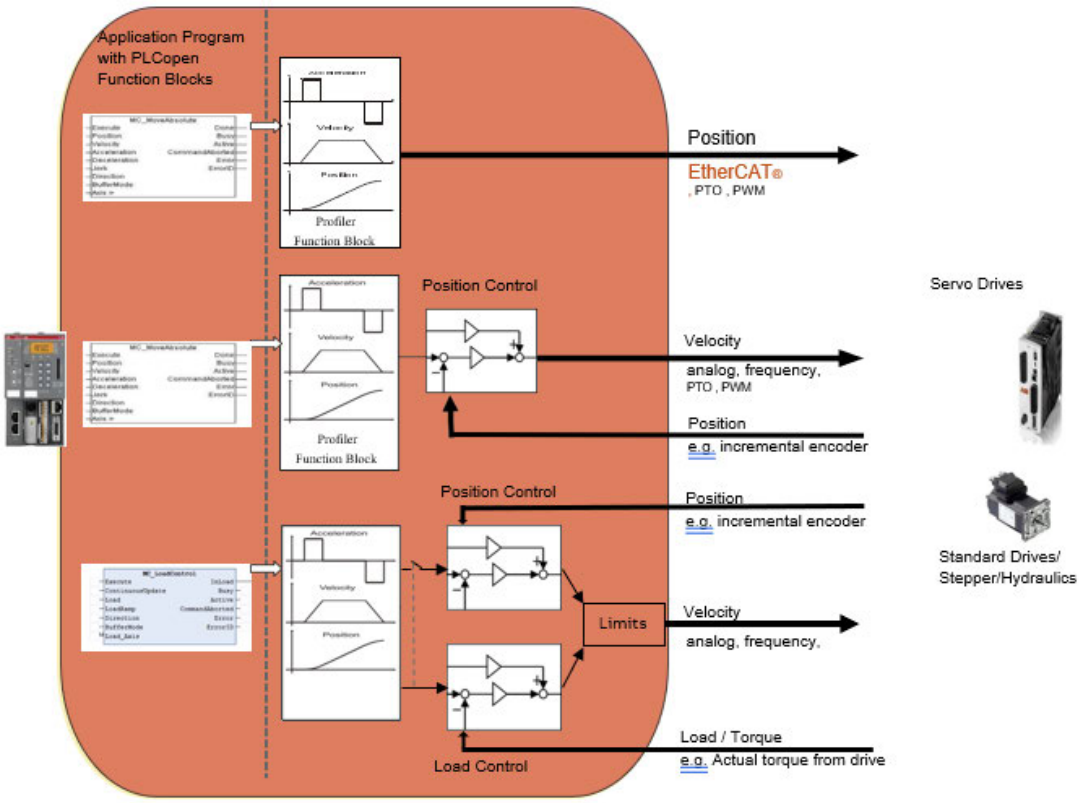


Fig. 52: Central Motion Control with AC500 PLC and PS5611-Motion, different axis implementations at the same time

1.5.10.2.2 Overview of PLCopen function blocks

The following tables give an overview of the defined function blocks, divided into administrative (not driving motion) and motion related sets. They give an overview which function block could be used for the different possible configurations.

These function blocks are part of the ABB_MotionControl_AC500 and ABB_MotionControl-Load_AC500 library.

If there are restrictions concerning a certain drive ("XXX") which lead to a different or limited behavior compared to the standard the respective chapter is supplemented with an additional paragraph "Notes for XXX".

The “KERNEL” function blocks are available in different variants.

The *“CMC_Basic_Kernel”* and *“CMC_Load_Motion_Kernel”* function block is designed to be used in standard V3 PLCs, and can either work with drives connected to a fieldbus or IOs.

The *“OBIO_PTOMotionKernel”* or *“OBIO_PWMMotionKernel”* function blocks (part of AC500_MotionControlEco) are solely to be used in AC500-eCo V3 CPUs and to make use of the integrated stepper-IO along with PLCopen function blocks. It connects automatically to the internal IOs.

For details of the limitations of PTO and PWM outputs in eCo V3 PLCs, refer to Automation Builder help file.

Table 384: Motion control administrative function blocks

Function block	PLC-based Motion Control		
	CMC_Basic_ Kernel	CMC_Load_Motion _Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMotion-Kernel
MC_Power	X	X	X
MC_ReadStatus	X	X	X
MC_ReadAxisError	X	X	X
MC_ReadParameter	X	X	X
MC_ReadBoolParameter	X	X	X
MC_WriteParameter	X	X	X
MC_WriteBoolParameter	X	X	X
MC_Reset	X	X	X
MC_ReadActualPosition	X	X	X
MC_ReadActualVelocity	X	X	X
MC_SetOverride	X	X	X
MC_SetPosition	X	X	X
MC_CamTableSelect	X	X	X

Table 385: Motion control single-axis function blocks

Function block	PLC-based motion control		
	CMC_Basic_K ernel	CMC_Load_Motion _Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMo-tionKernel
MC_MoveAbsolute	X	X	X
MC_MoveRelative	X	X	X
MC_MoveAdditive	X	X	X
MC_MoveSuperimposed	X	X	X
MC_HaltSuperimposed	X	X	X
MC_MoveVelocity	X	X	X
MC_MoveContinuousAbsolute	X	X	X
MC_MoveContinuousRelative	X	X	X
MC_Stop	X	X	X
MC_PositionProfile	X	X	X
MC_VelocityProfile	X	X	X
MC_AccelerationProfile	X	X	X
MC_Halt	X	X	X

Table 386: Motion control multi-axis function blocks

Function block	PLC-based motion control		
	CMC_Basic_Kernel	CMC_Load_Motion_Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMo-tionKernel
MC_CamIn	X	X	X
MC_CamOut	X	X	X
MC_GearIn	X	X	X
MC_GearInPos	X	X	X
MC_GearOut	X	X	X
MC_PhasingAbsolute	X	X	X
MC_PhasingRelative	X	X	X
MC_CombineAxes	X	X	X
MC_HaltPhasing	X	X	X

Table 387: Motion control homing function blocks

Function block	PLC-based motion control		
	CMC_Basic_Kernel	CMC_Load_Motion_Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMo-tionKernel
MC_StepAbsSwitch	X	X	X
MC_StepLimitSwitch	X	X	X
MC_StepRefPulse	X	X	X
MC_StepDirect	X	X	X

Table 388: Motion control ABB specific function blocks

Function block	PLC-based motion control		
	CMC_Basic_Kernel	CMC_Load_Motion_Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMo-tionKernel
MCA_Cam_Extra	X	X	X
MCA_Indexing	X	X	X
MCA_JogAxis	X	X	X
MCA_MoveByExternalRefer-ence	X	X	X
MCA_MoveVelocityContinuous	X	X	X
MCA_Parameter	X	X	X
MCA_ReadParameterList	X	X	X
MCA_WriteParameterList	X	X	X
MCA_SetPositionContinuous	X	X	X
MCA_GearInDirect	X	X	X
MCA_CamInDirect	X	X	X

Function block	PLC-based motion control		
	CMC_Basic_Kernel	CMC_Load_Motion_Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMo-tionKernel
MCA_SetOperatingMode	X	X	X
MCA_CamInfo	X	X	X
MCA_DriveBasedHome	X	X	X
MCA_MoveRelativeOpto	X	X	X
MCA_PhasingByMaster	X	X	X

Table 389: Motion control fluid power function blocks

Function block	PLC-based motion control		
	CMC_Basic_ Kernel	CMC_Load_Motion_ Kernel	OBIO_PTOMotion-Kernel/ OBIO_PWMMo-tionKernel
MC_LimitLoad	-	X	-
MC_LimitMotion	-	X	-
MC_LoadControl	-	X	-
MC_LoadProfile	-	X	-
MC_LoadSuperimposed	-	X	-
MC_TorqueControl	-	X	-

1.5.10.2.3 Overview of libraries

▷ Add the following libraries for the listed applications.

⇨ In some cases by adding a library, there will be other libraries added automatically.

Application	Library to be added manually
PLC-based motion control	ABB_MotionControl_AC500.-compiled-library
	ABB_MathFunctions_AC500.-compiled-library
PLC-based motion control, optional for EtherCAT	ABB_Ecat_CiA402_AC500.-library
Motion control with eCo V3 (OBIO_PTOMo-tionKernel & OBIO_PWMMotionKernel)	ABB_MotionControlEco_AC500.compiled ibrary
PLC-based motion control - Fluid Power Extensions	ABB_MotionControlLoad_AC500.compiled ibrary

The features of the function blocks provided with PS5611-Motion can be used from the PLC program according to PLCopen standard. Different drives and different Motion Control realizations could be used and can be combined with each other as well as different fieldbuses. ABB_Ecat_CiA402_AC500.library is editable and can be adapted based on the drive configuration and drive type.

1.5.10.2.4 Overview of data types

The following data types are used for the Motion Control library. The data types are defined in the library file ABB_MotionControl_AC500 compiled-library. The corresponding elements can be used for the function blocks inputs.

Table 390: Structures

Data type	Elements	Element data type
CMC_AXIS_IO	limitSwitchPos	BOOL
	limitSwitchNeg	BOOL
	absRefSwitch	BOOL
MC_PPROFILE *Chapter 1.5.10.4.6.1 "PositionPositionProfile" on page 2364*	master_position	LREAL
	interpolation_point	LREAL
	velocity_ratio	LREAL
	acceleration_ratio	LREAL
MC_TPROFILE *Chapter 1.5.10.4.6.2 "PositionTimeProfile" on page 2364*	interpolation_point	LREAL
	first_derivative	LREAL
	second_derivative	LREAL
	delta_time	TIME

Table 391: Enum

Data type	Possible values
MC_ABB_iTYPES_ENUM *Chapter 1.5.10.4.6.3 "Interpolation types for profiles" on page 2364*	MCA_SPLINE_COMPLETE
	MCA_SPLINE_NATURAL
	MCA_POLY5
	MCA_POLY3
	MCA_LINEAR
MC_BUFFERMODE	mcABORTING
	mcBUFFERED
	mcBLENDINGlow
	mcBLENDINGprevious
	mcBLENDINGnext
	mcBLENDINGhigh
MC_DIRECTION	DEFAULT
	POSITIVE
	SHORTEST
	NEGATIVE
	CURRENT
	POSITIVE_STOP
	NEGATIVE_STOP
	CURRENT_STOP
MC_HOMING_DIRECTION	MC_SwitchNegative
	MC_SwitchPositive

Data type	Possible values
	MC_Positive
	MC_Negative
MC_HOMING_EDGE	MC_EdgeOn
	MC_EdgeOff
	MC_On
	MC_Off
MC_HOMING_MODE	MC_REFPULSE
	MC_DIRECT
MC_SOURCE	mcActualValue
	mcSetValue
ERROR_ID	MC_Ok
	Wrong_State
	Drive_Problem
	Parameter_Exceeds_Limit
	No_Field_Access
	Bus_Problem
	Abs_Switch_Error
	Timeout
	NAK
	MC_TimeLimitExceeded
	MC_DistanceLimitExceeded
	MC_TorqueLimitExceeded
	Not_Implemented
	ErrorID_POSITION_FOLLOW
	ErrorID_POSSW
	ErrorID_NEGSW
	ErrorID_VELOCITY_FAULT
	ErrorID_INTERPOLATION_FAULT
	ErrorID_WARNING_VELOCITYLIMIT
	ErrorID_WARNING_POSITIONLIMITPOS
	ErrorID_WARNING_POSITIONLIMITNEG
	ErrorID_WARNING_POSITIONOVERRUN
	ErrorID_WARNING_ABORT
	ErrorID_WARNING_MOVEMENT_DIRECTION

1.5.10.2.5 Naming of function blocks and data structures

PLCopen

All function blocks and data types named MC_xxx are implemented according to PLCopen definition and follow the PLCopen documentation. They may have additional inputs but according to PLCopen rules.

All function blocks and data types named MCA_xxx are implemented corresponding to PLCopen rules with adaptations specific to AC500. They are AC500 specific extensions to the PLCopen library.

PLC-based motion control

All function blocks named CMC_xxx belong to the implementation of PLC-based motion control.

All data types named CMC_xxx belong to the implementation of PLC-based motion control.

All data types named AXIS_xxx exist according to PLCopen definition. The content is ABB specific and not documented.

All function blocks named zCMC_xxx belong to the implementation of PLC-based motion control. These are not documented and not intended for customer use.

All function blocks and data types named MC_xxx are implemented according PLCopen definition and follow the PLCopen documentation.

All function blocks and data types named OBIO_xxx in the ABB_MotionControlEco_AC500 library are intended for use with AC500-eCo V3 PLCs only.

All function blocks named xxx_APP are not write protected and may be modified for adaptations. Editable library is available in the example folder.

Editable library is available in the example folder.

1.5.10.3 PLCopen

Based on application requirements and project specifications engineers are required to use or select a wide range of Motion Control hardware. In the past this required unique software to be created for each application even though the functions are the same. PLCopen motion standard provide a way to have standard application libraries that are reusable for multiple hardware platforms. This lowers development, maintenance and support costs while eliminating confusion. In addition, engineering becomes easier, training costs decrease, and the software is reusable across platforms. Effectively, this standardization is done by defining libraries of reusable components. In this way the programming is less hardware dependent, the reusability of the application software increased, the cost involved in training and support reduced, and the application becomes scalable across different control solutions. Due to the data hiding and encapsulation, it is usable on different architectures, for instance ranging from centralized to distributed or integrated to networked control. It is not specifically designed for one application, but will serve as a basic layer for ongoing definitions in different areas. As such it is open to existing and future technologies.

ABB is a member of the PLCopen organization. More Information about PLCopen can be read on the *PLCopen website*.

Fig. 53: PLCopen Motion Control logo

Function blocks according to PLCopen are designed for controlling axes via the language elements consistent with those defined in the IEC 61131-3 standard. It was decided by the task force that it would not be practical to encapsulate all the aspects of one axis into only one function block. The retained solution is to provide a set of command-oriented function blocks that have a reference to the axis, e.g. the abstract data type Axis, which offers flexibility, ease of use and reusability.

Implementations based on IEC 61131-3 (for instance via function blocks and SFC) will be focused towards the interface (look-and-feel/proxy) of the function blocks. This specification does not define the internal operation of the function blocks.

PLCopen Motion Control function blocks can be used in any IEC 61131-3 programming language. The following picture shows an example of a function block used in Function Block Diagram (FBD) language.

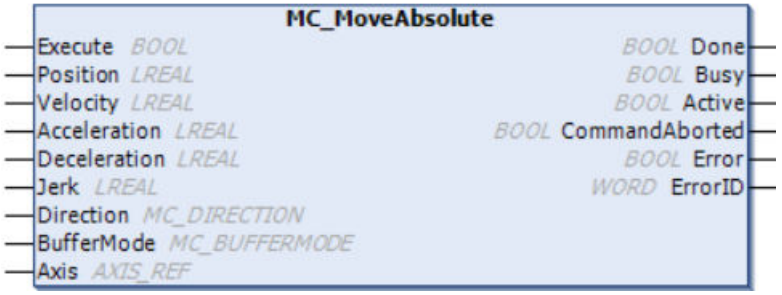


Fig. 54: Command for absolute positioning according to PLCopen standard

Application programs which use the manufacturer independent function blocks according to PLCopen will lead to the following advantages:

- Reusable software structure for different platforms.
- Programming based on function blocks.
- Function blocks can be used in any IEC 61131-3 language.

All function blocks which are defined by PLCopen will have the following qualities independently to the manufacturer of the motion control system:

- Same inputs/outputs
- Same functional behavior
- Same name

The following parts of the PLCopen motion control definition are completely or partly included in this product:

- Part 1: Function blocks for motion control
- Part 2: Extensions
- Part 3: User Guidelines
- Part 4: Homing Procedures
- Part 6: Function blocks for motion control – Fluid Power Extensions

1.5.10.3.1 Programming guidelines

This chapter explains some rules on the usage of the libraries and the structure Axis_Ref.

- In general, the kernel function block and the transfer of axis IO data should be processed in a cyclic task. This task should be as short and real-time as possible to achieve the best motion control performance. Always make sure Kernel function block is called at the highest priority task and other applications must be at a lower priority task.
- If Axis_Ref is used as input on a user defined function block or program or function, always use it as VAR_IN_OUT and never use it as VAR_INPUT or VAR_OUTPUT. The reason is that this would
  - Break the consistency and destroy data.
  - Consume a lot of computing power by copying data.

- Any instance of a function block should be called only once per cycle and in only one specific task.
  If the instance is used in several tasks, it has to be checked that is not called several times.
  Because this could corrupt the handshake from function block to Axis_Ref to CMC_Basic_Kernel and vice versa.
- Some PLCopen function blocks are only allowed to be called within the same task as the CMC_Basic_Kernel function block. This is mentioned in the function block descriptions.
- If PLCopen function blocks are called from a different task, the cycle time should be at least 2 times the cycle time for CMC_Basic_Kernel function block.

Axis data type Axis_Ref

The Axis_Ref is a structure that contains information on the corresponding axis. It is used as a VAR_IN_OUT in all Motion Control function blocks defined in this document. The content of this structure is implementation dependent and can ultimately be empty. If there are elements in this structure, the supplier shall support the access to them, but this is outside of the scope of this document. The refresh rate of this structure is also implementation dependent. According to IEC 61131-3 it is allowed to switch the Axis_Ref for an active function block, for instance from Axis1 to Axis2. However, the behavior of this can vary across different platforms, and is not encouraged to do.

Axis_Ref data type declaration:

TYPE Axis_Ref : STRUCT

(Content is implementation dependent)

END_STRUCT

Example:

TYPE Axis_Ref : STRUCT

AxisNo: UINT; AxisName: STRING (255);

…….

END_STRUCT

1.5.10.3.2 The single axis state diagram

The following diagram normatively defines the behavior of the axis at a high level when multiple motion control function blocks are simultaneously activated. This combination of motion profiles is useful in building a more complicated profile or to handle exceptions within a program. (In real implementations there may be additional states at a lower level defined). The basic rule is that motion commands are always taken sequentially, even if the PLC had the capability of real parallel processing. These commands act on the axis' state diagram.

The axis is always in one of the defined states (see diagram below). Any motion command that causes a transition changes the state of the axis and, as a consequence, modifies the way the current motion is computed. The single axis state diagram is an abstraction layer of what the real state of the axis is, comparable to the image of the I/O points within a cyclic (PLC) program. A change of state is reflected immediately when issuing the corresponding motion command.

The response time of immediately is system dependent, coupled to the state of the axis, or an abstraction layer in the software.

The diagram is focused on a single axis. The multiple axis function blocks, MC_CamIn, MC_GearIn and MC_Phasing, can be looked at, from a single axis state diagram point of view, as multiple single-axes all in specific states. For instance, the CAM-master can be in the state Continuous Motion. The corresponding slave is in the state Synchronized Motion. Connecting a slave axis to a master axis has no influence on the master axis.

The state Disabled describes the initial state of the axis. In this state the movement of the axis is not influenced by the function blocks. The axis feedback is operational. If the MC_Power function block is called with Enable=TRUE while being in state Disabled, this either leads to Standstill if there is no error inside the axis, or to ErrorStop if an error exists.

Calling MC_Power with Enable=FALSE in any state, the axis goes to the state Disabled, either directly or via any other state. If a motion generating function block controls an axis, while the MC_Power function block with Enable=FALSE is called, the motion generating function block is aborted (CommandAborted).

The intention of the state ErrorStop is that the axis goes to a stop, if possible. There are no further inputs from function blocks accepted until a reset has been done from the ErrorStop state.

The transition Error refers to errors from the axis and axis control, and not from the function block instances. These axis errors may also be reflected in the output of the function blocks instances errors.

Issuing MC_Home in any other state than StandStill will go to ErrorStop, even if MC_Home is issued from the state Homing itself.

Function blocks which are not listed in the single axis state diagram do not affect the state of the axis, meaning that whenever they are called the state does not change. They are:

MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_ReadActualPosition and MC_CamTable-Select.

Calling the function block MC_Stop in state StandStill changes the state to Stopping and back to Standstill when Execute = FALSE. The state Stopping is kept as long as the input Execute is TRUE. The output Done is set when the stop ramp is finished.

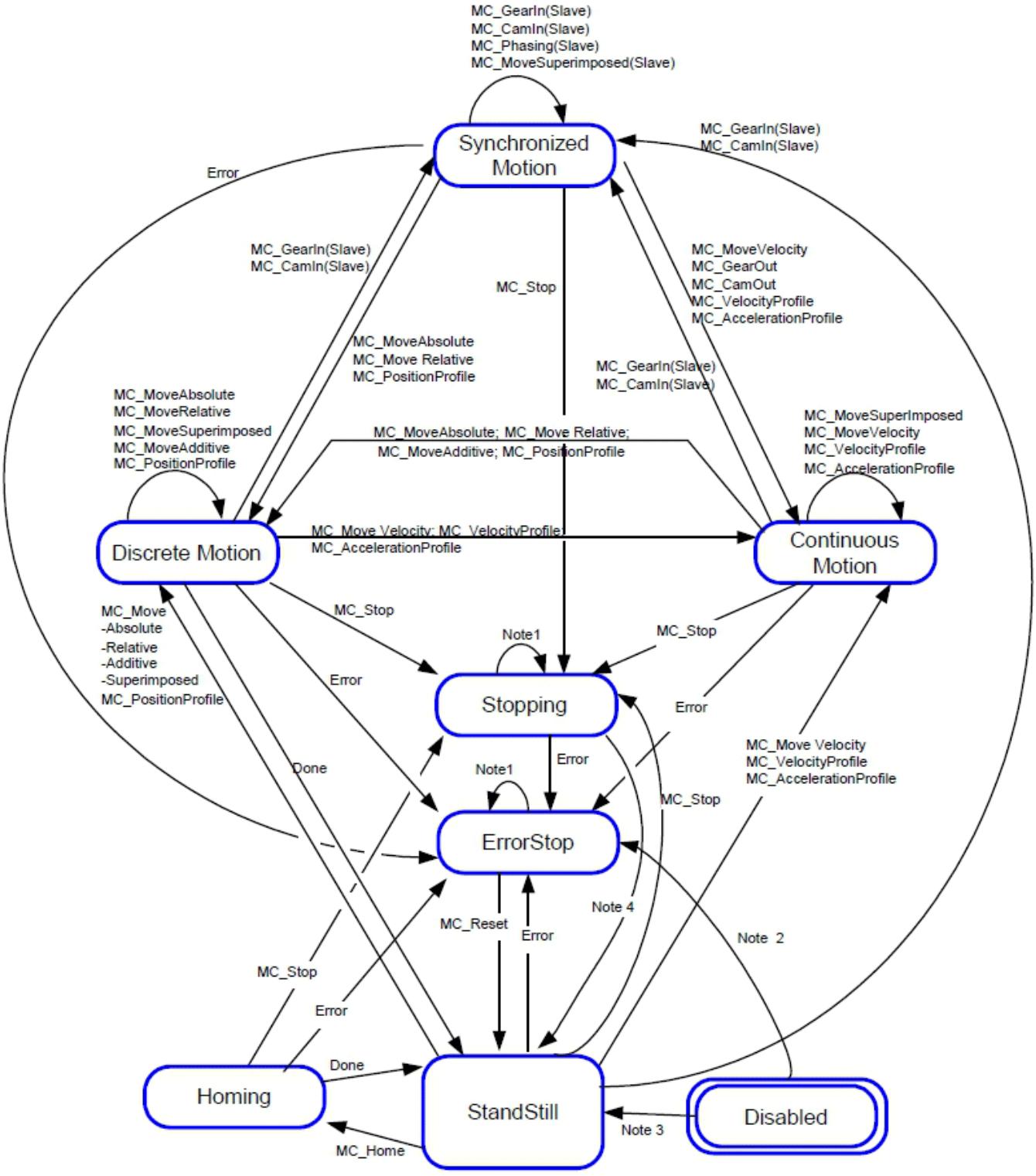


Fig. 55: Function block state behavior

1. In this state ErrorStop or Stopping, all function blocks can be called, although they will not be executed, except MC_Reset and Error – they will generate the transition to StandStill or ErrorStop respectively.

2. Power.Enable=TRUE and there is an error in the Axis.

3. Power.Enable=TRUE and there is no error in the Axis.

4. MC_Stop.Done AND NOT MC_Stop.Execute.

1.5.10.3.3 Visualizations

For usage with the PLCopen Library, a set of visualization objects is defined. These visualizations use the placeholder concept, which means that they could be used in an actual visualization several times and be instantiated by replacing the “placeholder” with an effective data-structure.

Two types of visualizations exist:

- As placeholder, an instance of Axis_Ref should be used. These are named: MC_VISU_Axis_name. Here the name could be state machine or its actual.
- As placeholder, an instance of the respective PLCopen function block should be used. These visualizations are named MC_VISU_FB_name where "name" could be MoveAbsolute or MoveVelocity, so the complete element is named MC_VISU_FB_MoveAbsolute or MC_VISU_FB_MoveVelocity.

The background colour and the colour for the title of each element could be changed. The colours are defined in some global predefined variables in MC_VISU_COLOR_INFORMATION. By changing these values, different colours will be used.

MC_VISU_COLOR_INFORMATION (GVL)

InOut:

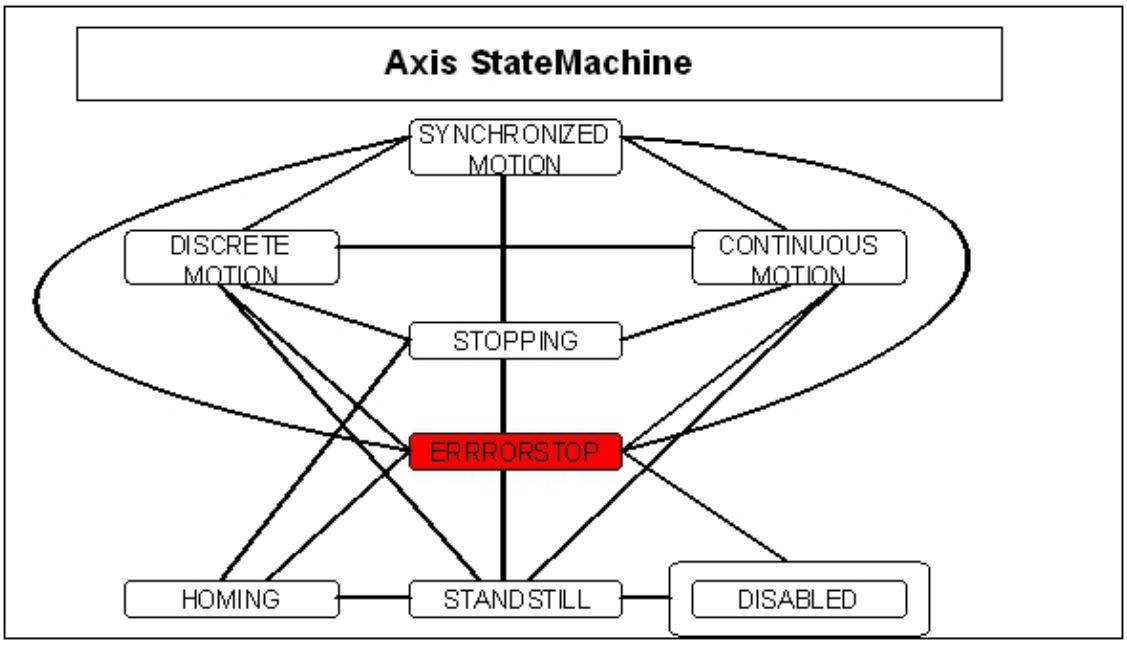
Name	Type	Initial	Comment
MC_VISU_BACKGROUND_COLOR	DWORD	16#FF888888	16#FFRRGGBB (* Color combination for the Grey color*)
MC_VISU_TITLE_COLOR	DWORD	16#FFFFFF00	16#FFRRGGBB (* Color combination for yellow*)

Below, some existing visualizations are shown.

MC_VISU_Axis_StateMachine

This shows the state machine of the axis according to PLCopen definition. The active state is shown green except the ErrorStop which is shown red. Usually, it starts with Disabled. When no remote connection to the drive is available, it will switch to ErrorStop immediately.

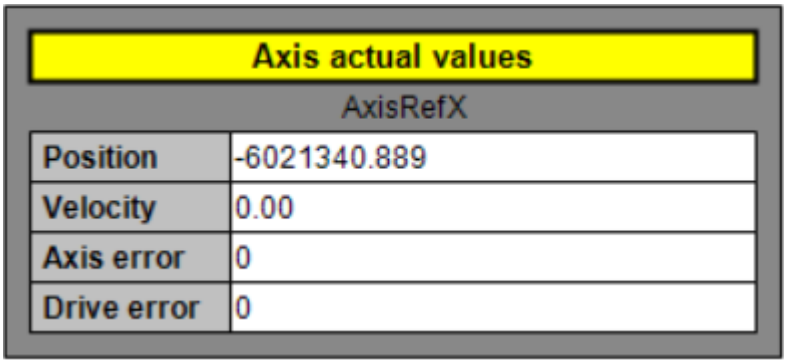
The placeholder of this visualization has to be connected to an instance of the data type Axis_Ref.



MC_VISU_Axis_actual

This object shows some actual values.

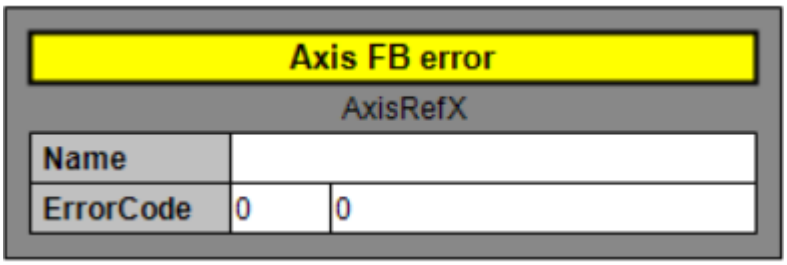
The Placeholder of this visualization has to be connected to an instance of the data type Axis_Ref.



MC_VISU_Axis_FB_error

This object shows the error information connected to the PLCopen function blocks. This is NOT a drive error. If no error occurs in the execution of a function block, just the name is shown. If an error occurred, it shows the name of the function block as well as the error number and a short description. In the example below, the MC_Power function block recognized that no fieldbus connection to the drive was available.

The Placeholder of this visualization has to be connected to an instance of the data type Axis_Ref.



1.5.10.3.4 Error codes

Besides the diagnosis information of the drive which is described in the respective drive documentation, there are a number of error codes directly related to the function blocks. These error codes are displayed at the output “ErrorID” of the function block.

Error Code	Mnemonic	Explanation
0	MC_Ok	No Error
1	WRONG_STATE	A function block was activated not according to the state machine, e.g. tried to start a movement while in state Disabled.
2	DRIVE_PROBLEM	The drive indicates an error, e.g. tripped.
3	PARAM-ETER_EXCEEDS_LIMIT	A parameter at the function block is outside the possible range. This does not refer to the parameter range which is allowed for the drive but just to the 32-Bit Integer which is used for internal calculation.
4	NO_FIELD_ACCESS	No fieldbus connection to the drive.
5	BUS_PROBLEM	Not used
6	ABS_SWITCH_ERROR	During Homing, (when done by function blocks) limit switch not according to moving direction e.g. the positive switch occurred when moving in negative direction.
7	TIMEOUT	Timeout in block execution.

Error Code	Mnemonic	Explanation
8	NAK	Parameter access not applicable
9	MC_TimeLimitExceeded	Used by function blocks with TimeLimit.
10	MC_DistanceLimitExceeded	Used by function blocks with DistanceLimit.
11	MC_TorqueLimitExceeded	Used by function blocks with TorqueLimit.
12	NOT_IMPLEMENTED	Functionality not implemented for certain axis type.
101	ErrorID_POSITION_FOLLOW	Following error, caused by > position error => ERRORSTOP. (parameter POS_LAG_PERCENTAGE)
102	ErrorID_POSSW	Positive software limit switch => ERRORSTOP. The actual position did exceed the positive Software limit switch position. This supervision has to be activated with MC_WriteParameter.
103	ErrorID_NEGSW	Negative software limit switch => ERRORSTOP. The actual position did exceed the negative Software limit switch position. This supervision has to be activated with MC_WriteParameter.
104	ErrorID_VELOCITY_FAULT	The measured velocity and commanded velocity are > 50% (related to maximum velocity) apart, for a certain time =>ERRORSTOP (parameter V_CHECKTIME)
105	ErrorID_INTERPOLATION_FAULT	following error, caused by interpolation problem =>ERRORSTOP. Position following error occurred, but reason most likely a interpo- lation problem, not drive problem (e.g. CAM Table, position step).
110	ErrorID_WARNING_VELOCITYLIMIT	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity, MaxVelocityAppl, MaxDecelerationAppl
111	ErrorID_WARNING_POSITIONLIMITPOS	Position is in limitation towards position limit (SWLimit2DecPos), axis decelerates near positive software limit switch
112	ErrorID_WARNING_POSITIONLIMITNEG	Position is in limitation towards position limit (SWLimit2DecNeg)., axis decelerates near negative software limit switch
113	ErrorID_WARNING_POSITIONOVERRUN	A linear axis created a 32bit position overrun (> 2147483647 u=>inc) =>configure modulo
114	ErrorID_WARNING_ABORT	Axis aborted due to too large position gap due to velocity limitation
115	ErrorID_WARNING_MOVEMENT_DIRECTION	Either positive or negative direction blocked by MC_Power

1.5.10.3.5 Error handling

All access to the drive/motion control is via function blocks. Internally these function blocks provide basic error checking on the input data. Exactly, how this is done is implementation dependent. For instance, if MaxVelocity is set to 6000, and the Velocity input to a function block is set to 10,000, a basic error report is generated. In the case where an intelligent drive is coupled via a network to the system, the MaxVelocity parameter is probably stored on the drive. The function block must take care of the errors generated by the drive internally. With another implementation, the MaxVelocity value could be stored locally. In this case the function block will generate the error locally.

Both centralized and decentralized error handling methods are possible when using the motion control function blocks.

Centralized error handling is used to simplify programming of the function block. Error reaction is the same independent of the instance in which the error has occurred.

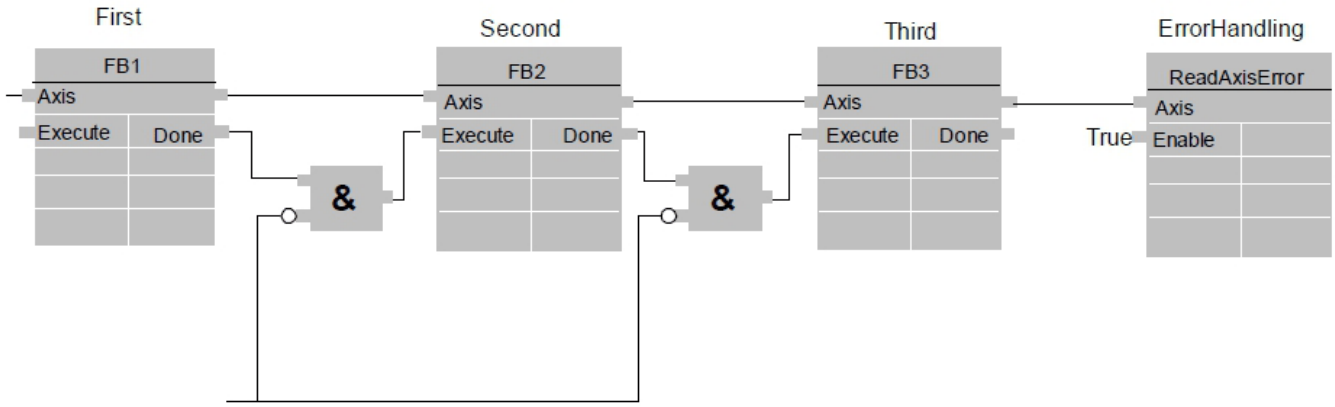


Fig. 56: Function blocks with centralized error handling

Decentralized error handling gives the possibility of different reactions depending on the function block in which an error occurred.

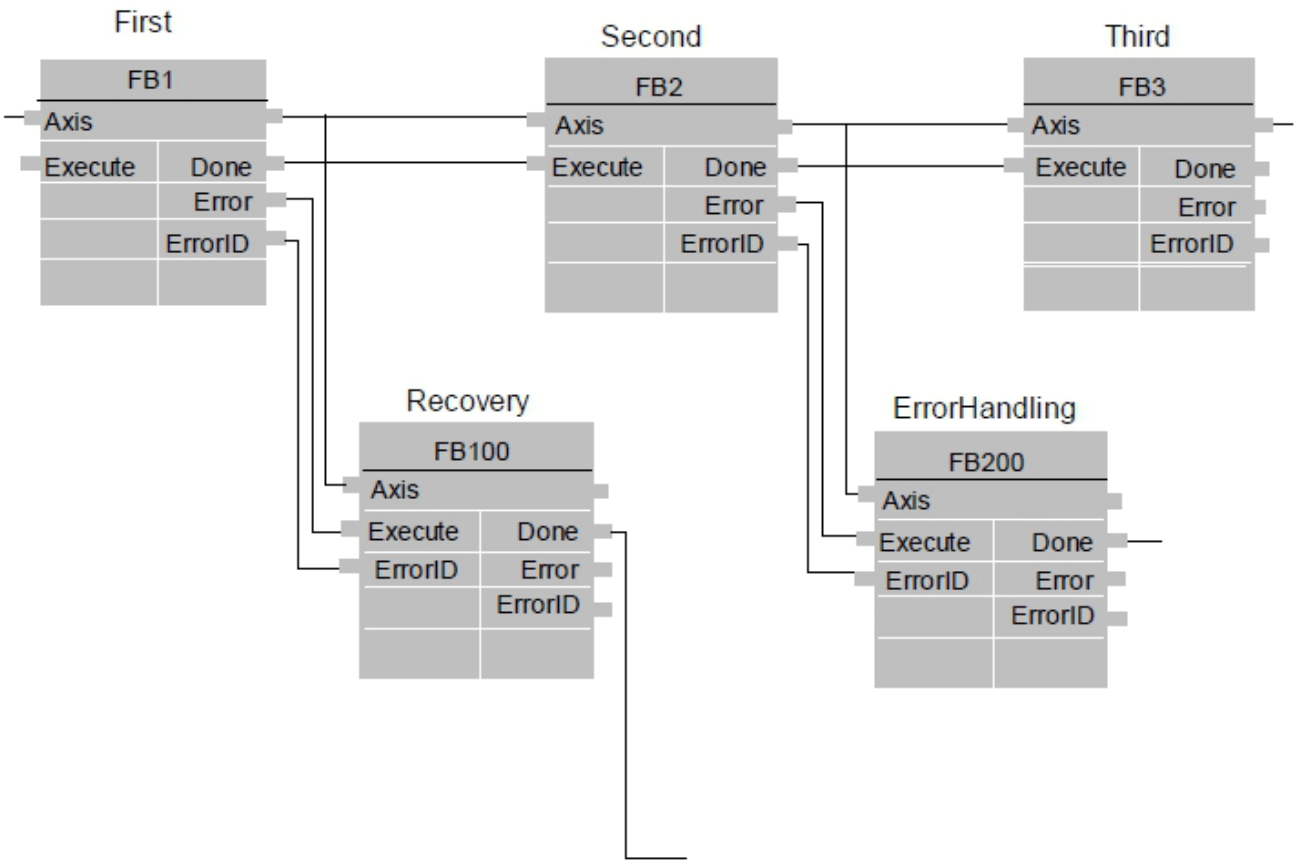


Fig. 57: function blocks with decentralized error handling

1.5.10.3.6 PLCopen parameter

Additional parameters are available by ReadParameter and WriteParameter function blocks.

Following function blocks can be used for the read and write operation. Functionality of these blocks and its variables are explained in the integrated documentation

- *MC_ReadParameter*
- *MC_WriteParameter*
- *MC_ReadBoolParameter*
- *MC_WriteBoolParameter*

Parameter number (PN)	Name	Datatype	Min.	Max.	Default	R/W	Comments
1	CommandedPosition	DINT				R	Commanded position.
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position.
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position.
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch.
5	EnableLimitNeg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch.
6	EnablePosLagMonitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error).
7	MaxPositionLag	DINT	1	2147483647 ***		R	Maximal position lag.
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system.
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application.
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity.
11	CommandedVelocity	DINT	-32767	32767		R	Commanded velocity.
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system.

Parameter number (PN)	Name	Datatype	Min.	Max.	Default	R/W	Comments
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application.
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis.
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis.
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis.
2001	MODULO_NOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for PLC-based Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DENOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for PLC-based Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate

Parameter number (PN)	Name	Datatype	Min.	Max.	Default	R/W	Comments
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement FALSE = Limits position and velocity, decelerates and shows a warning until the position limit is reached, then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP
2005	EnableLimtVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

0* means: no limitation of jerk is performed.

**Axis will stay in stop.

***is modified by CMC_Axis_Control_Parameter, the max. Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

In addition to the above parameters certain other operation can be done using the below parameters from the data type "Axis_Parameter"

Name	Type	Initial	Comment
paraFilterVariant	INT		Filter for actual velocity 0 = PT1 1 = LinearRegression
paraFilterTime	INT	10	Time in PLC cycles, used with paraFilter-Variant
paraFilterForecast	INT	0	Time in PLC cycles, used with paraFilter-Variant = 1
paraReverseDirection	INT	0	Changes the direction for actual and reference positions based on the mode selected. 0 = normal direction 1 = reverse input position 2 = reverse output position and speed reference 3 = reverse both
paraEarlyClosedLoop	BOOL	FALSE	TRUE: hold the position when Drive_Release is set (not wait for Drive_InOperation = TRUE)
paraLateOpenLoop	BOOL	FALSE	TRUE: hold the position until Drive_InOperation = FALSE

1.5.10.3.7 Limits

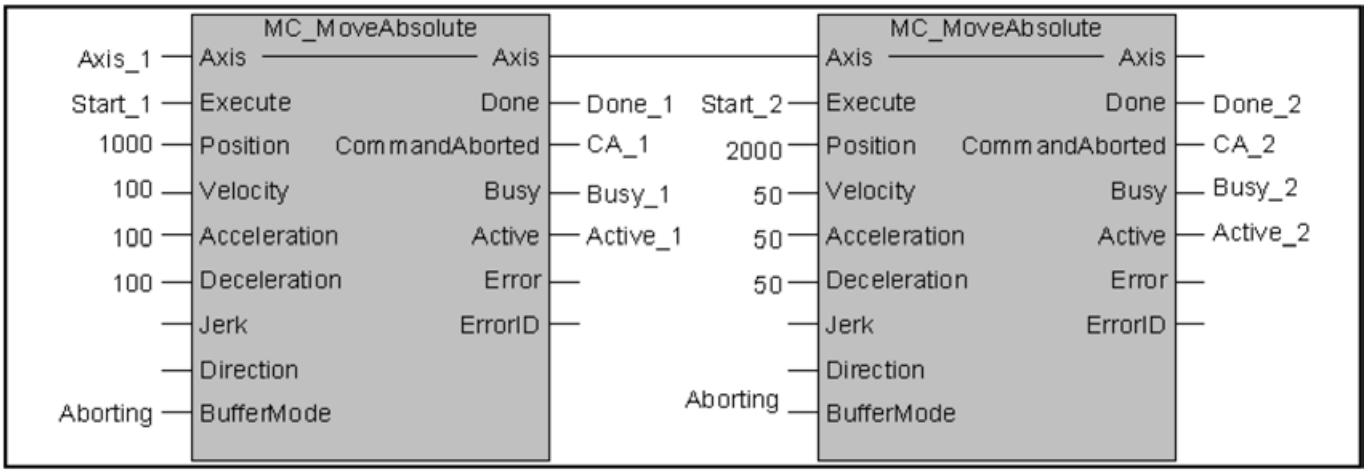
Table 392: Limitations for the inputs of PLCopen function blocks when used with CMC_Basic_Kernel

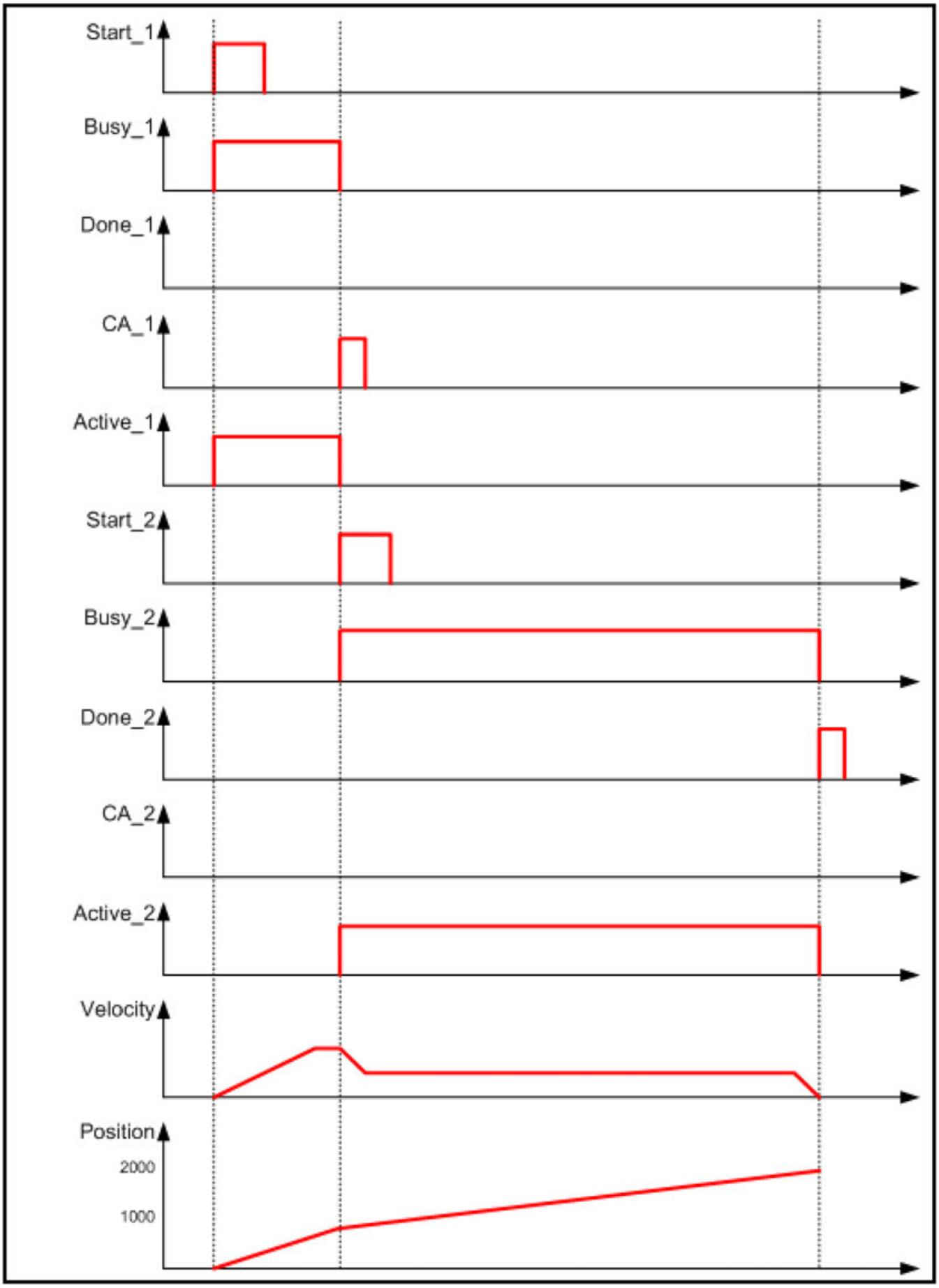
Parameter	Min.	Max.
Velocity	0	x
Acceleration, Deceleration	0	x
Position	-2147483647	2147483647

1.5.10.3.8 General restrictions

Restrictions for the available function blocks

- As buffered mode, MC_Aborting is realized as a default. This does NOT mean that the axis stops when another movement is started while an ongoing movement is still active. It means instead that the new movement will take control immediately and change the velocity to its own velocity by using its own acceleration or deceleration.
- The buffered mode MC_Buffered could be reached with using the axis state StandStill as enable signal for the Execute of the next block.
- From the Extended Inputs and Outputs at the function blocks, the following are not realized:
  - BufferedMode: The realization just supports the MC_Aborting mode.
  - The following Outputs at ReadStatus are not supported: ConstantVelocity, Accelerating and Decelerating.
  - TorqueLimit for Homing function blocks.




Start_1
Busy_1
Done_1
CA_1
Active_1
Start_2
Busy_2
Done_2
CA_2
Active_2
Velocity
Position
2000
1000

MC_Aborting Mode

The diagram shows the behavior with BufferMode MC_Aborting, which is the only available BufferMode. When the second Block is activated, it will take control and will continue on its own velocity. The velocity is changed by using the acceleration value from the second function block. The movement will not be stopped in between. The first function block shows CommandAborted when the second function block is activated.

MC_Buffered

A behavior according to BufferMode MC_Buffered could be reached by using the Done output from the first function block to enable the Execute of the second function block.

1.5.10.3.9 Behavior of the function block inputs and outputs

General rules

Table 393: General rules

Output exclusivity	The outputs Busy, Done, Error, and CommandAborted are mutually exclusive: Only one of them can be TRUE on one function block. If Execute is TRUE, one of these outputs has to be TRUE. Only one of the outputs Active, Error, Done and CommandAborted is set at the same time.
Output status	The outputs Done, InGear, InSync, InVelocity, Error, ErrorID and CommandAborted are reset with the falling edge of Execute. However, the falling edge of Execute does not stop or even influence the execution of the actual function block. It must be guaranteed that the corresponding outputs are set for at least one cycle if the situation occurs, even if execute was reset before the function block completed. If an instance of a function block receives a new execute before it has finished (as a series of commands on the same instance), the function block will not return any feedback, like Done or CommandAborted, for the previous action.
Input parameters	The parameters are used with the rising edge of the execute input. To modify any parameter, it is necessary to change the input parameter(s) and to trigger the motion again.
Missing input parameters	According to IEC 61131-3, if any parameter of a function block input is missing (open) then the value from the previous invocation of this instance will be used. In the first invocation the initial value is applied.
Position versus distance	Position is a value defined within a coordinate system. Distance is a relative measure related to technical units. Distance is the difference between two positions.
Sign rules	Velocity, Acceleration, Deceleration and Jerk are always positive values. Position and Distance can be both positive and negative.
Error Handling Behavior	All function blocks have two outputs, which deal with errors that can occur while executing that function block. These outputs are defined as follow:
	Error Rising edge of Error informs that an error occurred during the execution of the function block. ErrorID: Error number

	The outputs Done, InVelocity, InGear, and InSync mean successful completion so these signals are logically exclusive to Error. Types of errors: ● Function blocks (e.g. parameters out of range, state machine violation attempted), ● Communication, ● Drive Instance errors do not always result in an axis error (bringing the axis to StandStill). The error outputs of the relevant function block are reset with falling edge of Execute.
Function block naming	In case of multiple libraries within one system (to support multiple drive/ motion control systems), the function block naming may be changed to MC_FunctionBlockName_SupplierID.
Behavior of Done output	The outputs Done, InGear, InSync... are set when the commanded action has been completed successfully. With multiple function blocks working on the same axis in a sequence, the following applies: When one movement on an axis is interrupted with another movement on the same axis without having reached the final goal, Done of the first function block will not be set.
Behavior of CommandAborted output	CommandAborted is set, when a commanded motion is interrupted by another motion command. The reset-behavior of CommandAborted is like that of Done. When CommandAborted occurs, the other output-signals such as **InVelocity** are reset.
Inputs exceeding application limits	If a function block is commanded with parameters which result in a violation of application limits, the instance of the function block generates an error. The consequences of this error for the axis are application specific and thus should be handled by the application program.
Behavior of Busy output	Every function block can have an output Busy, reflecting that the function block is not finished. Busy is SET at the rising edge of Execute and RESET when one of the outputs Done, Aborted, or Error is set. It is recommended that this function block should be kept in the active loop of the application program for at least as long as Busy is true, because the outputs may still change. For one axis, several function blocks might be busy, but only one can be active at a time. Exceptions are MC_SuperImposed and MC_Phasing, where more than one function block related to one axis can be active.
Output **Active**	The output Active is required on buffered function blocks. This output is set at the moment the function block takes control of the motion of the according axis. For un-buffered mode the outputs Active and Busy can have the same value.
Enable and Valid/Status	The input Enable is coupled to output Valid. Enable is level sensitive, and Valid shows that a valid set of outputs is available at the function block. The output Valid is TRUE as long as an output value of Valid is available and the input Enable is TRUE. The relevant output value can be refreshed as long as the input Enable is TRUE. If there is a function block error, the output is not Valid (Valid set to FALSE). When the error condition disappears, the values will reappear and output Valid will be set again.

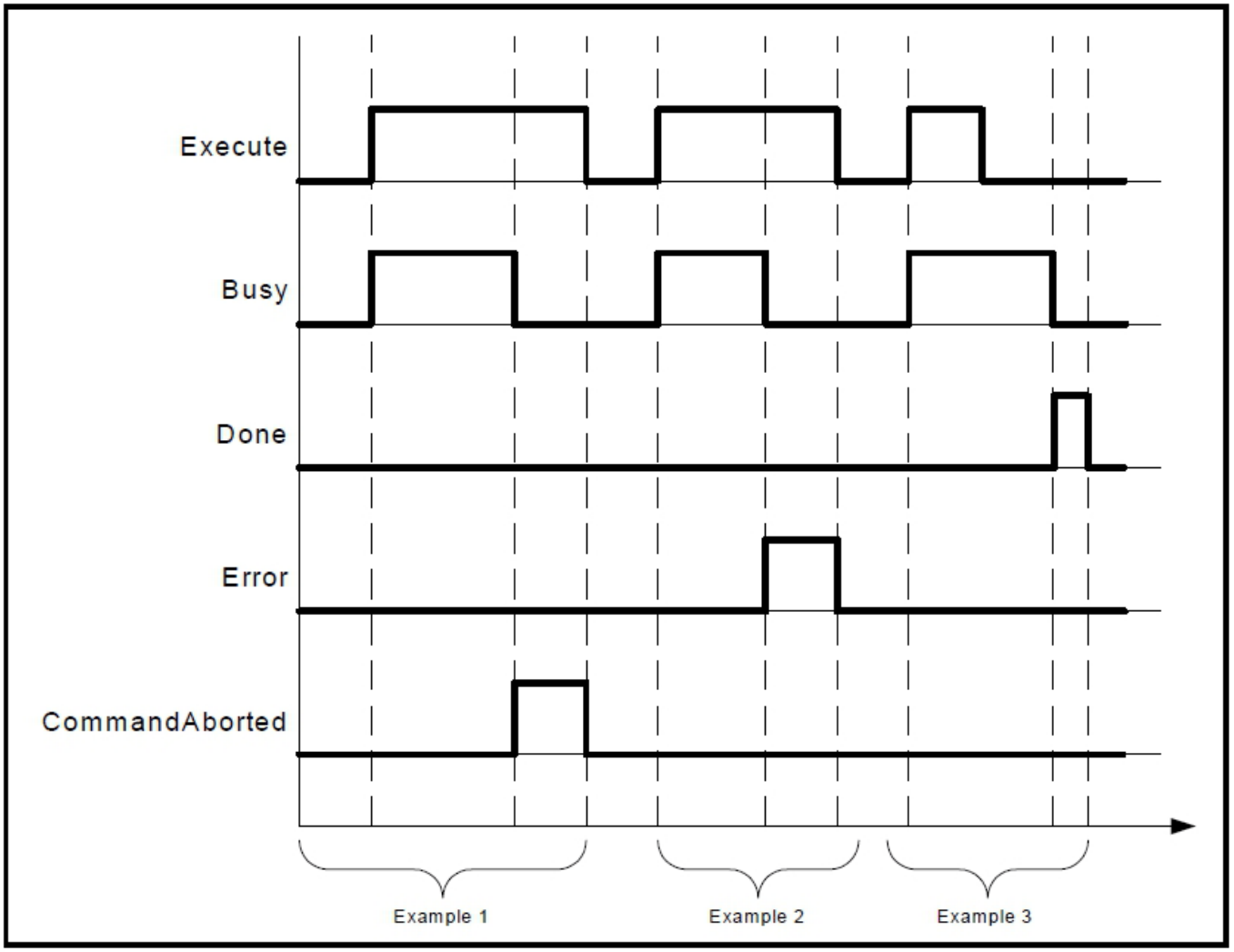


Fig. 58: Behavior of the Execute/Done style function blocks.

Why is the command input edge sensitive?

The input Execute for the different function blocks described in this document always triggers the function with its rising edge. The reason for this is that with edge triggered Execute new input values may be commanded during execution of a previous command. The advantage of this method is a precise management of the instant a motion command is performed. Combining different function blocks is then easier in both centralized and decentralized models of axis management. The output Done can be used to trigger the next part of the movement. The example given below is intended to explain the behavior of the function block execution.

The following figure illustrates the sequence of three function blocks First, Second and Third controlling the same axis. These three function blocks could be for instance various absolute or relative move commands. When First is completed the motion its rising output First.Done triggers Second.Execute. The output Second.Done AND In13 triggers the Third.Execute.

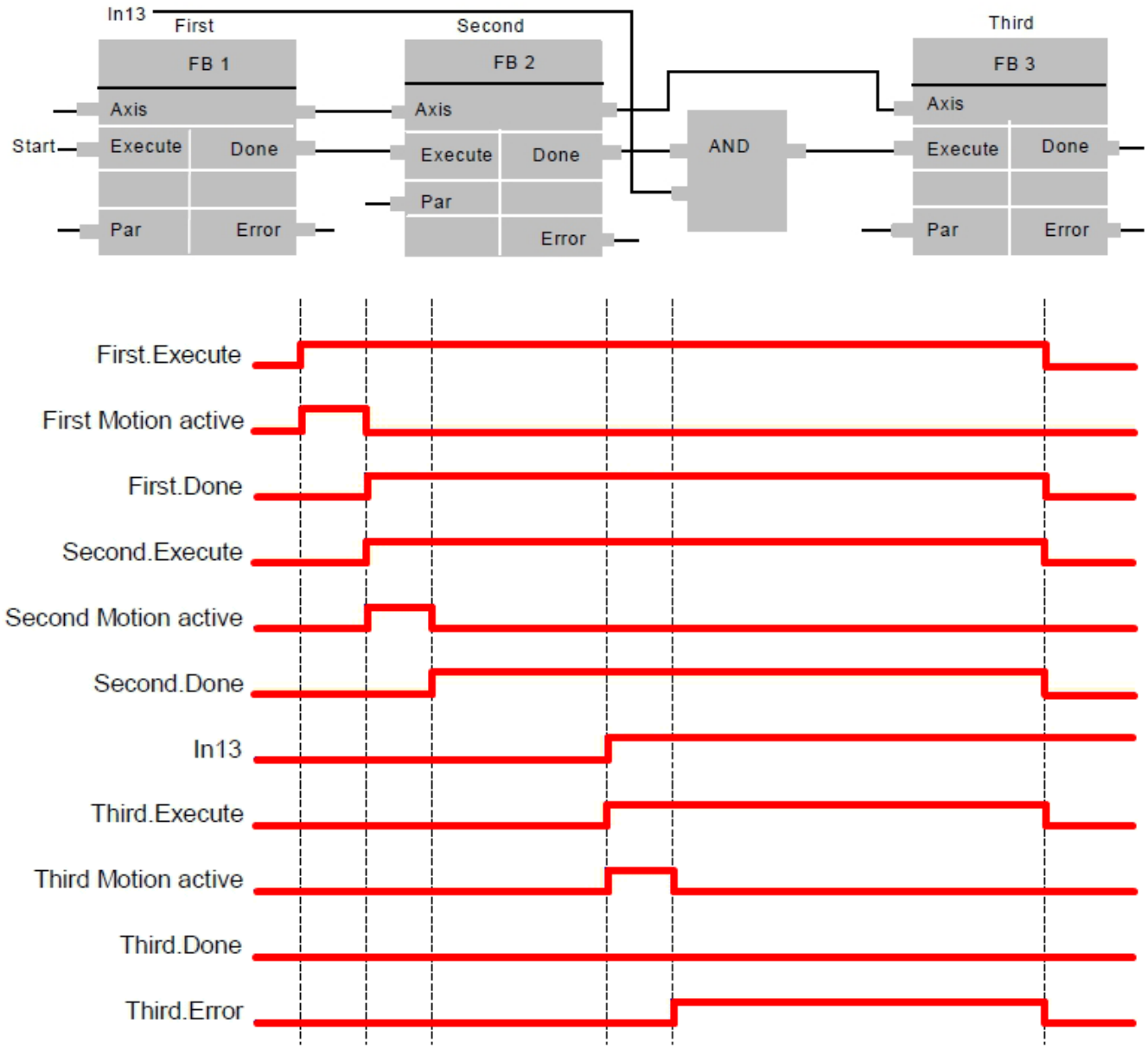


Fig. 59: Function blocks to perform a complex movement

The input ContinuousUpdate

Like described in the previous chapter, the input Execute triggers a new movement. With a rising edge of this input the values of the other function block inputs are defining the movement. Until version 1.1 of PLCopen there was the general rule that a later change in these input parameters does not affect the ongoing motion.

Nevertheless, there are numerous application examples, where a continuous change of the parameters is needed. The user could retrigger the input Execute of the function block, but this complicated the application.

Therefore, the input ContinuousUpdate has been introduced. It is an extended input to all applicable function blocks. If it is TRUE, when the function block is triggered (rising Execute), it will - as long as it stays TRUE – make the function block use the current values of the input variables and apply it to the ongoing movement. This does not influence the general behavior of the function block nor does it impact the single axis state diagram. In other words it only influences the ongoing movement and its impact ends as soon as the function block is no longer Busy or the input ContinuousUpdate is set to FALSE.

It can be that certain inputs like BufferMode are not really intended to change every cycle. However, this has to be dealt with in the application, and is not forbidden in the specification

If ContinuousUpdate is FALSE with the rising edge of the input Execute, a change in the input parameters is ignored during the whole movement and the original behavior of previous versions is applicable. The ContinuousUpdate is not a retriggering of the input Execute of the function block. A retriggering of a function block which was previously aborted, stopped, or completed, would regain control on the axis and modify its single axis state diagram. Opposite to this, the ContinuousUpdate only effects an ongoing movement. Also, a ContinuousUpdate of relative inputs (e.g. Distance in MC_MoveRelative) always refers to the initial condition (at rising edge of Execute).

Example

- MC_MoveContinuousRelative is started at Position 0 with Distance 100, Velocity 10 and ContinuousUpdate set TRUE. Execute is Set and so the movement is started to position 100.
- While the movement is executed (let the drive be at position 50), the input Distance is changed to 130, Velocity 20.
- The axis will accelerate (to the new Velocity 20) and stop at Position 130 and set the output Done and does not accept any new values.

1.5.10.3.10 Unit of length

The only specification for physical quantities is made on the unit of length (noted as [u]) that is to be coherent with its derivatives i.e. (velocity [u/s]; acceleration [u/s^2]; jerk [u/s^3]). Nevertheless, the unit [u] is not specified (manufacturer dependent). Only its relations with others are specified.

1.5.10.3.11 Aborting versus buffered modes

Some of the function blocks have an input called BufferMode. With this input, the function block can either work in a Non-buffered mode (default behavior) or in a Buffered mode. The difference between those modes is when they should start their action:

- A command in a non-buffered mode acts immediately, even if this interrupts another motion,
- A command in a buffered mode waits till the current function block sets its output Done (or InPosition, InVelocity...).
- The library just supports the mode "aborting" (MCAborting)

The following examples describe the different behavior of these modes:

Example 1: Standard behavior of two following absolute movements

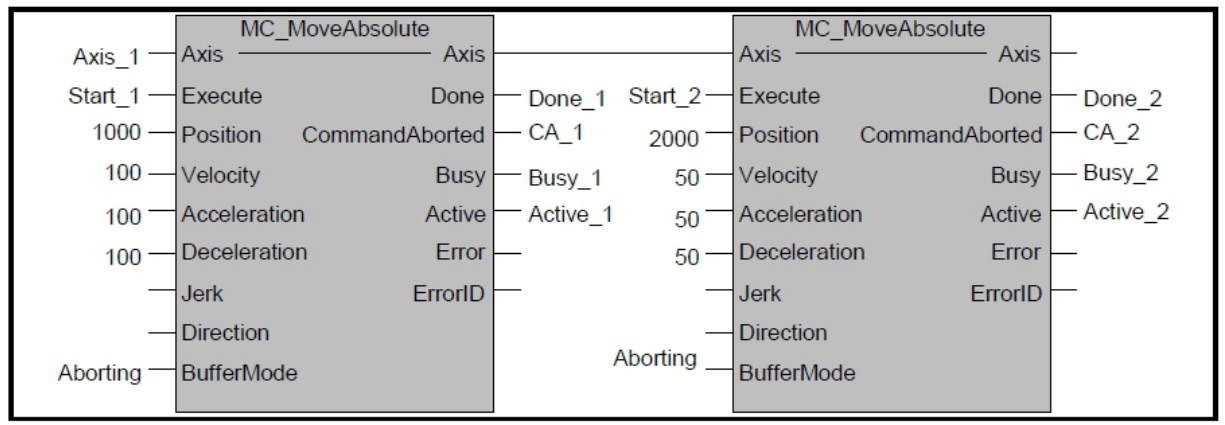


Fig. 60: Basic example with two MC_MoveAbsolute on same axis

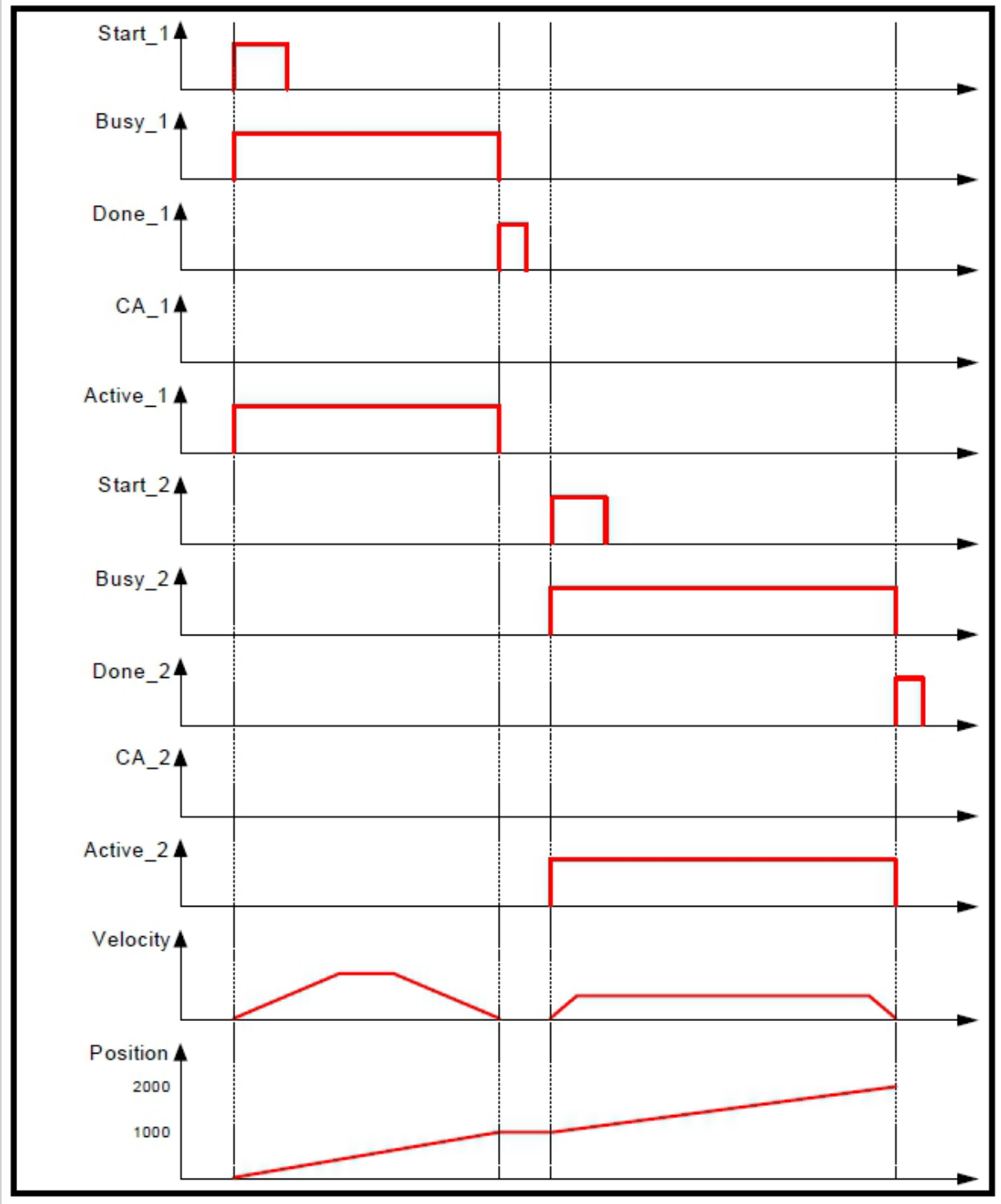
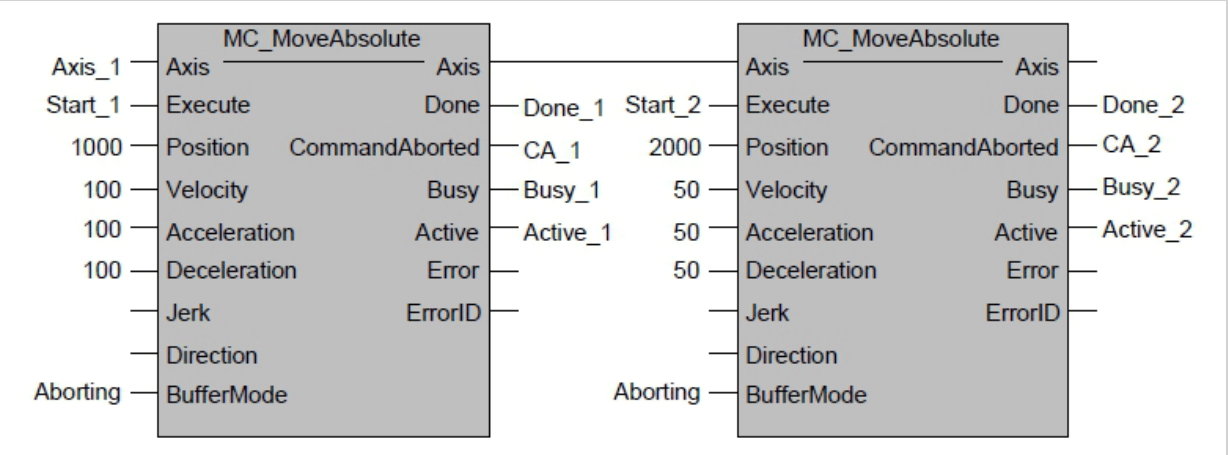


Fig. 61: Timing diagram for example above without interference between function block 1 and function block 2

Example 2: Aborting motion



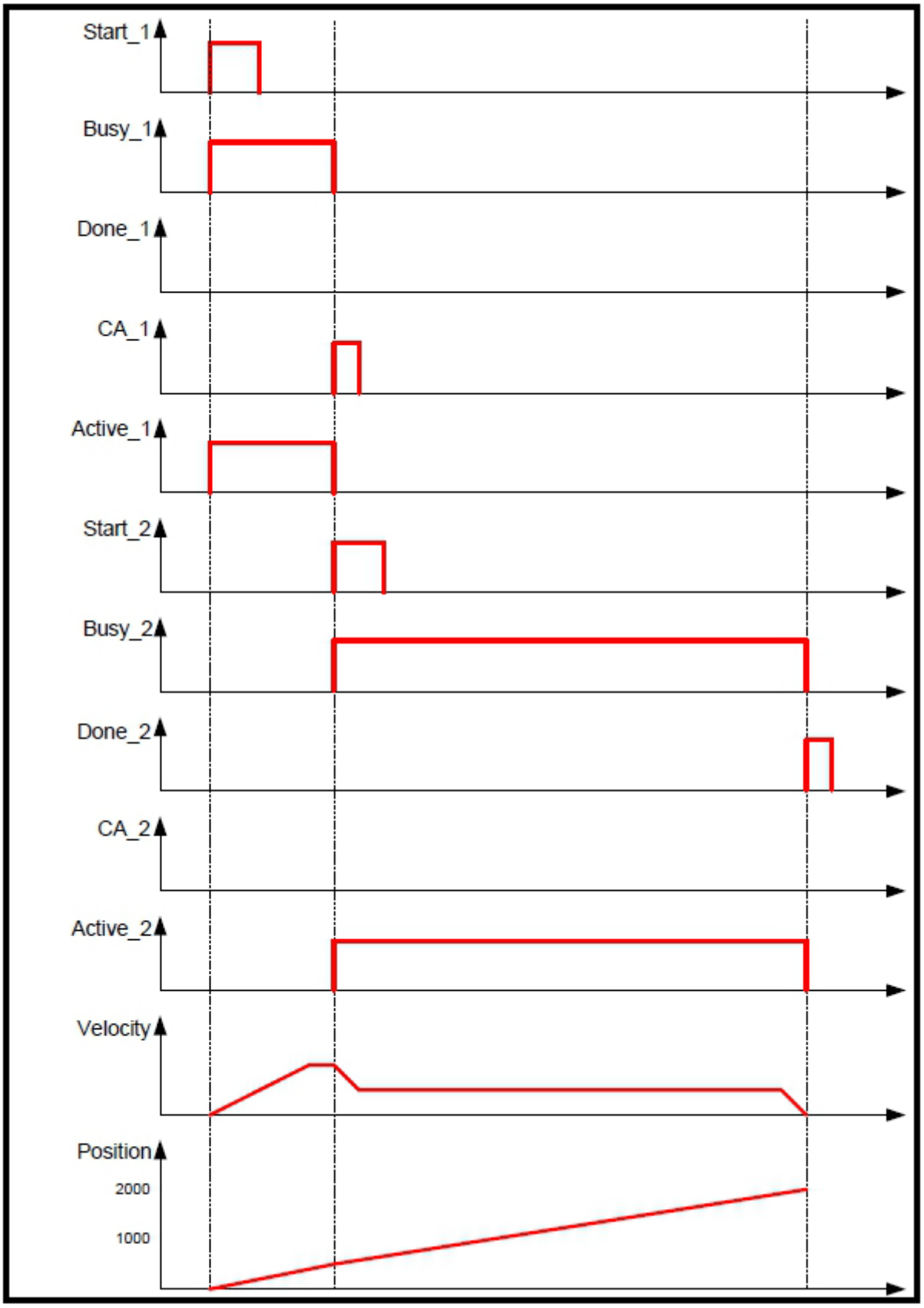


Fig. 62: Timing diagram for example above with function block 2 interrupting function block 1 (McAborting Mode)

If an on-going motion is aborted by another movement, it can occur that the braking distance is not sufficient due to deceleration limits.

In rotary axis, a modulo can be added. A modulo axis could go to the earliest repetition of the absolute position specified, in cases where the axis should not change direction and reverse to attain the target position.

In linear systems, the resulting overshoot can be resolved by reversing, as each position is unique and therefore there is no need to add a modulo to reach the correct position..

1.5.10.3.12 PLCopen examples

Example: A function block instance controls different motions of an axis

The following figure shows an example where the function block (MC_MoveVelocity) is used to control AxisX with three different values of Velocity. In a Sequential Function Chart (SFC) the velocity 10, 20, and 0 is assigned to V. To trigger the input Execute with a rising edge the variable E is stepwise set and reset.

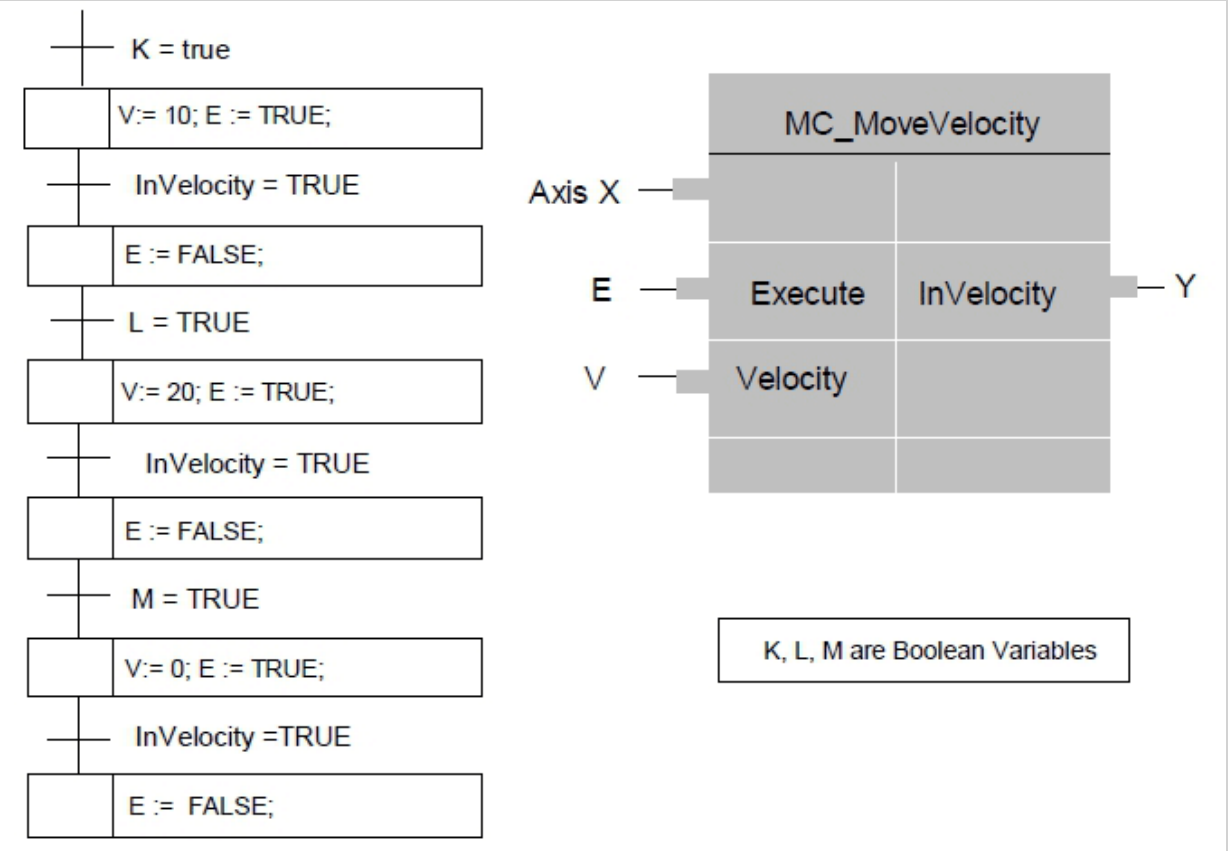


Fig. 63: Single function block with SFC

The following timing diagram explains how it works:

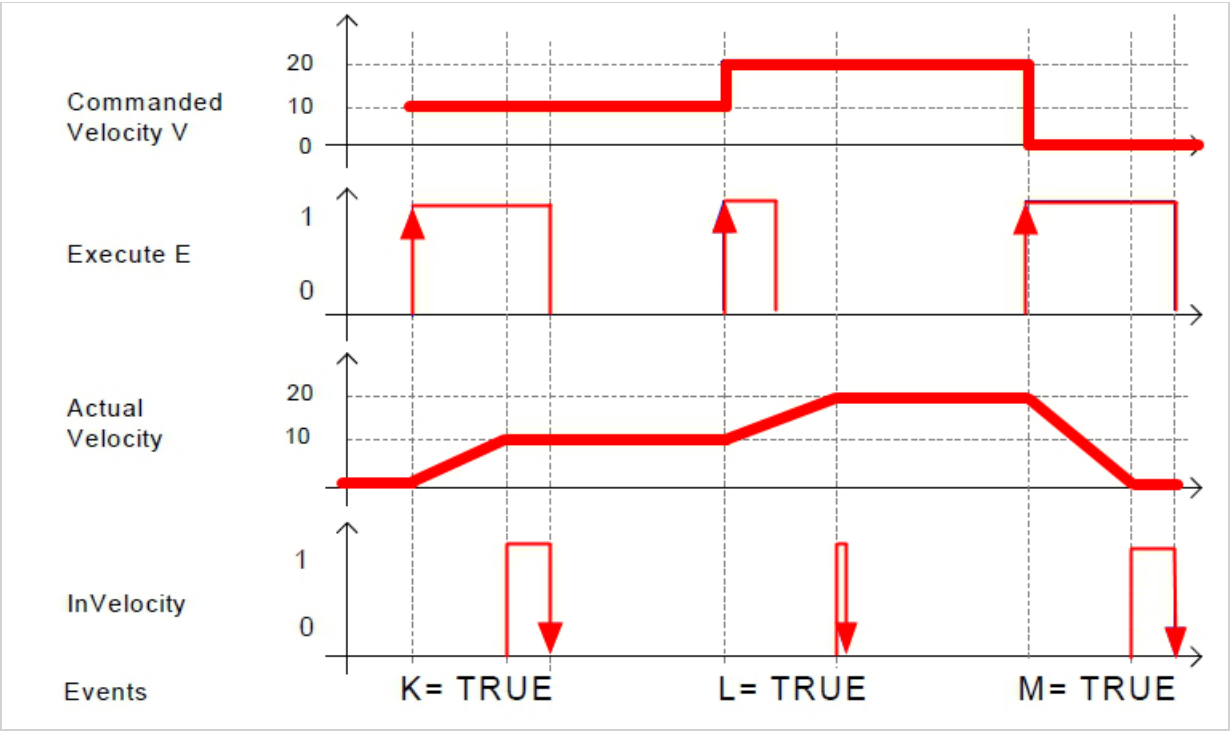


Fig. 64: Timing diagram for a usage of single function block

The second InVelocity is set for only one cycle because the Execute has gone low before the ActualVelocity equals CommandedVelocity.

Example: Different function blocks instances control the motions of an axis

Different instances related to the same axis can control the motions on an axis. Each instance will then be responsible for one part of the global profile.

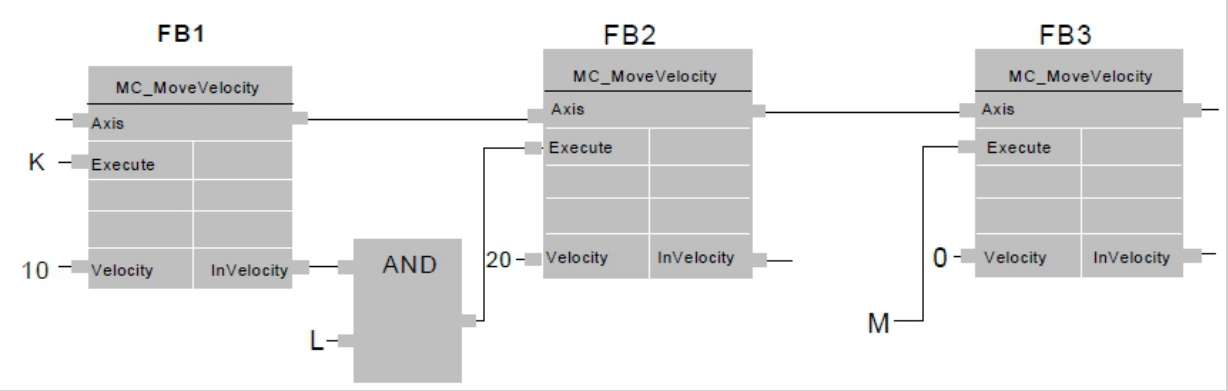


Fig. 65: Cascaded function blocks

The timing diagram:

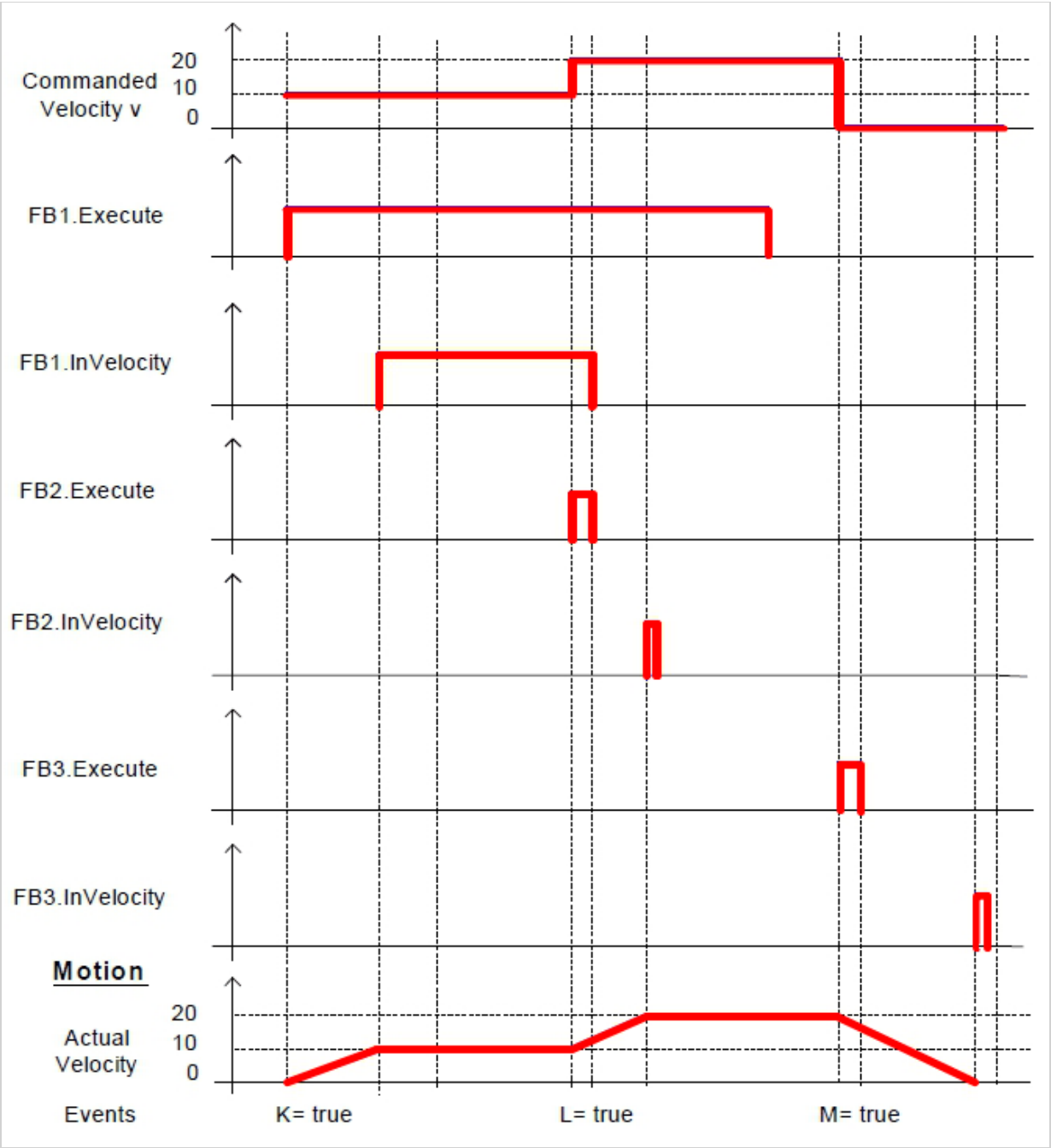


Fig. 66: Cascaded function blocks timing diagram

A corresponding solution written in LD looks like:

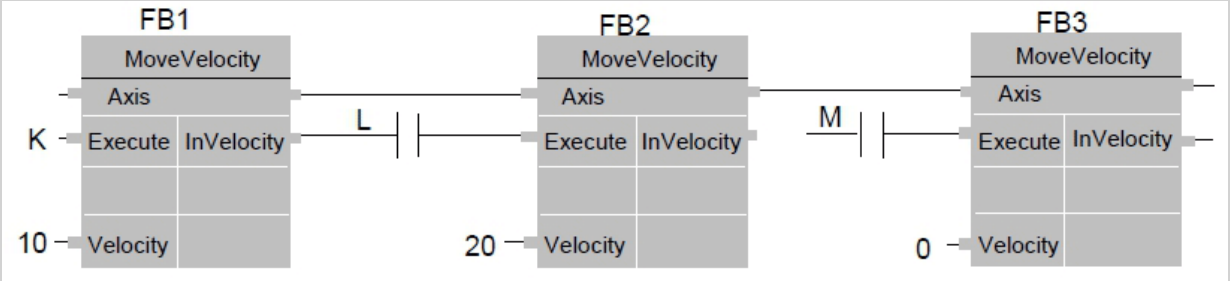


Fig. 67: Cascaded function blocks with LD

1.5.10.4 PLC-based motion control

1.5.10.4.1 PLC-based motion control architecture

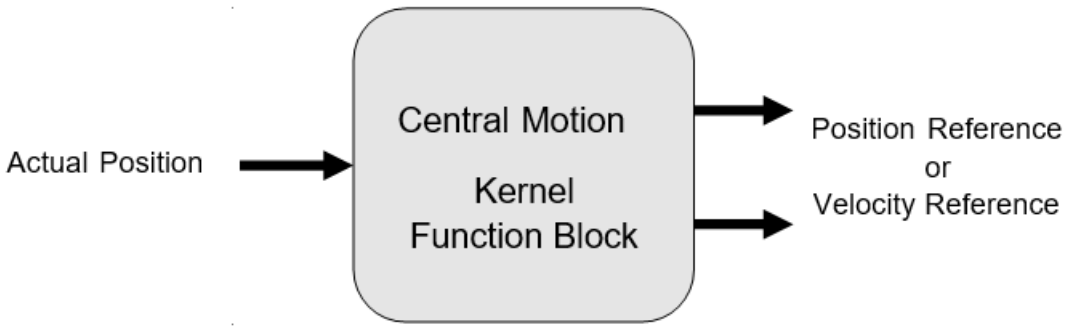
With PS5611-Motion different motion control system structures are possible. Independent of the system structure a typical motion control application consists of the following system elements:

- An application program which contains PLCopen function blocks that defines the general application behavior and logics.
- A profile generator which generates a position profile based on the dynamic specifications of the application program to guide the axis to the desired positions.
- A position control loop which outputs a speed reference signal to minimize the following error.

To achieve the best system structure for an application these components can be separated into different devices. Each type of structure has its own kind of interface and type of signals which need to be transferred between the interacting devices.

> *All shown motion control system structures (Central motion control with or without position control loop) can be combined together in the same application program for a motion control project.*

With the function blocks of motion library a motion control profiler can be used inside the PLC. As shown in the following figure it is needed to provide the actual position of the drive. The output can be either a position or a velocity reference signal. The used output signal will then be used to move the axis in the desired way.



There are 2 possibilities to send a reference value to the drive:

- When the position control loop is closed by the PLC by a CMC_Basic_Kernel function block, the output Speed_Reference should be connected to the drive. The value of Speed_Reference can be scaled with the axis parameters Max_Rpm and Ref_Max.
- When the position control loop is closed by the drive, the output Position_Reference should be connected to the drive. The unit for the output Position_Reference is incremented as well as the input Drive_ActualPosition.

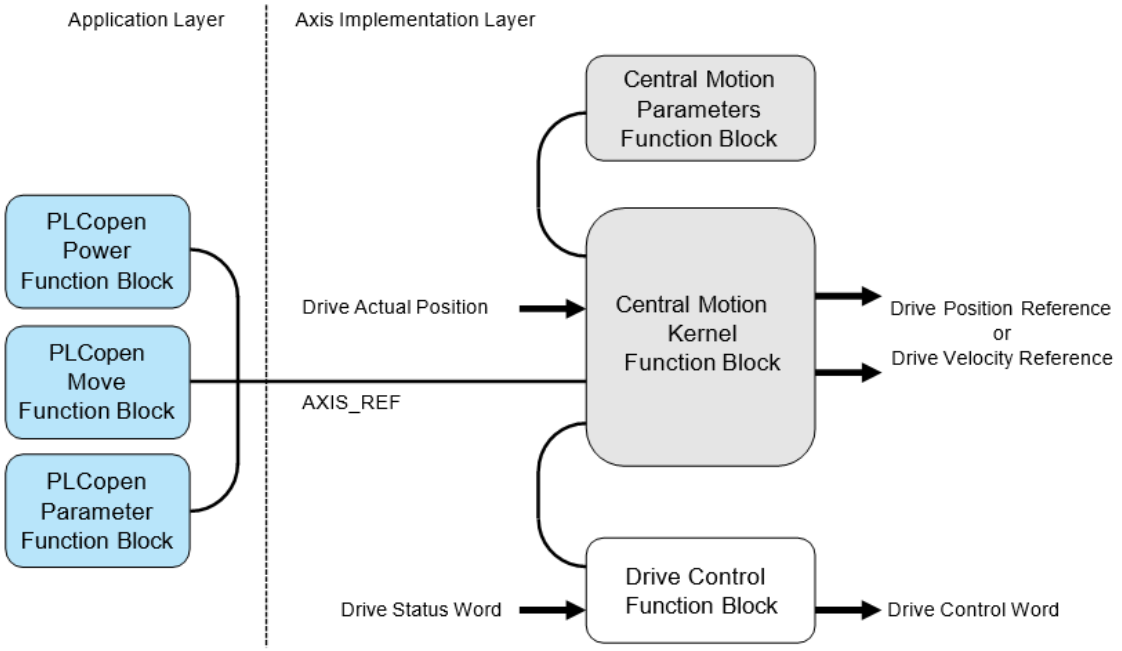


Fig. 68: Architecture for centralized motion control

In general the programming of a machine consists of two layers as shown in the figure above.

In the application layer function blocks according to PLCopen motion control are used to program the application sequences with all necessary types of movements and administrational commands. Due to the standard PLCopen motion control this can be reused in any other machine programs that used PLCopen function blocks.

The axis implementation layer is responsible for the execution of the commands from the application layer and can be programmed for each axis in a different way depending on the used hardware components.

Table 394: Needed function blocks for an application with PLC-based Motion Control

Library	Content
ABB_MotionControl_AC500.library	Kernel function block, Parameters function block, Axis Simulation function block
	Data types for AC500 Motion Control
	Motion Control function blocks according to PLCopen

For a central motion axis implementation the use of the function blocks CMC_Basic_Kernel and CMC_Axis_Control_Parameter are mandatory.

The library design is independent from any bus architecture or any specific drive features.

Example for a possible system architecture

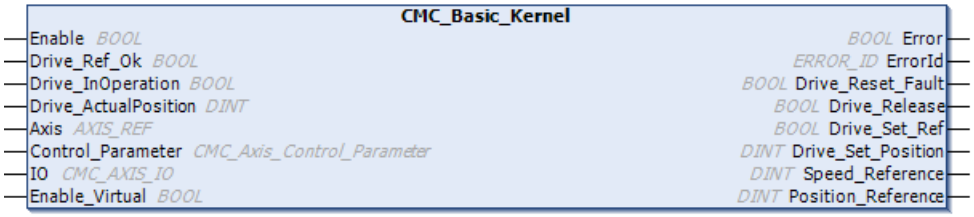
System	Velocity reference	Position feedback
System A	Output via analog output channel as voltage or current	From incremental encoder connected to CD522 I/O module
System B	Output via EtherCAT network	Input via EtherCAT network
System C	Output as frequency signal of CD522 I/O module	From incremental encoder connected to CD522 I/O module
System D	Output via PROFINET IO network	Input via PROFINET IO network
System E	Output via PTO & PWM channel in eCo V3	Input via either encoder (included in onboard IO), or the PTO or PWM pulse count.

In case the velocity reference value is used from the kernel function block the position control loop is closed inside the drive. In this case, it is necessary to adjust the related parameters from the parameters function block. When the position reference will be used the position control loop is closed inside the drive. In this case, the internal control loop is just used to monitor the position and velocity.

When the position reference is used for the drive the following aspects have to be taken care of:

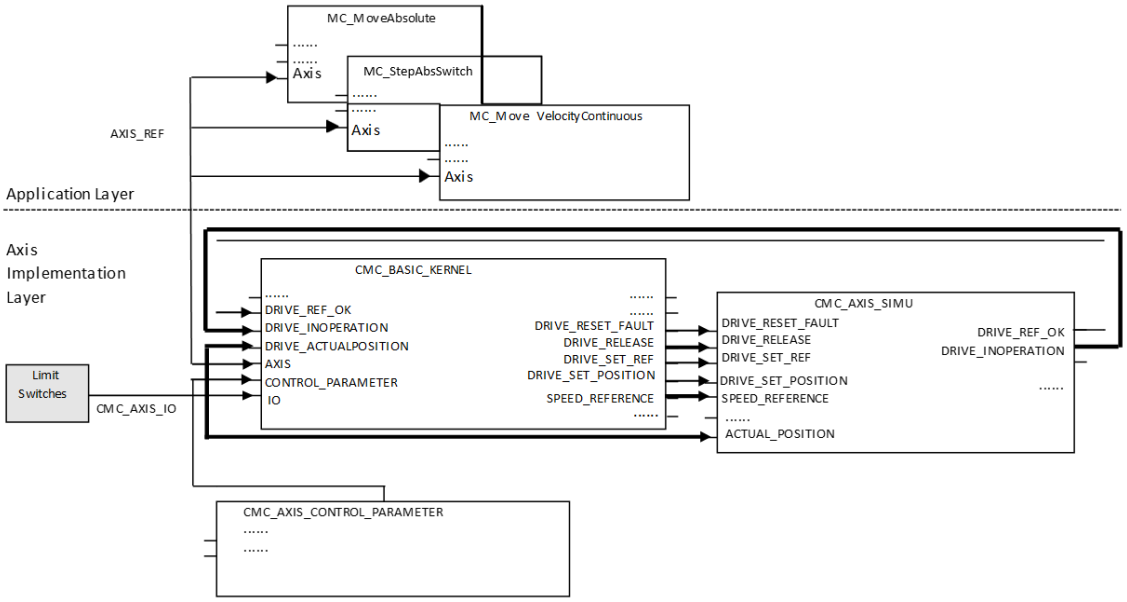
- *It is necessary to use a real time fieldbus, like EtherCAT.*
- *The PLC cycle has to be synchronized to the fieldbus cycle.*
- *The task calculation times may not exceed the used cycle time.*

The drive's status should be managed by a specialized function block that supports the used type of drive as shown in the figure above. The kernel function block is the main function block which is needed to operate an axis with PLC-based Motion Control. It must be used with the parameter function block which is the interface to input parameters which are used to setup the axis.

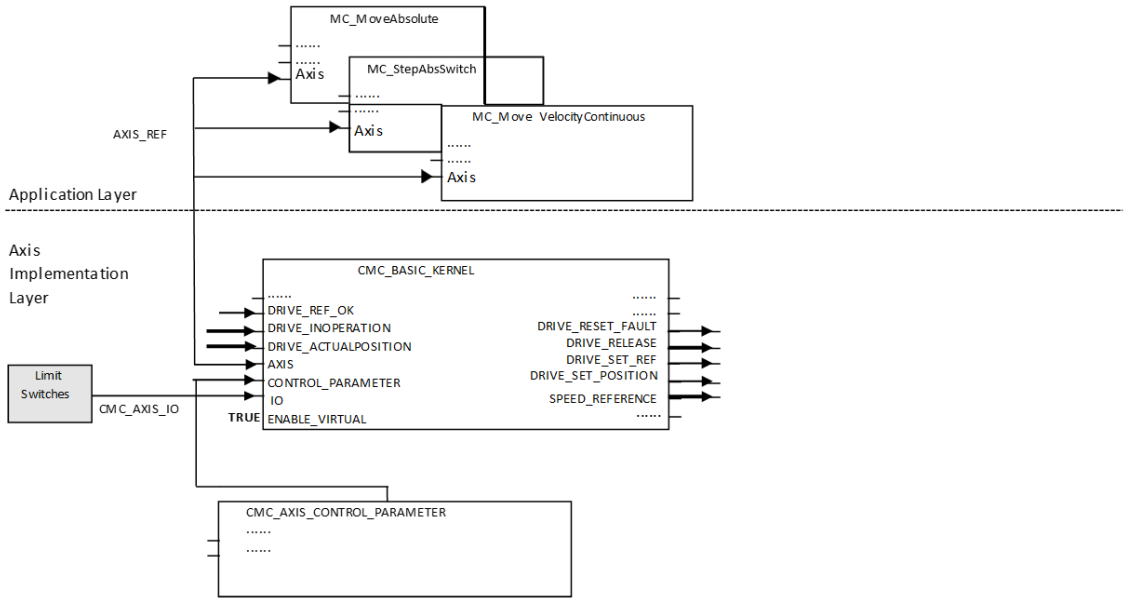


The drive has to be accessed outside the CMC_Basic_Kernel function block. Actual values and reference values might be transferred by a synchronized fieldbus or by I/Os. The function block CMC_Basic_Kernel has to be called every cycle and at least once before any function block MC or MCA is activated.

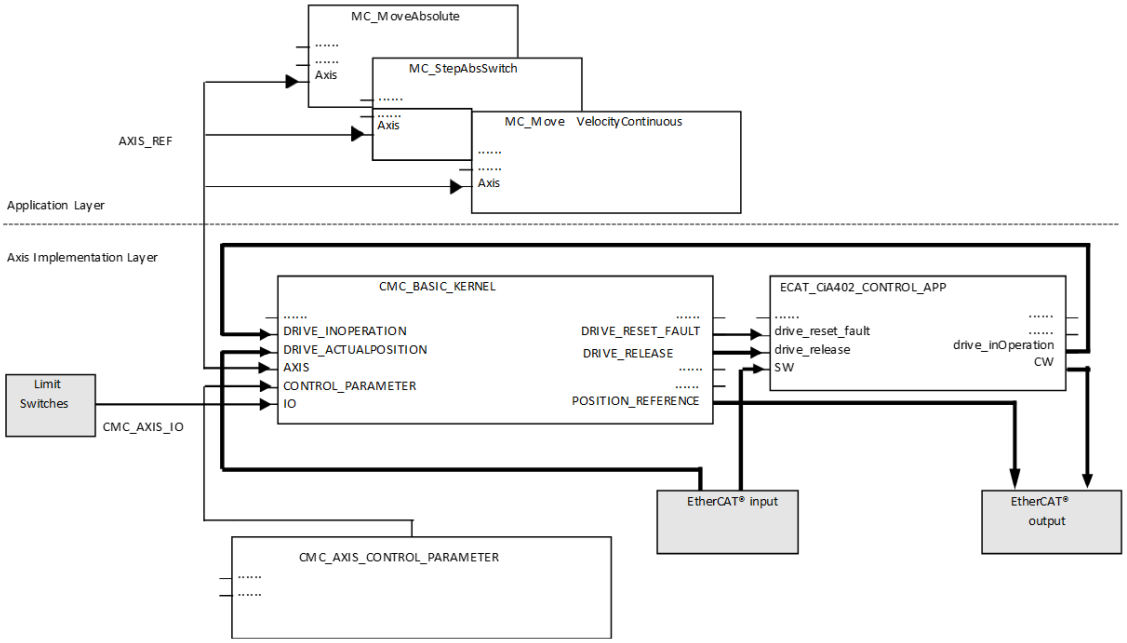
The following figure shows an example with an axis simulation. The main data signals are drawn in bold lines. Here, the drive will receive a speed reference signal which means that the position control loop is closed inside the PLC by the Central Motion function blocks. The time behavior of the simulated drive can be set by the parameter T1 at the axis simulation function block. If the time constant is to slow and the axis parameter Control_Time is too short the simulation axis will run into instability – like a real drive. Sample values: ↳ *Chapter 1.5.10.4.2.3 "How to use the axis simulation" on page 2337*



A different option to create a virtual or simulated axis is to engage the Enable_Virtual input at CMC_Basic_Kernel. This virtual axis will follow the speed reference without additional delay, whereas the CMC_Axis_Simu creates a first order delay.



The following figure shows an example with a CiA402 drive on an EtherCAT network. The main data signals are drawn in bold lines. Here, the drive will receive a position reference signal which means that the position control loop is closed inside the drive.



In the example the main signals are to be transferred via EtherCAT network. The drive control function block for the Microflex e190 can be found in the ABB_Ecat_CiA402_AC500.library.

If using the eCo V3 PLCs, use the OBIO_PTOMotionKernel function block (separate library ABB_MotionControlEco_AC500.library) instead of CMC_Basic_Kernel for the PTO functionality.

In the eCo V3 PLC, if PWM is used in the configuration, use the kernel function block OBIO_PWMMotionKernel function block instead of CMC_Basic_Kernel function block.

Kernel function block

Kernel Arithmetic

The “KERNEL” function blocks are available in two variants.

The OBIO_PTOMotionKernel / OBIO_PWMMotionKernel function blocks are solely to be used in eCo V3 CPUs and to make use of the integrated stepper-IO. It connects automatically to the internal IOs. Use the PTO or PWM specific kernel block based on your configuration ↳ *Chapter 1.6.3.3.1.1.7.2 “Functionality” on page 2451*.

The CMC_Basic_Kernel block is designed to be used in any V3 PLCs and can either work with drives connected to a fieldbus or IOs.

Topic	OBIO_PTOMotion-Kernel/ OBIO_PWMMotion-Kernel	CMC_Basic_Kernel
Recommended PLC	eCo V3 PLC	All V3 PLC`s

How does the parameter for jerk influence the axis movements

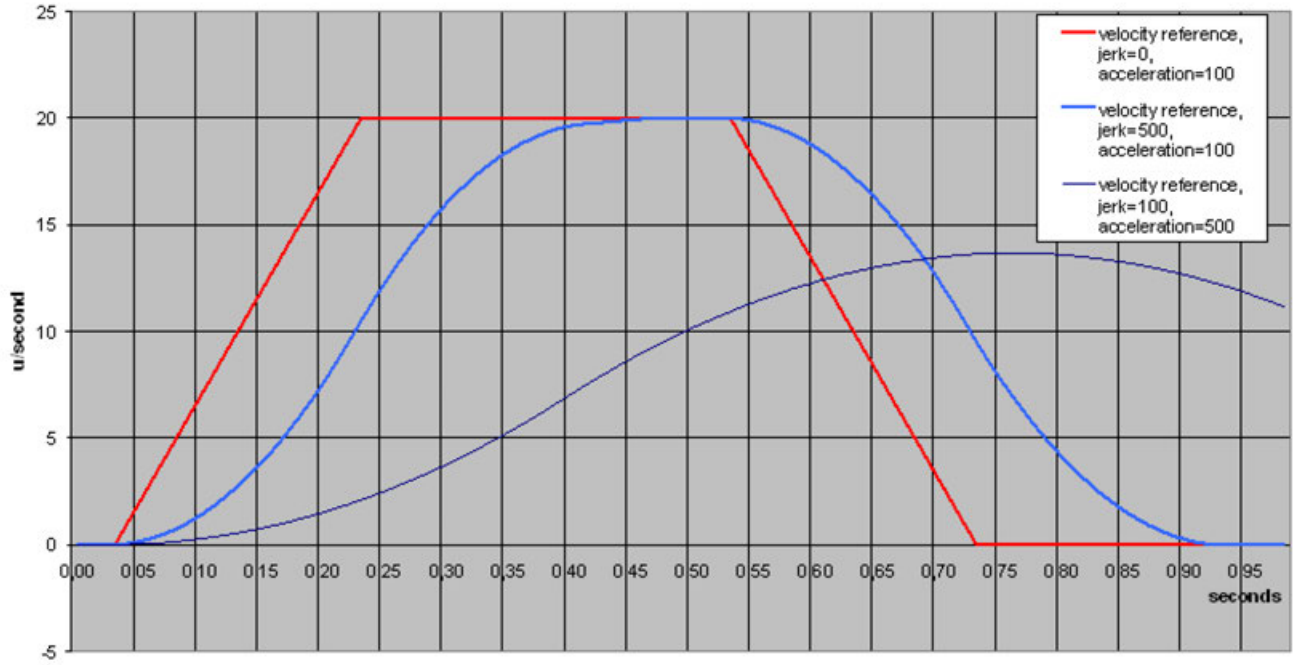


Fig. 69: Velocity reference with different jerk values

The diagram shows the result with different jerk values and the same velocity and acceleration. The time needed for acceleration with jerk=0 is:

Time1=velocity/acceleration=(20/100)s=0.2s

The additional time with jerk=500 will be:

Time2=acceleration/jerk=(500/100)s = 5s

So the total time is:

Time=Time1 + Time2=0.2s + 5s=5.2s

In the last example with jerk=100, the velocity and acceleration values are not reached.

1.5.10.4.2 Basic functionalities

How to connect a drive

The connection to a drive must be done with the inputs and outputs of the function block CMC_Basic_Kernel. All inputs and outputs of the kernel function block with the prefix “Drive_” are intended to be used with a drive, but in some cases not all of them are needed. In all cases the input Drive_ActualPosition has to be connected with the actual position of the axis. This value can be received by an I/O module of the PLC or via a fieldbus.

Depending on which device closes the position control loop either the output Speed_Reference or Position_Reference output has to be used. The value of Speed_Reference can be connected to an analog output module or be transferred via a fieldbus. The value of Position_Reference should be exclusively sent via a real-time fieldbus like EtherCAT.

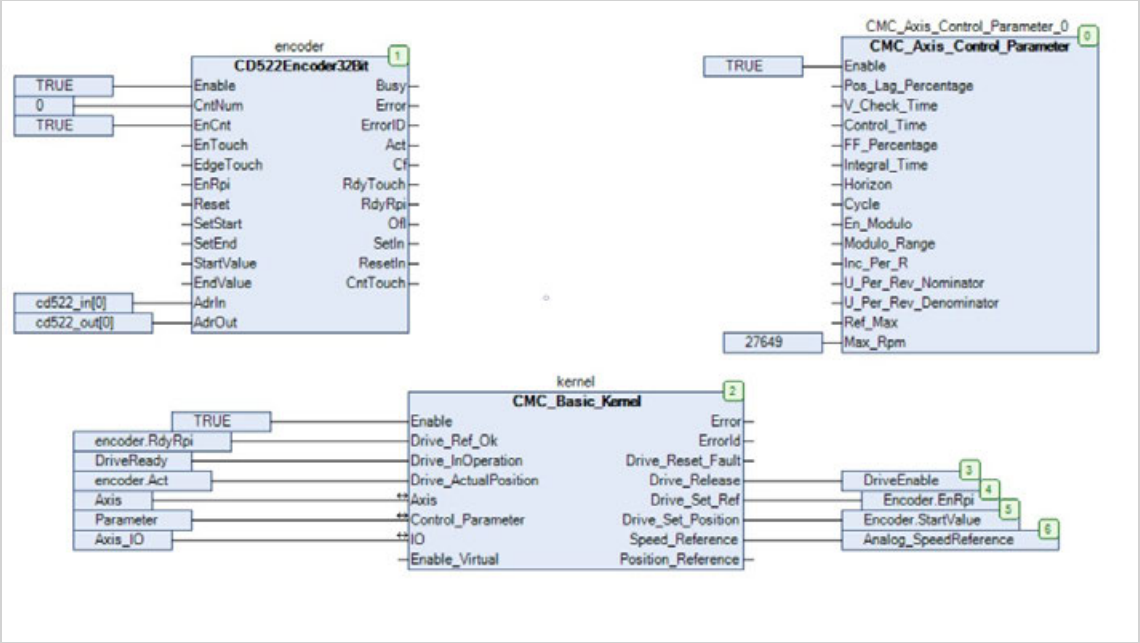
Example 1: Analog drive - Motor with incremental encoder

In the example the position control loop will be closed by the PLC, therefore the input Drive_ActualPosition and the output Speed_Reference are to be used.

In combination with the I/O module CD522 and the corresponding function block CD522Encoder32Bit the position of the encoder can be used. For the effective resolution of the encoder parameter Inc_Per_R of the parameter function block has to be used.

The output Speed_Reference can be written directly to the global variable of an output channel of an analog module but can also be transferred via a fieldbus. The scaling of this output value can be done with the parameters Ref_Max and Max_Rpm of the function block CMC_Axis_Control_Paramter_Real.

The scaling of the Speed_Reference value can be set with the inputs Ref_Max and Max_Rpm of the parameter function block.



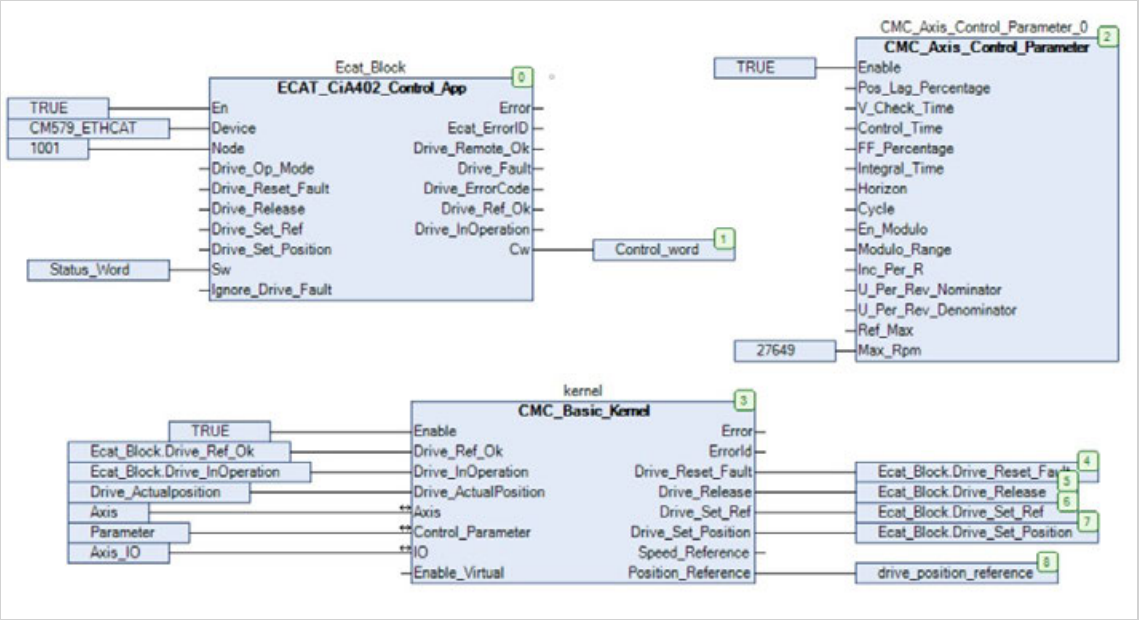
In order to finish a homing sequence which is done by the function block MC_StepRefPulse the outputs Drive_Set_Ref and Drive_Set_Position from the kernel function block have to be connected with the inputs EN_RPI and START_VALUE of the CD552 I/O module function block. Also the output RdyRpi of the CD552 I/O module function block has to be connected with Drive_Ref_Ok from the kernel function block.

To enable and disable the drive Drive_Release could be connected to a binary output to activate the drive. Drive_InOperation could be connected to a binary input to get the information that Drive_Release was successful.

Example 2: Servo Drive - Microflex e190 via EtherCAT in continuous positioning mode (csp)

In the example the position control loop will be closed by the drive, therefore the input Drive_ActualPosition and the output Position_Reference are to be used. The inputs referring to the position control loop of the parameter function block do not have to be set.

For the effective resolution of the motor's encoder parameter Inc_Per_R of the parameter function block has to be adjusted.



To enable and disable the drive Drive_Release and Drive_Inoperation have to be connected to the control function block ECAT_CiA402_Control_App of the library ABB_Ecat_CiA402_AC500.library, which controls the status and control word of the drive.

All function blocks from this library are not password protected and free to be changed in order to be adapted for different drives. The library and the function blocks are marked with the ending _APP.

How to enable and disable a drive

In order to enable a drive the function block MC_Power has to be used within the applicational layer. The kernel function block will then, if possible, output a rising edge on the output Drive_Release which can be connected to the drive-control function block which performs the needed actions on the drives control word to enable the drive. As soon the drive states enabled, this signal can be connected to the input Drive_In_Operation of the kernel function block. The axis state according to the single axis state diagram of PLCopen will then switch from Disabled to Standstill.

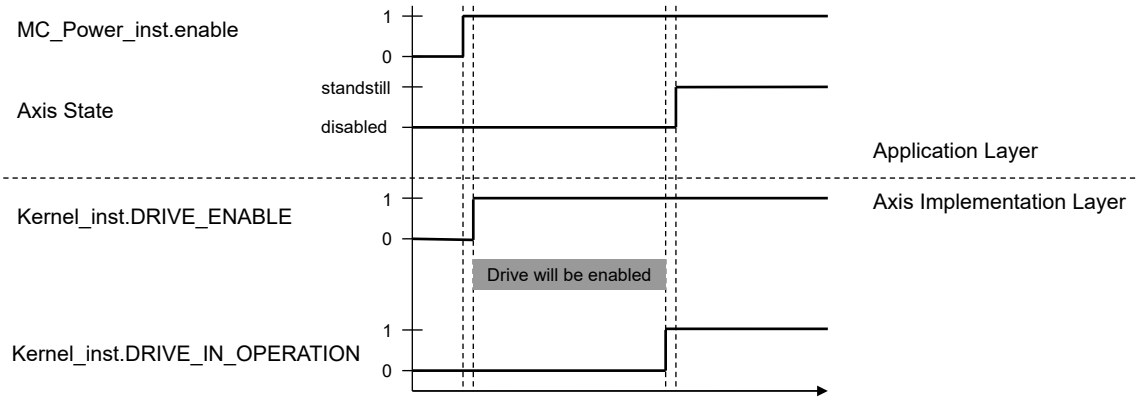


Fig. 70: Enabling sequence of a drive

As long as the drive is in state Disabled or ErrorStop the input Drive_Actual_Position will be copied to the output Position_Reference of the kernel function block. The output Speed_Reference will be zero.

When the axis is in operation, which means it is not in state Disabled or ErrorStop, then the output Position_Reference will be calculated by the kernel function block and the position control loop will be closed, which outputs non zero value for the output Speed_Reference in case of a following error. The input Actual_Position should then follow the position reference. The difference of both values is the following error and will be supervised by the kernel function block.

In case of drive problem, Drive_InOperation should be reset. The function block will open the position control loop and Speed_Reference will be set to zero.

For the most drives the status is control by the drives control word whereas the drives status word represents its actual status. In order to enable the drive it might be necessary to pass through several drives states according a defined scheme which depends on the used drive. Therefore the library ABB_Ecat_CiA402_AC500.library is added to PS5611-Motion package which contains function blocks to operate with different drives on an EtherCAT network. There is also the PS5605-DRIVES library package which can be used to control the state of other ABB drives and other protocols.

How to use the axis simulation

It is possible to use a simulated axis instead of a real drive.

The axis simulation can be used in the following use cases:

- When the real drive is not available the simulation can be used to test all available motion functionalities to verify the application program.
- The simulation can be used to create a virtual master axis and synchronize other axes to it.

The simulation is realized by the function block CMC_Axis_Simu or input Enable_Virtual = TRUE can be used at the KERNEL-block.

Homing will be possible if the limit-switches (data type CMC_Axis_IO) are simulated also. This is not done by CMC_Axis_Simu but could be realized in the PLC program.

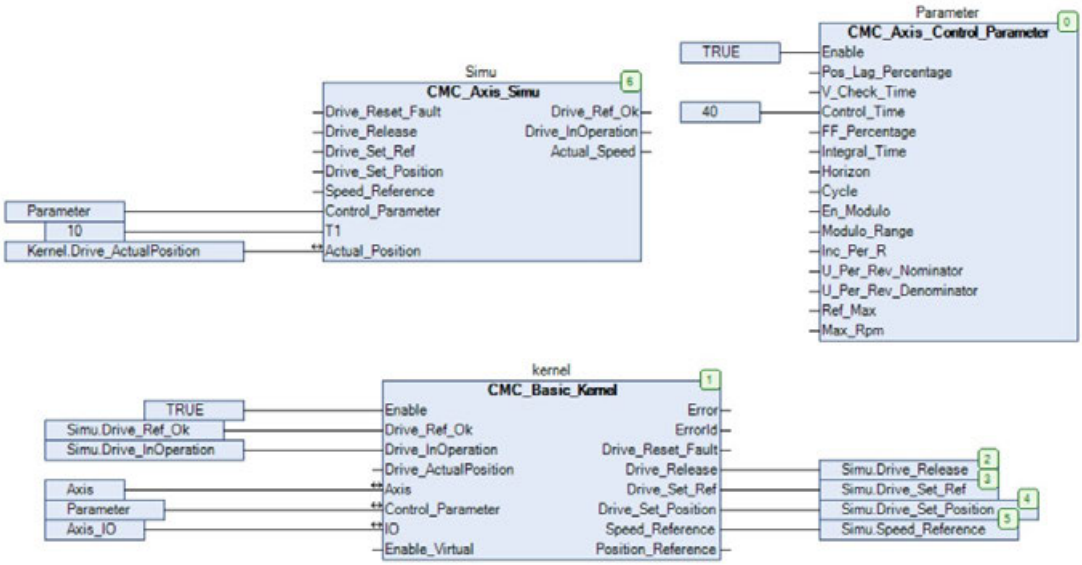


Fig. 71: Example for Simulation

The drive velocity is simulated by PT1-Characteristic. The input T1 gives the time constant for this PT1 as multiple of the cycle time. All other properties are simulated according to the CMC_Axis_Control_Parameter.

The value of the time behavior from the axis simulation function block set by the input T1 has to be at least four times smaller than the value of the axis parameter Control_Time from the parameter function block. If Enable_Virtual = TRUE is used, no delay will be applied to the simulated drive speed, and it will not be possible to test the position-control loop, but it will be fine to be used as virtual axis.

How to perform a homing

The homing of an axis is a procedure which consists of up to two phases. For each phase there are different function blocks available. The available function blocks are according to PLCopen and belong to the application layer.

Table 395: Overview of the available homing function blocks

	Phase 1	Phase 2/Finish Homing
MC_StepAbsSwitch	X	
MC_StepDirect		X
MC_StepLimitSwitch	X	
MC_StepRefPulse		X

In order to create a complete homing sequence one function block of each phase can be used.

First phase

The used function blocks will change the axis state to Homing and will move the axis to approach installed limit switches or a dedicated absolute switch in the desired directions. No manipulation of a position value will be done in this phase. The use of function blocks of this phase is optional for a homing.

The signals of the installed limit switches have to be written to a variable of the data type CMC_Axis_IO.

Second phase

Function blocks from this phase will also change the axis state to Homing if this has not already happen and will finish the homing. Therefore a new position will be set to the axis. The axis state will then switch back to Standstill.

The use of a function block of the second phase is mandatory for a homing.

In general with AC500 PLC-based Motion Control there are two position values: One position value will represent the encoder counts of a drive or the CD522 module which is connected to the input Drive_ActualPosition of the kernel function block. The other position is a user defined scaled unit which is used for PLCopen function blocks.

There are different ways to finish the homing by manipulate and adjust a position value. Which value should be manipulated depends on the used drive or module and its capabilities. See the following types A, B and C.

Type A

The user defined position unit will be changed only. The function block MC_StepDirect must be used here. This type of homing is less complex than the other types but also less precise.

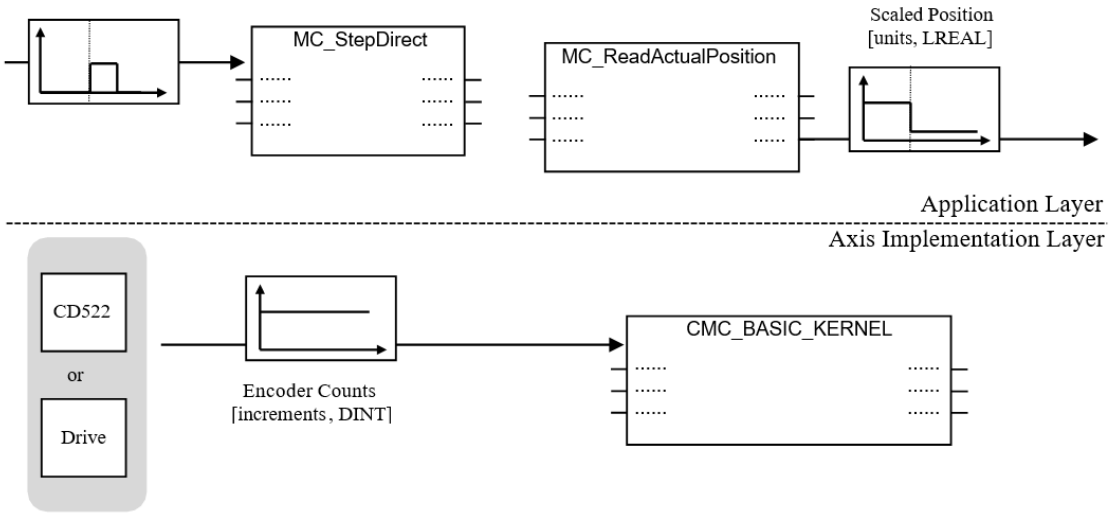


Fig. 72: Homing Type A

Type B

The Drive or the CD522 module will change its own position value, the encoder counts.

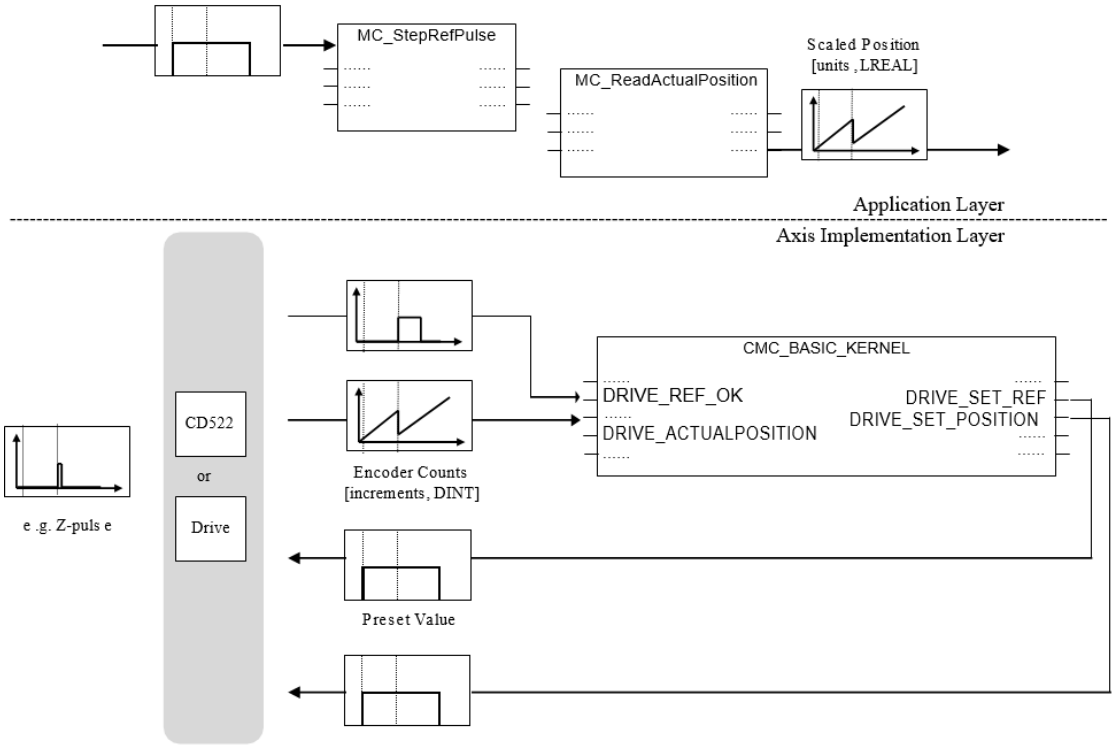


Fig. 73: Homing Type B

The process will be started by the execution of the function block MC_StepRefPulse.

The axis will start to move.

The output Drive_Set_Ref of the kernel function block will then set the drive to sense for a digital signal. At the same time the kernel function block outputs a preset value which will replace the actual encoder count value at the moment the digital signal occurs.

This signal can be a Z-pulse of an incremental encoder but also any other signal from a sensor. This functionality may require a configuration of the drive or the CD522 module in order to be used.

In the same cycle when the new position value is set there also has to be a boolean signal stating a new position value at the input Drive_Ref_Ok of the kernel function block. The user defined position value will then be shifted accordingly.

Example of type B for phase 2: ↳ *Chapter 1.5.10.4.2.1 “How to connect a drive” on page 2334*

Type C

The encoder count position value will not be changed but involves registration capabilities of a drive or the CD522 module.

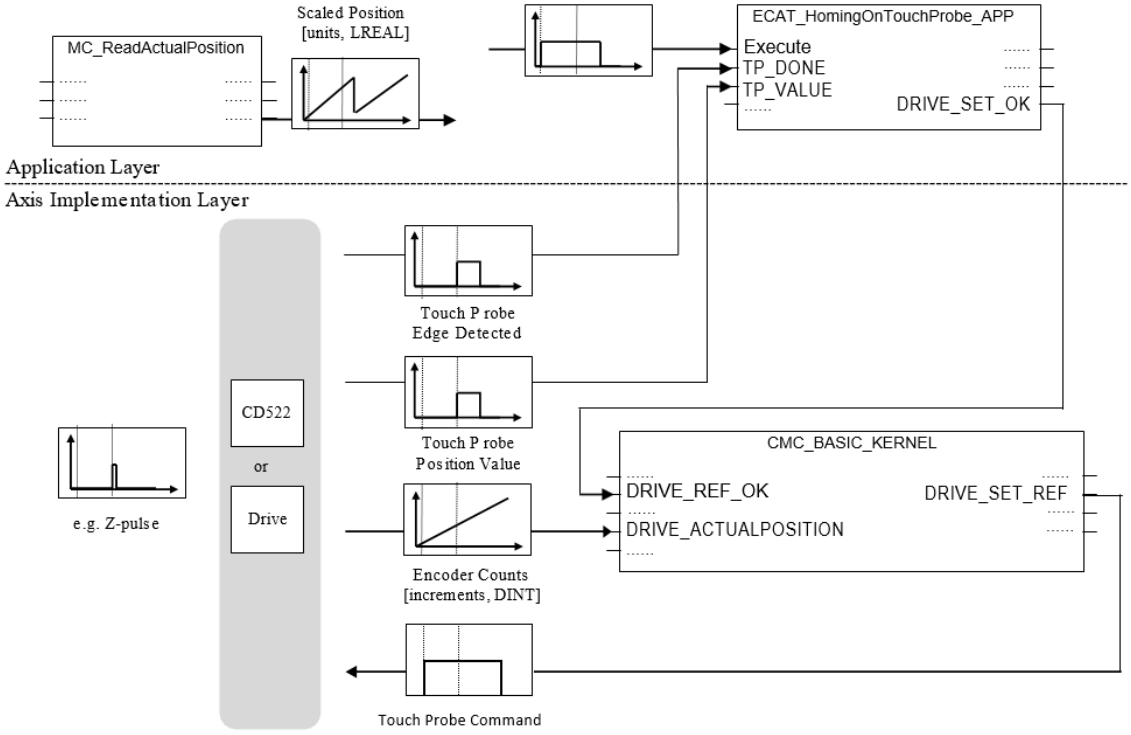


Fig. 74: Homing Type C

The process will be started by the execution of the function block ECAT_HomingOnTouchP-robe_APP (ABB_Ecat_CiA402_AC500.library).

The axis will start to move.

The output Drive_Set_Ref of the kernel function block will then command the drive or the CD522 module to activate the Touch Probe functionality. This will configure the drive to latch a position at the moment a digital signal occurs. The digital signal can be a Z-pulse of an incremental encoder but also any other signal from a sensor. This functionality may require a configuration of the drive or the CD522 module in order to be used.

In combination with the latched position value there is a boolean signal which states that a new latch value has been received. In case of the module CD522 this encoder count position value has to be converted from encoder counts to equivalent user scaled units by the use of the function “CMC_Get_Units_From_Inc” (ABB_MotionControl_AC500.library) before it can be connected to the function block ECAT_HomingOnTouchProbe_APP.

To manage the Touch Probe objects of a drive within the CiA402 profile (e.g. Microflex e190) the function block ECAT_HomingOnTouchProbe_APP (ABB_Ecat_CiA402_AC500.library) can be used. This will also cover the conversion from encoder counts to user scaled units.

At the end of the process the function block ECAT_HomingOnTouchProbe_APP will manipulate the user scaled position value according to the latched position from the drive and the users settings.

For further information see: *AN00220-001 - AC500 and MicroFlex e190 - EtherCAT Homing Methods*

How to Use a CAM curve

The CAM functionality is only available in combination with the kernel function block CMC_Basic_Kernel.

From Automation Builder 2.5.0 onwards user can make use of inbuild Cam Configurator to generate Cam Table. For more details on how to use Cam Configurator please refer Automation Builder Help file.

It is recommended to use the CAM Editor from Automation Builder for those who are new to Cam table or to get the structure of the Cam Table. User can create the complete CAM Table using Cam Editor or can make a copy of CAM Table (IEC Code) and adapt it directly in the IEC code if needed.

Details on the CAM Table structure and different parameters to be considered while creating the CAM is described below.

General usage

The usage of a CAM function is based on the following elements:

- CAM table defined with the data type MC_PProfile.
- An instance of the function block MC_CamTableSelect
- An instance of the function block MCA_Cam_Extra (optional)
- An instance of function block MC_CamIn
- An instance of function block MC_CamOut

The following steps are necessary to use a CAM table

1. Declare a CAM table as an array of the data type MC_PProfile in the program.
2. Write data to this array.
3. Use the address of the CAM table at the input CamTable of the function block MC_CamTableSelect.
4. Execute the function block MC_CamTableSelect to process the data of the CAM table with the function block's input parameters
5. Additionally you can execute the function block MCA_Cam_Extra for optional parameters after the processing of the function block MC_CamTableSelect.
6. Execute the function block MC_CamIn to start the slave axis movement according to the CAM table data and parameters.
   ⇨ The axis will operate in the axis state Synchronized Motion.
7. To leave the axis state you can execute the function block MC_CamOut.
   ⇨ The axis state will switch to state Continuous Motion and maintains its last velocity as long as there is no other command.
8. You can also use any other motion command interrupt the Synchronized Motion.

CAM table

CAM data is done with one table (two dimensional – describing master and slave positions together).

The data of the elements (array of data type MC_PProfile) can either be assigned within the declaration or can be assigned during run time before the execution of the function block MC_CamTableSelect.

It can be filled with data in the following ways:

- To use a predefined variable list.
- To calculate the values within the program (before using the MC_CamTableSelect).
- To send values by any communication access to the PLC.

In order to use the new data it is necessary to execute the function block MC_CamTableSelect again. In case the CAM table is executed the function block MC_CamTableSelect may not be executed.

Elements of the data type MC_PProfile: ↳ *Chapter 1.5.10.2.4 “Overview of data types” on page 2297*

The inputs MasterSyncPosition and MasterSyncDistance of the function block MC_CamIn can be used to define a distance to synchronize the slave axis onto the CAM table during the start. In case master axis moves with negative velocity the parameter MasterSyncDistance can be negative. The MasterSyncPosition should always be within the range of the CAM table master position.

MasterSyncDistance = 0 will deactivate the synchronization. In this case the slave axis should be moved on the CAM curve before MC_CamIn is executed, otherwise a following error can occure.

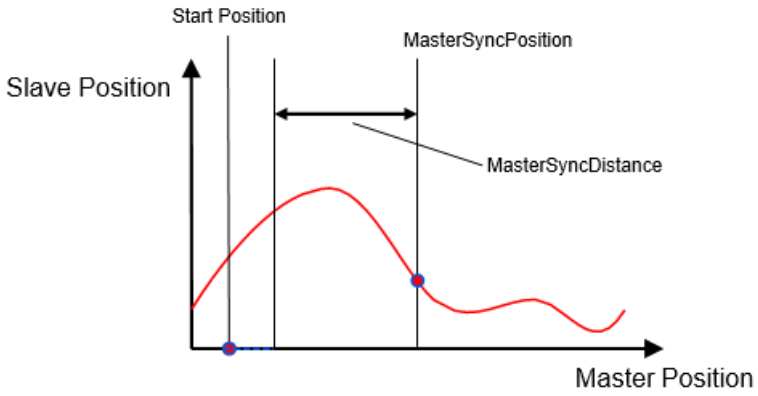


Fig. 75: CAM profile figure

The master position in the CAM table must be strictly monotonic rising.

The length of a CAM table is just restricted by the memory size of the PLC. When long tables are used, it is recommended to call CamTableSelect in a task with lower priority as it will need a considerable computing time.

It is possible to hold several CamTables as a pool and to switch from one to another. This has to be done at matching positions as no means for synchronization are available.

The offset and scaling values (except the time-scale) are transferred continuously. This will allow to follow a "Moving Target" by adjusting these values.

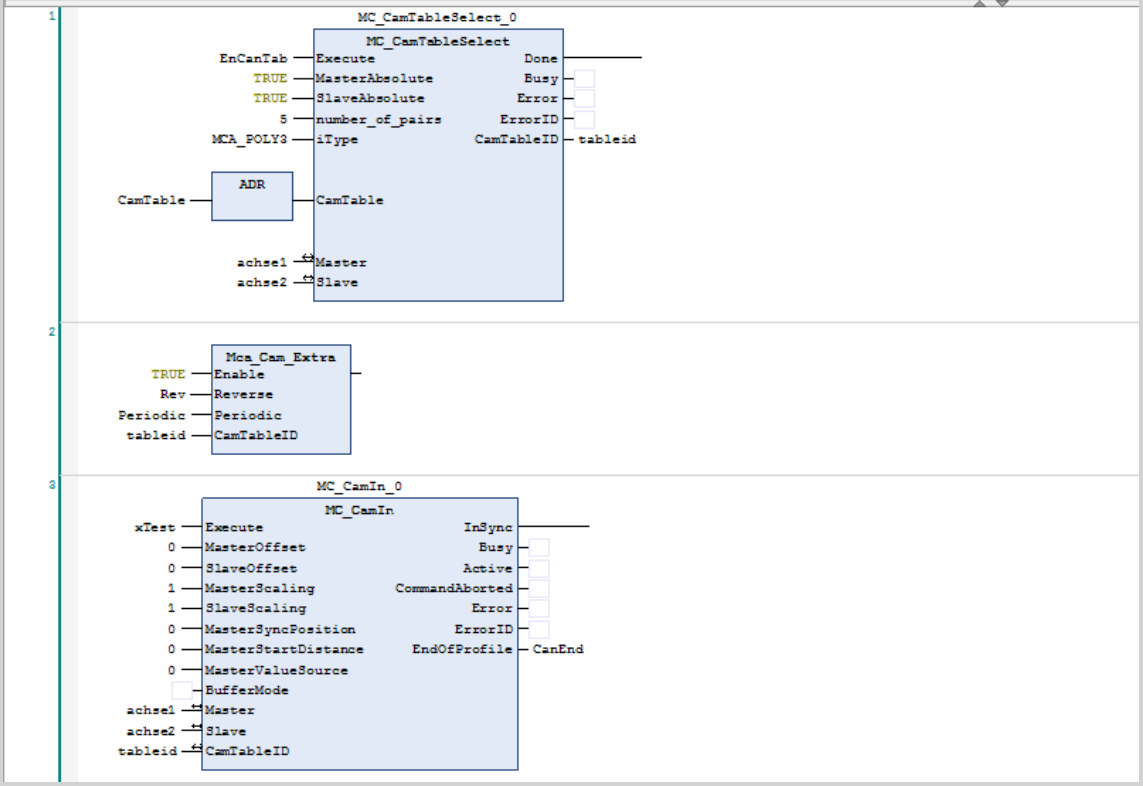
The parameters at MC_CamTableSelect, MC_CamIn and function and MCA_Cam_Extra also modify the behavior:

Parameter MC_CamTableSelect	Type	Default value	Comment
MasterAbsolute	BOOL	FALSE	TRUE=Master_position from MC_PProfile equals the master axis absolute position. FALSE=CAM is executed relative to the master axis actual position at start.
SlaveAbsolute	BOOL	FALSE	TRUE=interpolation_point from MC_PProfile equals the slave axis absolute position. FALSE=CAM is started from actual slave position. The values "interpolation_point" are relative to the slave axis position at start.
iType	MC_ABB_iTypes_ENUM		Interpolationtype.
Number_of_pairs	INT		Number of points used in TimePosition Array.

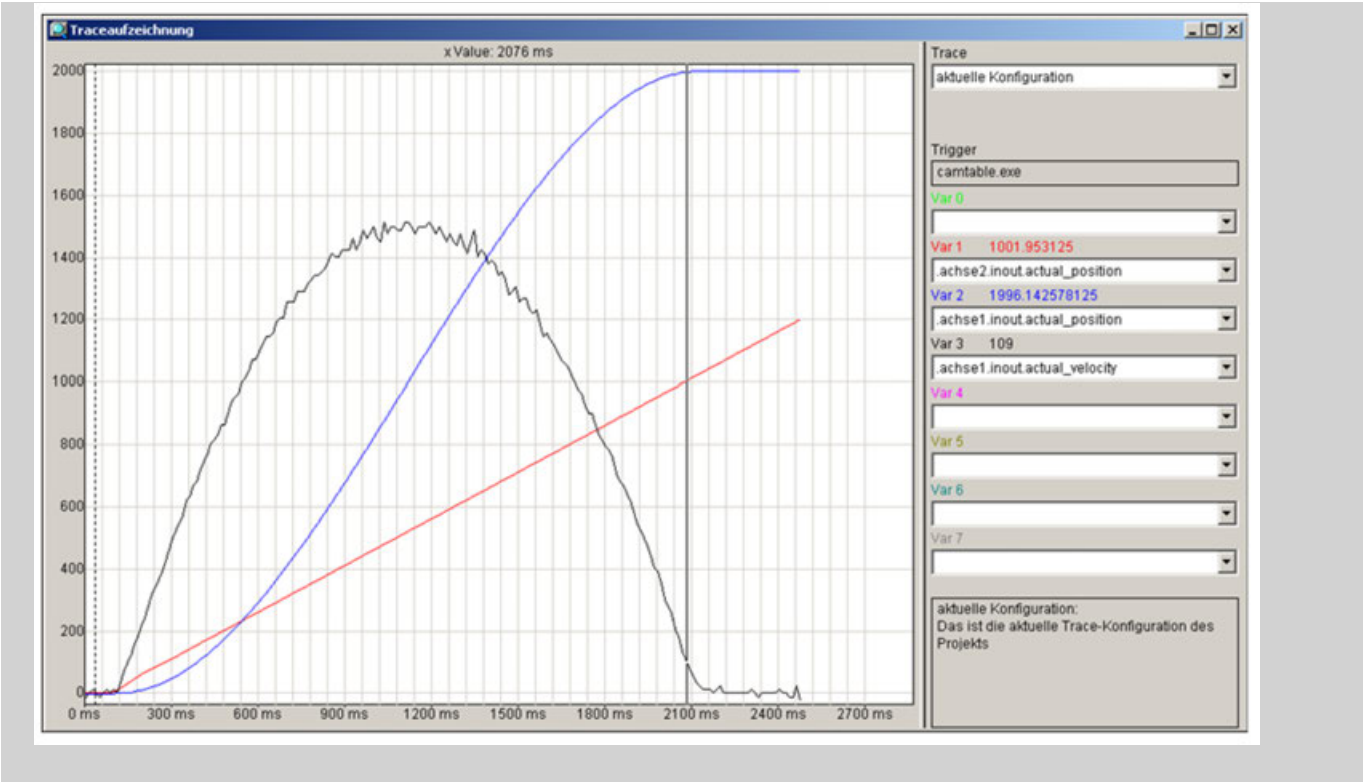
Parameter MC_CamIn	Type	Default value	Comment
MasterOffset	LREAL	0	Just used with MasterAbsolute=TRUE, ignored otherwise. Used position for cam-table is: Master axis position-Masteroffset.
SlaveOffset	LREAL	0	Just used with SlaveAbsolute=TRUE, ignored otherwise. Used position is slave axis position=interpolation_point+Slaveoffset.
MasterScaling	LREAL	1	The position used for interpolation is multiplied by MasterScaling, e.g MasterScaling=2, the scaled master will pass the position range with double velocity and within the half distance compared to its real velocity and position.
SlaveScaling	LREAL	1	Interpolation result is multiplied by Slave-Scaling, e.g SlaveScaling=2: Slave axis will run twice the distance.
MasterSyncPosition	LREAL	0	Start synchronization at master axis position=MasterSyncPosition-Master-StartDistance+MasterOffset, meet the CamTable at master axis position=MasterSyncPosition. In case of MasterAbsolute=FALSE: start at "actualPosition+MasterSyncPosition-MasterStartDistance", meet the CamTable at "actualPosition+MasterSyncPosition"!!! It is just possible to use the "sync" mechanism when the axis is in StandStill on start.
MasterStartDistance	LREAL	0	A negative value will create a reverse synchronization mode, which means the master should move in negative direction to synchronize. It is independent from the Reverse-Bit which indicates how to end the movement.
These 2 parameters are "extras" to be written with the MCA_Cam_Extra function. When the parameters are used, the MCA_Cam_Extra has to be called after the MC_CamTableSelect.			
Periodic	BOOL	TRUE for master “Modulo”, FALSE for master linear axis	CamTable will not reach "EndOfProfile" but will be repeated periodically. When the master is a linear axis, it has to move forward and backward within the CamTable position range, but even when it leaves this position range, the CamTable will stay active.
Reverse	BOOL	FALSE	Just necessary when a CamTable is NOT "periodic" and will run in reverse direction (master with negative velocity) Reverse=FALSE, the CamTable is ready when the master leaves the position range in positive direction, e.g. when it moves from 359° to 0° on a rollover axes Reverse=TRUE, the CamTable is ready when the master leaves the position range in negative direction.

Example for CAM curve

```
PROGRAM CamIn
VAR

    CamTable:ARRAY[1..5]OF MC_PProfile := [
    (Master_position:= 0   ,interpolation_point := 0   , Velocity_ratio:= 0 , Acceleration_ratio:= 0 ),
    (Master_position:= 150 ,interpolation_point := 100, Velocity_ratio:= 0 , Acceleration_ratio:= 0 ),
    (Master_position:= 300 ,interpolation_point := 300, Velocity_ratio:= 0 , Acceleration_ratio:= 0 ),
    (Master_position:= 600 ,interpolation_point := 350, Velocity_ratio:= 0 , Acceleration_ratio:= 0 ),
    (Master_position:= 800 ,interpolation_point := 400, Velocity_ratio:= 0 , Acceleration_ratio:= 0 )];

    MC_CamIn_0: MC_CamIn;
```



In the example, the slave will run from 0 to 2000 while the master runs from 0 to 1000. The slave will start and end with velocity=0, no matter which velocity the master has during start. The slave will reach the maximum velocity when it is at position 1000 and the master is at position 500.



How to use an external axis

To use multiaxis PLCopen function blocks with an externally sensed axis as master axis the following structure can be used for the axis implementation:

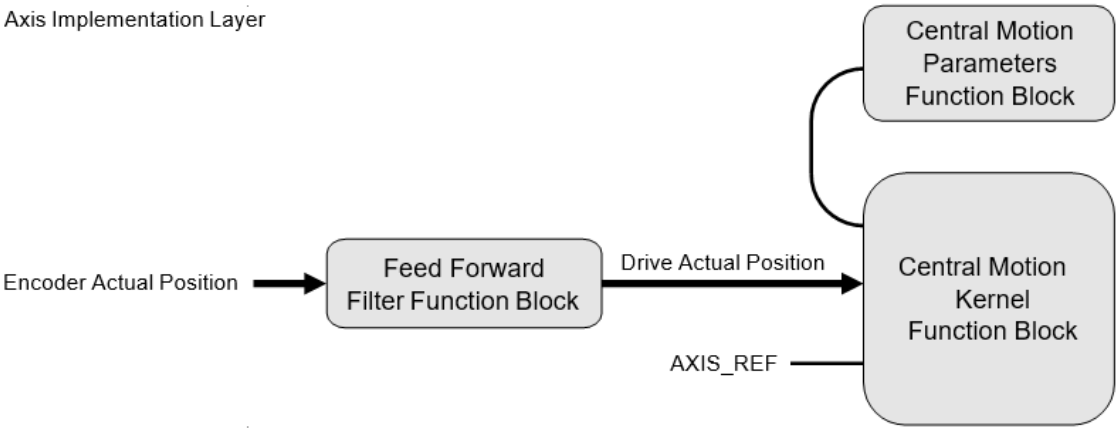


Fig. 76: Structure synchronization to an external axis

The use of a feed forward filter function block is needed if the slave axis has to follow the position of the external axis. In this case there will be a time delay between sensing the position of the external axis and moving the follower axis along the sensed position. The filter function block will then add a certain distance to the external axis' position depending of its speed.

The filter function block MATH_LINEAR_REGRESSION from the library ABB_MathFunctions_AC500.library can be used here.

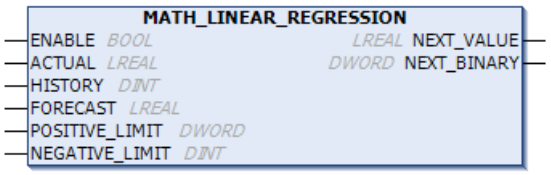


Fig. 77: Filter function block to feed forward an externally sensed position

For an axis which is following the external axis, the value “mcActualValue” (from MC_Source enumeration) for the input “MasterValueSource” for multi-axis PLCopen function blocks has to be used.

When the filter function block MATH_LINEAR_REGRESSION is used to process an actual position, 2 different purposes are fulfilled:

- A jitter or noise can be compensated
- It is possible to calculate a forecast-position to compensate for a delay in position measurement

> *Process the actual position or any other master axis always before the slave axis.*
>
> *Otherwise, an additional one cycle-delay is introduced.*

The MATH_LINEAR_REGRESSION function block calculates the progress for a variable which is captured in equidistant periods of time and is assumed to follow a linear curve. It uses the Gauss “least squares” -algorithm to do so. The line is calculated in a way that the sum of squares for the distances from the measured points to the assumed straight line is minimized.

A noise or jitter influence of the value is compensated and a predictive value for the variable with an adjustable forecast horizon can be calculated.

Linear equation:

$$\text{Line[i]} = \text{gradient} * \text{i} + \text{offset}$$

Sum of squares:

$$sum = \sum_{i=1}^{\infty history} (\text{x[i]} - \text{line[i]})^2$$

The gradient and offset for the line are calculated in a way that “sum” is minimized. Then these 2 values are used to calculate the forecast value:

$$\text{NEXT\_VALUE} := \text{gradient} * \text{FORECAST} + \text{offset}$$

FORECAST=0 would mean: value right now, no future or past considered.

When the ACTUAL value is a modulo value, for example a single turn encoder or a rollover axis, this has to be considered in the calculation. The 2 input values POSITIVE_LIMIT and NEGATIVE_LIMIT can be used to configure this. They define the upper and lower limit for ACTUAL. Also, the NEXT_BINARY will as a result be limited to these borders.

Example

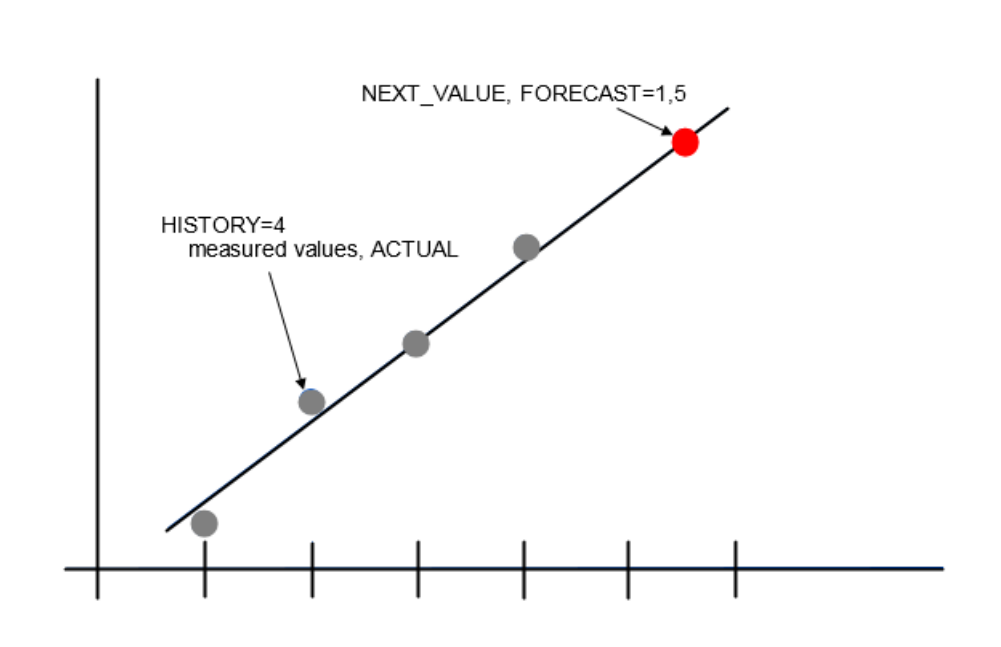


Fig. 78: Next Value_Forecast

How to use an encoder/drive with <> 32-bit position overrun

The incremental position as actual position at the function block CMC_Basic_Kernel is usually assumed as position with a 32-bit position overrun. As well as it is the reference position which is sent to the drive.

Any modulo-axis configuration should be done inside the PLC.

Some drives are requested to correct their positions themselves for a non-linear axis which should constantly run into the same direction.

In this case, the drive has to be configured as a modulo-axis and the function block CMC_Basic_Kernel needs some additional function blocks to create the 32-bit value ↳ *Chapter 1.5.10.4.3.4 “Roll-Over axis” on page 2357*.

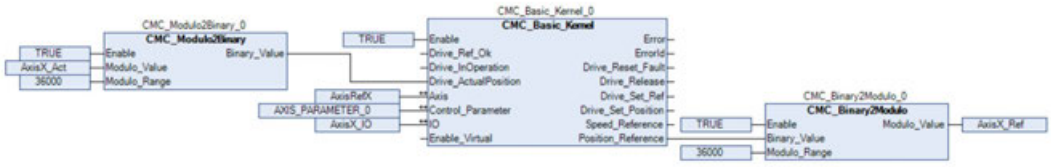


Fig. 79: Kernel

The function block CMC_Modulo2Binary will convert any position with any Modulo_Range to a 32-bit binary position.

The actual_position is assumed to run between 0 to Modulo_Range.

The actual_position should not change > 1/4 Modulo_Range between two scan cycles.

The function block CMC_Modulo2Binary will convert the 32-bit binary position reference from CMC_Basic_Kernel to a position reference which runs from 0 to Modulo_Range.

How to do position correction “on the fly”

Sometimes it is required to have a position correction "on the fly". For example, it can happen that a position is wrong due to mechanical slip and that a switch which is passed by during the movement is used to capture a position value.

In other cases, it is required to synchronize the position to a print mark, so an actual_position has to be corrected, but not the movement of the printed material.

For both applications, the function block MCA_SetPositionContinuous can be used. It will use ramps and a limited velocity for the correction, so it will be tolerable to execute it during an ongoing movement and while the axis is activated in a multi-axis movement.

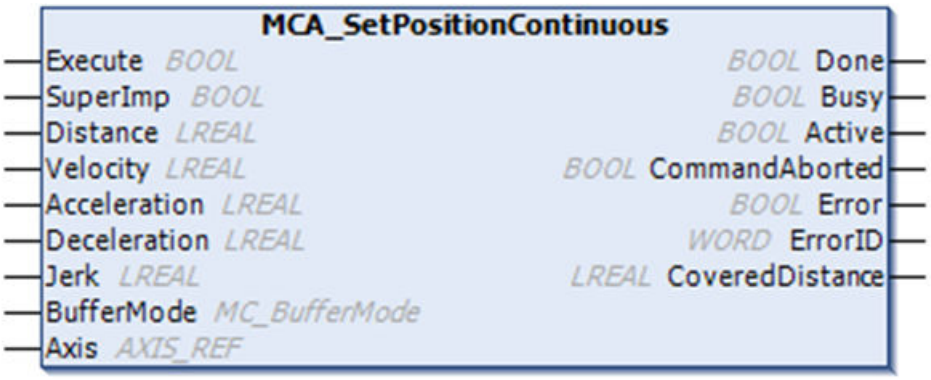


Fig. 80: MCA_Set_PositionContinuous_V3

The block can be used in any axis state except ERRORSTOP and HOMING.

Two different operation modes are possible:

1. SuperImp=FALSE
   - The actual_position will be modified.
   - The block will not cause any movement.
   - If a PLCopen block in DISCRETE_MOTION (positioning) is active during the execution, this block will not reach Done as the actual_position is modified.
   - If a slave axis is coupled to an axis while MCA_SetPositionContinuous is executed (with SuperImp=FALSE) it will follow.
   - This mode is possible while the axis is in state DISABLED.
2. SuperImp=TRUE
   - The actual_position will stay constant.
   - A mechanical movement is executed (without changing the axis state machine).
   - A slave axis will not follow.
   - This behavior is similar to a superimposed movement.
   - It is not possible when the axis is in state DISABLED.

The block can just be aborted by another MCA_SetPositionContinuous.

How to limit the movement

It is possible to limit the movement by position (software limit switches) and by velocity. By default, no software limit switches are activated in PS5611-Motion. It is possible to activate them by accessing some PLCopen parameter.

The functionality described below is just available with linear axes.

	Parameter	Data type	Minimum	Maximum	Default	R/W	Description
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive software limit switch position.
3	SWLimitNeg	DINT	-2147483647	2147483647	2147483647	R/W	Negative software limit switch position.
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch.
5	EnableLimitNeg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch.
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement FALSE = Limits position and velocity, decelerates and shows a warning until the position limit is reached, then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP
2005	EnableLimtVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGap	LREAL	0	21474836 4700	0	R/W	Used to stop the ongoing movement if position is behind

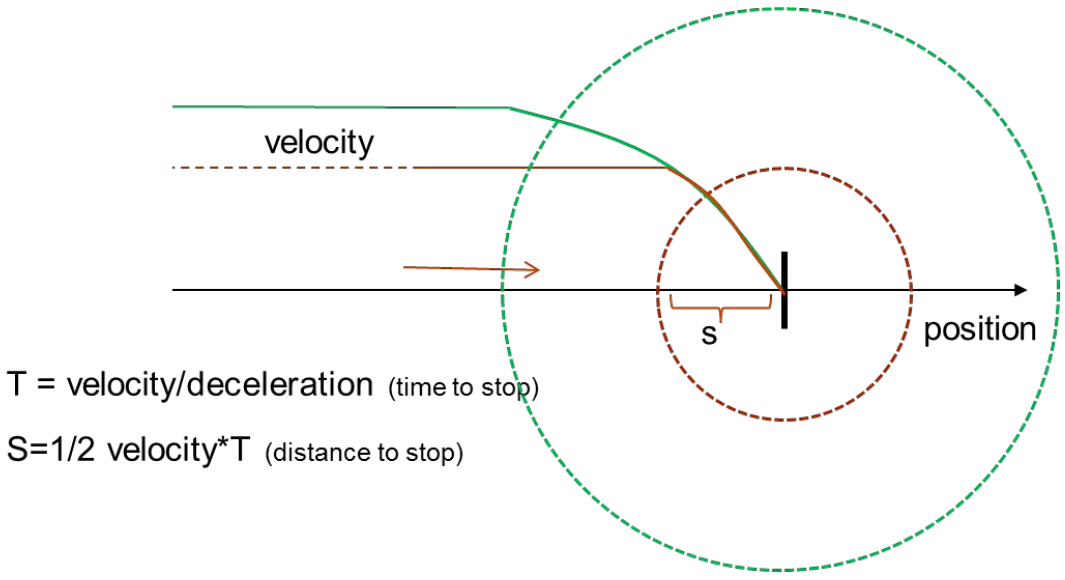
The following different behavior is possible:

- No limitation at all (default)
- Limit position with ERRORSTOP:
  - Limit position between SWLimitNeg to SWLimitPos, axis to state ERRORSTOP in case the position range is left.
- Limit velocity and acceleration:
  - Limit velocity to paraMaxVelocityAppl and acceleration/deceleration to paraMaxDecelerationAppl, create WARNING_VELOCITY, not state changes for axis, abort movement is optional when MaxPositionGap is reached due to limitation.
- Limit Position with ramp-down:
  - In addition, it is possible to limit the position between SWLimit2DecNeg and SWLimit2DecPos. paraMaxDecelerationAppl is used to ramp down.

When activated with EnableLimitPos or EnableLimitNeg, the reaction will be as follows:

- When the control position reaches the respective limit switch, the axis will go to state ERRORSTOP, and Drive_Release will be switched off. The actual_position might be behind, depending on the following error. It is assumed that a drive or application specific braking is performed. The axis will be stopped behind the limit.
- The axis could be switched on again by MC_Power. A movement in the opposite direction will be possible.
- The functionality of EnableLimitPos and EnableLimitNegis unchanged.

You can use the limitation of movement to achieve a soft or adjustable braking in advance before reaching the software limit switch. The limitation is activated by three Boolean parameter and will calculate a position distance to the limit switch, which depends on the actual velocity and given deceleration ramp. “paraMaxDecelerationAppl” is used for deceleration. It will decelerate the axis by the given deceleration ramp when the calculated position is reached and stop at the software limit switch. The original behavior is not modified, so if also these software limit-switches are activated, the axis might be set to state ERRORSTOP.



There are 2 different modes:

- EnableLimitAbort = TRUE
  Any ongoing motion will be aborted immediately (when the distance to stop is reached, as shown in the above diagram), a warning is shown
  The axis will be decelerated to reach the software limit switch.
- EnableLimitAbort =FALSE, EnableLimitDecelerate=TRUE
  A warning is shown and the velocity is reduced, with respect to the given deceleration and position limit.
  The ongoing motion is not aborted. If it was just a “tight fit”, e.g. in a master slave movement and the direction is turned soon enough, it might be possible to continue the movement.
  As the ongoing movement is not interrupted, an activated movement might not be completed, for example a MC_MoveAbsolute will never reach its target position. A warning is shown at function block CMC_Basic_Kernel.

When EnableLimitPos = TRUE or EnableLimitNeg = TRUE, and the values for SWLimitPos or SWLimitNeg are set, the axis will be set to state ERRORSTOP when these position limits are reached.

In addition, the function block will allow to limit the velocity. With EnableLimitVelocity = TRUE, it will monitor the velocity demand from the position reference and limit the position reference, so the given velocity limit will not be exceeded. A warning will be shown. The velocity used for limitation is MaxVelocityAppl.

The velocity limitation can be used to prevent short-term velocity peeks. The limited position will be caught up later, whenever possible. This can result in not-expected behavior. The WARNING issued by CMC_Basic_Kernel can be checked and used to stop a movement. The movement will be aborted automatically when the position is by MaxPositionGap behind.

- *For a single axis movement, the commanded velocity is limited at the beginning. No position gap will occur.*
- *In a multi-axis movement, the slave axis follows a master. This can result in a position gap. A velocity peek from the master axis can be reduced by using the limitation. If the master is too fast because of the value for MaxPositionGap, the movement will be aborted.*

When EnableLimit2Decelerate or EnableLimitAbort are used, the velocity is limited to MaxVelocitySystem with EnableLimitVelocity = FALSE. The function modifies the position reference. This modified position reference is used to control the drive. Whenever the limitation interferes the kernel will show a warning or an error. The warning or error message will disappear when the situation is cleared.

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	TRUE	ERRORSTOP when positions exceed, no previous warning or deceleration.
5	EnableLimitNeg	TRUE	
2003	EnableLimit2Decelerate	FALSE	
2004	EnableLimitAbort	FALSE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	FALSE/TRUE	Reduce the velocity when reaching a position limit within the deceleration distance calculated by using MaxDecelerationAppl. Display a warning at CMC_Basic_Kernel. The underlying movement stays active. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP also if EnableLimitPos or EnableLimitNeg are used. Otherwise, just the movement is limited, without affecting the state machine. An activated positioning movement will not reach its target. Velocity is limited to MaxVelocitySystem.
5	EnableLimitNeg	FALSE/TRUE	
2003	EnableLimit2Decelerate	TRUE	
2004	EnableLimitAbort	FALSE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	FALSE/TRUE	Reduce the velocity when reaching a position limit within the deceleration distance calculated by using MaxDecelerationAppl. Display a warning at CMC_Basic_Kernel. The underlying movement stays active. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP also if EnableLimitPos or EnableLimitNeg are used. Otherwise, just the movement is limited, without affecting the state machine. An activated positioning movement will not reach its target. Velocity is limited to MaxVelocitySystem. The active PLCopen function block is aborted as soon as the warning is issued. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP.
5	EnableLimitNeg	FALSE/TRUE	
2003	EnableLimit2Decelerate	---	
2004	EnableLimitAbort	TRUE	
2005	EnableLimtVelocity	FALSE	

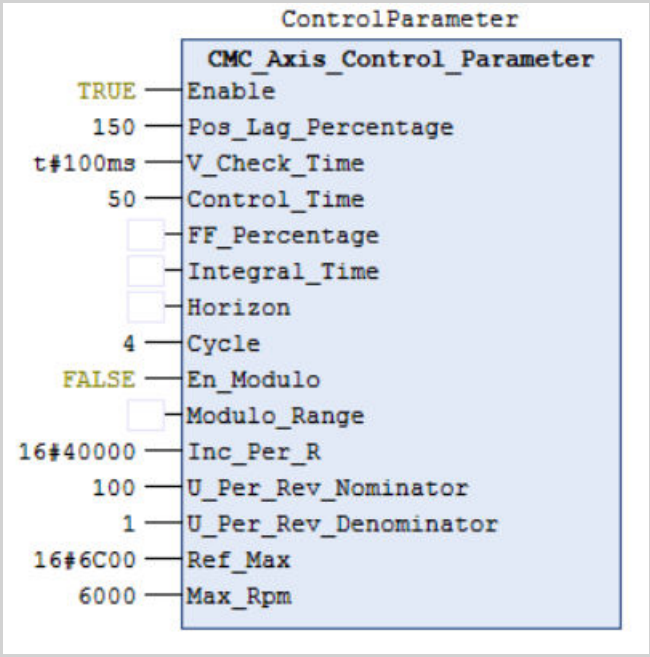
Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	---	The velocity is checked and also limited to the value Max-VelocityAppl. A warning is shown. The active movement is not aborted. This function-ality works independent from software limit switches.
5	EnableLimitNeg	---	
2003	EnableLimitDecelerate	---	
2004	EnableLimitAbort	---	
2005	EnableLimtVelocity	TRUE	

1.5.10.4.3 Axis parameters

The parameters for axis configuration and adjustment are set by the function blocks CMC_Axis_Control_Parameter.

Depending on the version of the kernel function block the corresponding version of the parameters function block has to be used. The instance will then be connected to the kernel function block by its instance name.

Example



In the example the control structure is a simple position control loop with just proportional gain. When the application does not require minimized position following error it should be used this way as it is simple to adjust, robust and requires minimal performance. The proportional gain is then adjusted by Control_Time. Just change values at CMC_Axis_Control_Parameter when the position control loop is open (Drive_Release=FALSE, the axis state is Disabled). The values are sending to the control loop whit a positive edge at "Enable". The CMC_Basic_Kernel function block needs to be already enabled.

Supervision

Pos_Lag_Percentage

This parameter configures the position window for the supervision of the following error.

The default value is 150[%]. A value of 0[%] will deactivate the supervision function.

The size of the position window depends on the setting of the parameters Control_Time and Max_Rpm ↳ *"Control_Time" on page 2354*.

Position Window [Increments] = (Inc_Per_R) * (Max_Rpm/60) * (Control_Time/1000)

Position Window [Units] = (U_Per_Rev_Nominator/ U_Per_Rev_Denominator) * (Max_Rpm/60) * (Control_Time/1000)

Example

Position Window [Increments] = (10000) * (6000/60) * (50/1000) = 50000 [Increments]

Position Window [Units] = (1/1) * (6000/60) * (50/1000) = 5 [Units]

A value of 100% will result in a position window which corresponds to the expected following error with the giving Control_Time at Max_Rpm. Therefore it is recommended to use values higher than 100[%]. In case the parameter FF_Percentage is used smaller values can be used.

If the supervised position window is exceeded the axis state will change to ERRORSTOP.

V_Check_Time

After the configured time the drive's actual velocity has to be at least 50 % of the commanded velocity. This function can also be used in case the Position Reference is transferred to the drive.

A value of 0 will deactivate this supervision function.

If the supervised velocity window is exceeded the axis state will change to ERRORSTOP.

Position control loop

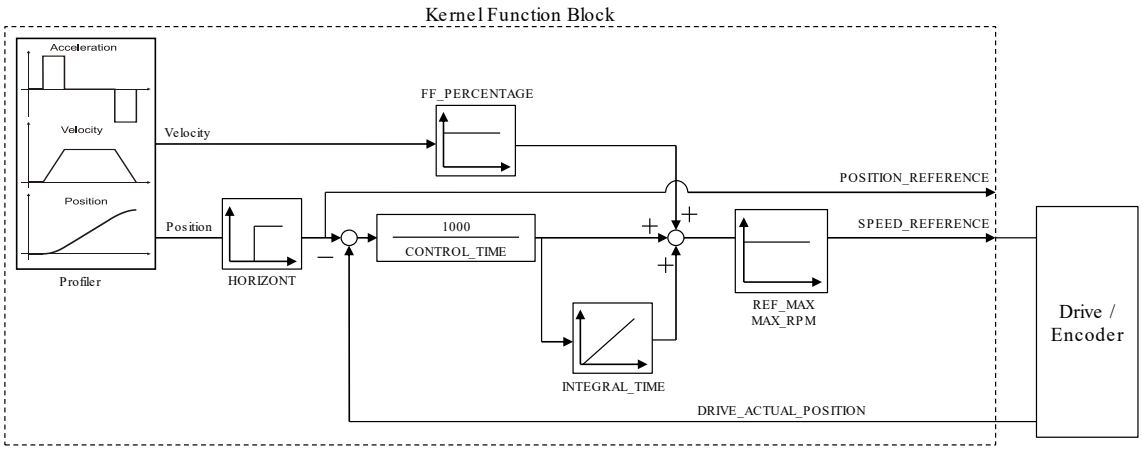


Fig. 81: Basic structure of position control loop

Control_Time

The default value is 100 which leads to a proportional gain of 10.

In case the value of Control Time is too short the position control loop will run into instability.

In case the position control loop is not used this parameter must not be set to 0.

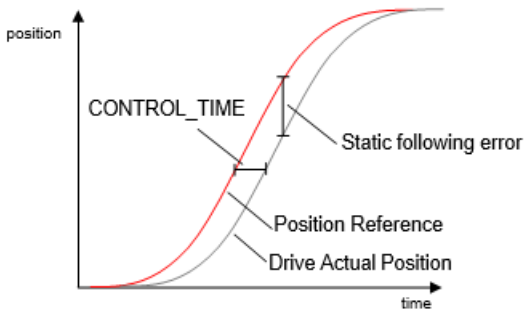
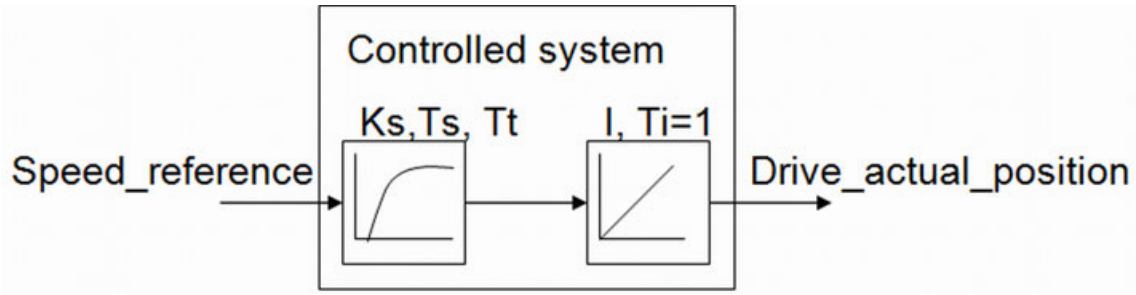


Fig. 82: Control Time and static following error in case the feed forward of velocity and the integrational part of the position control loop is not used.

The static following error depends on the axis velocity and can be calculated easily: Control Time multiplied by the axis velocity (p_error = v * CT).

In general it should be aimed to reach a high position control loop gain with a short Control Time to achieve a small following error. As the reaction times take account in the possible Control Time of the complete system (parameters of the drive control loop, PLC cycle time as well as the communication fieldbus) should be considered.

As a basic rule the Control Time should be at least four times longer than the reaction time between the output of the Speed Reference and the input of actual position.

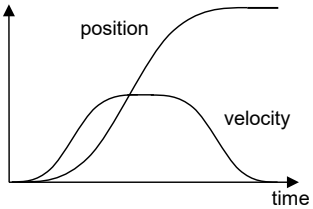


When the time Ts and Tt is measured, a control_time of 4 * (Ts + Tt) will result in an aperiodic damping of the position control loop. It is important to measure the values from inside the PLC (e.g. Trace) to have the complete reaction times included. Practical values for Control_Time might be from 50 - 500ms. The PLC cycle time as well as bus cycle times and mechanical reaction will influence the value.

FF_Percentage The default value is 0.

In case a velocity feedforward has to be configured a value of up to 80 is recommended. For larger values than 80 the parameter Horizon needs to be used as the resulted position will overshoot otherwise.

A value of 100 adds a velocity to the Speed Reference output which corresponds exactly to the ongoing Position Reference value.



Integral_Part The integral part of the position control loop can be used to eliminate a permanent positioning error, e.g. in case of hanging loads.

The time value can be regarded as the time the integrator needs to sum up the input value to reach the same value for its output.

In case the Integral Part Time is too short the position control loop will run into instability.

Horizon

A communication delay of the Speed Reference value to the drive system can cause an overshoot during positioning caused by the velocity feedforward gain.

This function will compensate this communication delay to prevent an overshoot by time shifting the signals Velocity Feed Forward and Position Reference relatively to each other.

The value of Horizon can be approximately assumed to be the time delay of the communication delay.

The delay time might be caused by the cycle time of the control loop and by any delay in sending the speed reference, delay in the drive to build up the torque and delay to receive the actual position. To overcome this delay, a Horizon > 0 might be used. The feed forward reference will be created in advance, while the proportional gain is applied to the original motion profile. The delay is then compensated.

This function should not be used if the feed forward parameter FF_Percentage is 0.

A value of 0 will deactivate this function, which is the default value.

While this function is used, it will increase the needed PLC calculation time for this axis.

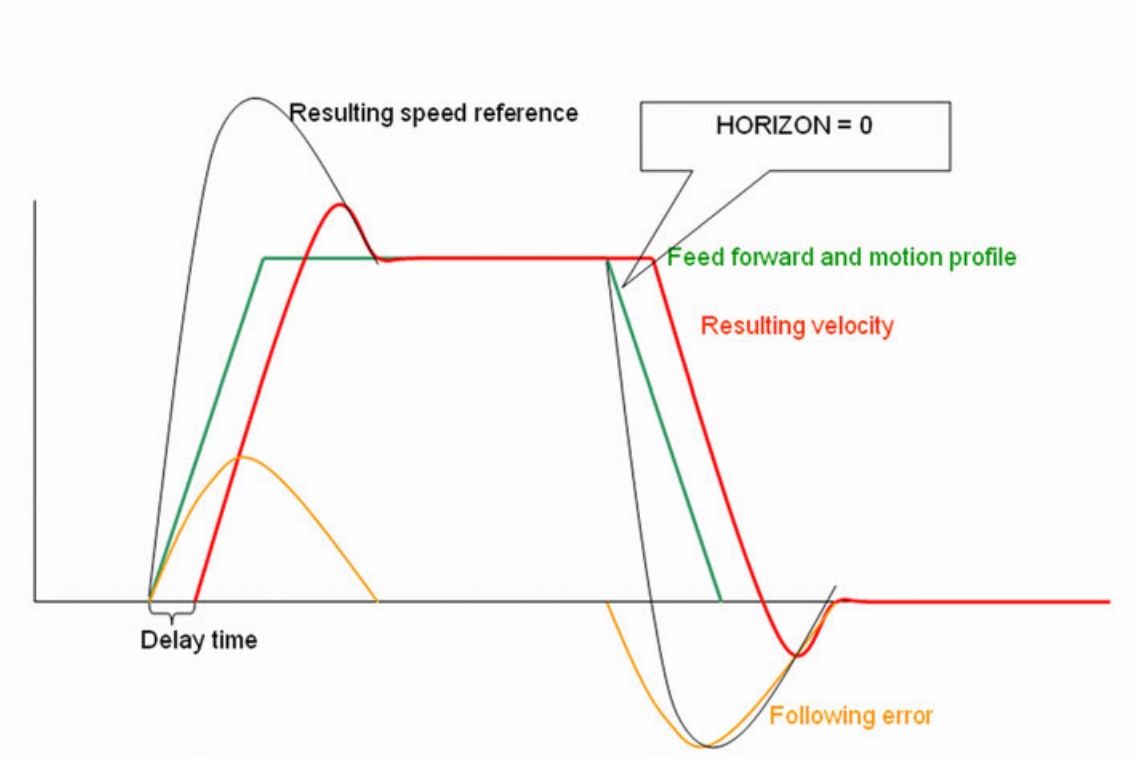


Fig. 83: Result with Horizon=0

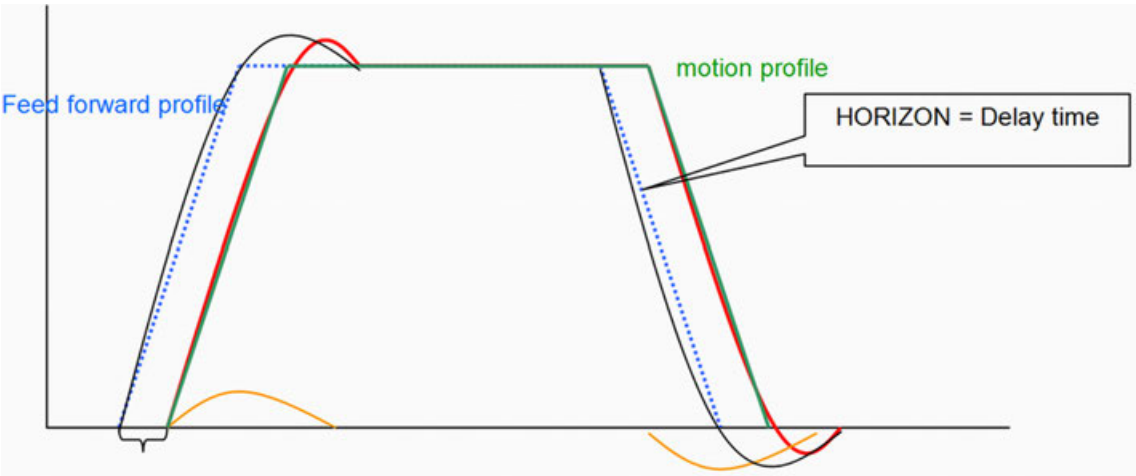


Fig. 84: Result with Horizon>0

PLC cycle time

Cycle

This parameter represents the cycle time in which the kernel function block of the axis is called. If the configured cycle time is not correct the resulting acceleration and speed of an axis will be not correct also.

In case the task execution of the axis is synchronized to a fieldbus (e.g. EtherCAT) the cycle time of the fieldbus has to be used.

Roll-Over axis

If the Position Reference value is used, the drive must able to perform a position over-run after 32 bit. If the drive's position over-run is different, it can be adapted with the function blocks CMC_Binary2Modulo and CMC_Modulo2Binary from the library ABB_MotionControl_AC500.library. Incompatibility can cause an axis to trip after hours of operation.

The possible position following error has to be smaller the ½ Modulo_Range. Make sure that the modulo range is large enough.

Position following error = (100 - FF_Percentage) * Max_Rpm * Inc_Per_R * Control_Time/6000000. This is the maximum value at constant velocity.

En_Modulo

With this parameter the axis can be configured as a roll-over axis.

Modulo_Range

The modulo range will be defined in drive position counts (DINT). It will result that the scaled unit position which is used by the PLCopen function blocks will stay within the defined range.

Example

```
En_Modulo             = TRUE
Modulo_Range          = 20000
Inc_Per_Rev             = 10000
U_Per_Rev_Nominator      = 360      (e.g. degree)
U_Per_Rev_Denominator      = 1
```

The scaled unit's position will cover the range from 0 to 720 (degrees).

In some cases it is not suitable to set the modulo range of an application with the DINT value of the parameter Modulo_Range only. In such cases the parameters 2001 Modulo_Nominator and 2002 Modulo_Denominator can be used to scale the parameter Modulo_Range to a more precise value.

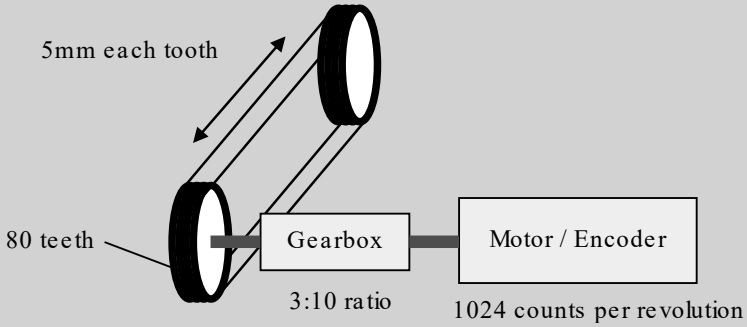
Parameter Modulo_Nominator and Modulo_Denominator (supported with CMC_Basic_Kernel)

These parameters can be used to modify the Modulo_Range in a way that fractions of an increment could be used for 1 modulo (=rollover) distance

- Default: Modulo_Nominator=1 and Modulo_Denominator=1: the actual position for an axis is limited between 0 and Modulo_Range increments.
- Limitations: Modulo_Range*Modulo_Nominator < 2147483647. Otherwise: default values will be used.
- When modifying these parameters, the position control loop should be opened.

Example

```
En_Modulo             = TRUE
Modulo_Range          = 1024
Modulo_Nominator          = 10
Modulo_Denominator          = 3
Inc_Per_R             = 1024
U_Per_Rev_Nominator      = 80*5*3
U_Per_Rev_Denominator      = 10
```



Result of parameters Modulo_Range, Modulo_Nominator and Modulo_Denominator: The modulo range will cover one revolution of the toothed-belt wheel.

Result of parameters U_Per_Rev_Nominator and U_Per_Rev_Denominator: One scaled unit corresponds to one mm of the tooth belt.

Example: Gearbox 10.1

	Option1	Option2
En_Modulo	TRUE	TRUE
Modulo_Range	10240	10240
Modulo_Nominator	1	1
Modulo_Denominator	1	1
Inc_Per_R	1024	10240
U_Per_Rev_Nominator	36	360
U_Per_Rev_Denominator	1	1
Max_Rpm	3000	300

The two options above describe exactly the same configuration. The Modulo_Range is equivalent to 10 motor revolutions and is 10240 increments. For the position, 1u means 1° and the resolution is 360°/10240inc = 0,035°/Inc = 1°/28,44 Inc.

Example: Gearbox 10.3

	Option1	Option2
En_Modulo	TRUE	TRUE
Modulo_Range	1024	10240
Modulo_Nominator	10	1
Modulo_Denominator	3	3
Inc_Per_R	1024	10240
U_Per_Rev_Nominator	108	1080
U_Per_Rev_Denominator	1	1
Max_Rpm	3000	300

The two options above describe exactly the same configuration. The gearbox is 10:3, so the Modulo_Range is equivalent to 1024*10/3 = 3413 + 1/3 increments. For the first option, the resulting modulo range is calculated 1024*10/3, for option2, it is 10240*1/3. For the position, 1u means 1° and the resolution is 108°/1024inc = 0,105°/Inc = 1°/9.481 Inc.

Scaling of the unit of length

Inc_Per_R With this parameter the number of the drive position counts each revolution of the motor (DINT) have to be entered.

U_Per_Rev_Denominator & U_Per_Rev_Nominator With these two parameters the number of units which correspond to one revolution of the motor have to be entered.

The units of length can be scaled to values like: mm, inch, degree, …

All dynamic parameters of the PLCopen function blocks like velocity, acceleration and jerk are based on seconds. Velocity [units/s], acceleration [units/s²], jerk [units/s³]

Example 1

```
Inc_Per_Rev            = 10000
U_Per_Rev_Nominator     = 360
U_Per_Rev_Denominator     = 1
```

This will scale one unit to one degrees of the motor shaft. Correspondingly a velocity [units/s] of 360 will turn the motor shaft one revolution per second.

Example 2 In the example one unit will be scaled to one millimeter of the conveyor.

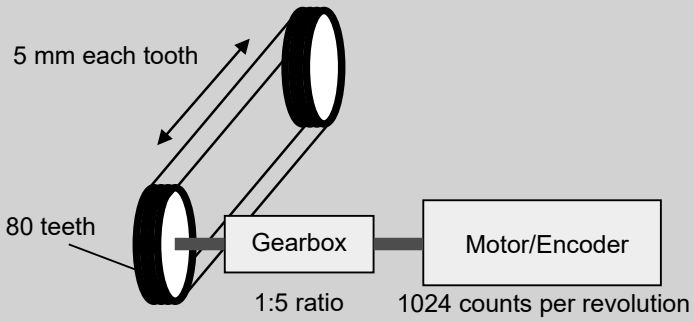


Fig. 85: Scaling units

How many units will pass after one revolution of the motor? (80*5mm) / 5 = 80

```
Inc_Per_Rev            = 1024
U_Per_Rev_Nominator     = 80
U_Per_Rev_Denominator     = 1
```

Example 3 In the example one unit will be scaled to one millimeter of the conveyor.

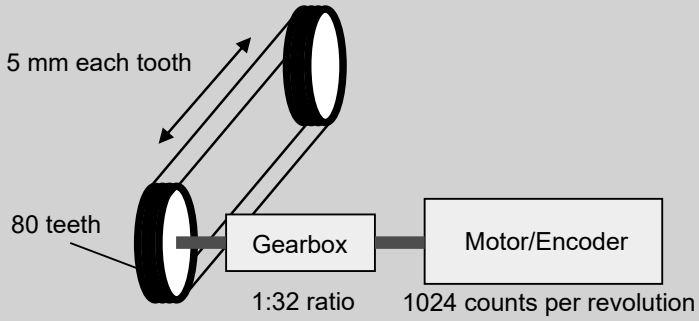


Fig. 86: Scaling units

How many units will pass after one revolution of the motor? (80*5mm) / 32 = 12,5 = 125 / 10

```
Inc_Per_Rev             = 1024
U_Per_Rev_Nominator      = 125
U_Per_Rev_Denominator      = 10
```

Scaling of the speed reference output

These two parameters are used to scale Speed Reference output of the kernel FB in order to reach the intended velocity by the output value and to limit the highest possible output value.

Ref_Max Highest possible output value of the Speed Reference output. The Speed Reference value that corresponds to the parameter Max_Rpm should be used.

Max_Rpm Maximum speed of the motor in revolutions per minute.

Example

- Analog Drive: 1000 rpm at 2 Volts, 3200 rpm at 6,4 Volts (max.)
- Analog output module: 10 Volts output at digital value 27648
- Ref_Max = 17695 (= 27648 / 10 * 6,4)
- Max_Rpm = 3200

Access and modify parameters

All modifications will be effective immediately. There is no extra plausibility check and values are not checked for limitations.

Use this functionality with care.

Some parameters are collected inside a structure in Axis_Ref, and can be accessed and modified immediately. They are the same parameters as used with function blocks MC_Write-Parameter and MC_ReadParameter *Chapter 1.5.10.3.6 “PLCopen parameter” on page 2308*.

The differences are:

- Only available with CMC_Basic_Kernel
- The parameter values are LREAL instead of DINT and can be used with decimals.
- The parameters will be effective immediately.
- There is no check for consistency or limits.
- The parameters for position control can be checked and modified by accessing the structure Axis_Parameter.CMC_Pos_Control in addition.

Parameter for position control	Description
KP	Proportional gain in positive direction. Used directly to multiply the following error and create the Reference_Prop.
KF	Feed forward in positive direction. Used directly to multiply the speed reference and create the Reference_FF.
KP_BACK	Proportional gain in negative direction. Used directly to multiply the following error and create the Reference_Prop.
KF_BACK	Feed forward in negative direction. Used directly to multiply the speed reference and create the Reference_FF.
TI	Integration time. When parameter is used the position control loop has an additional integral part. In TI cycle, the Reference_ITG will reach the value of Reference_Prop, when KI=100*KP.
KI	Proportional gain, used for integral part of position control loop.
KF_100	Value for feed forward gain, if 100% would be used.
Max_Time	Delay time used for supervision of velocity. With Max_Time=0, no supervision is executed.
D_XS_Max	Maximum possible velocity in [u/cycle]. The maximum allowed following error is part of the parameter structure, PLCopen parameter paraMaxPositionLag.
Ref_Max	Limit for Speed_Reference.

Element actual of Axis_Ref

The element `actual` represents actual values from inside the position control loop.

Value	Description
Position	Actual position in [u] to control the axis.
Control_Position	Reference position in [u] which is actually used for control loop.
D_XS	Distance in [u] to be moved per cycle.
D_XSS	Following error in [u].
Reference_Prop	Proportional part for Speed_Reference.
Reference_FF	Feed forward part for Speed_Reference.
Reference_ITG	Integral part for Speed_Reference.

Possible to use different gain for forward/ backward movement , possible improvement for hydraulic axis or vertical movement

See parameter KP/KP_BACK and KF/KF_BACK.

Limitation for velocity and acceleration and deceleration

From library version 3.1 on, these values are not limited to the 16-bit range of values (32767). The limit for velocity is calculated by the values given at CMC_Axis_Control_Parameter and the acceleration is limited such that this velocity can not be reached faster than 1 cycle.

1.5.10.4.4 Programming guidelines

To achieve the best results for Motion Control the actual position has to be transferred in best possible quality (with minimal jitter) to the PLC. The position feedback is expected to be in increments as the data type is a DINT.

The kernel function block (CMC_Basic_Kernel or OBIO_PTOMotionKernel or OBIO_PWMMotionKernel) has to be called every cycle and its task requires a fixed cycle time.

A variable of type Axis_Ref is used to connect to the PLCopen function blocks and their kernel function block.

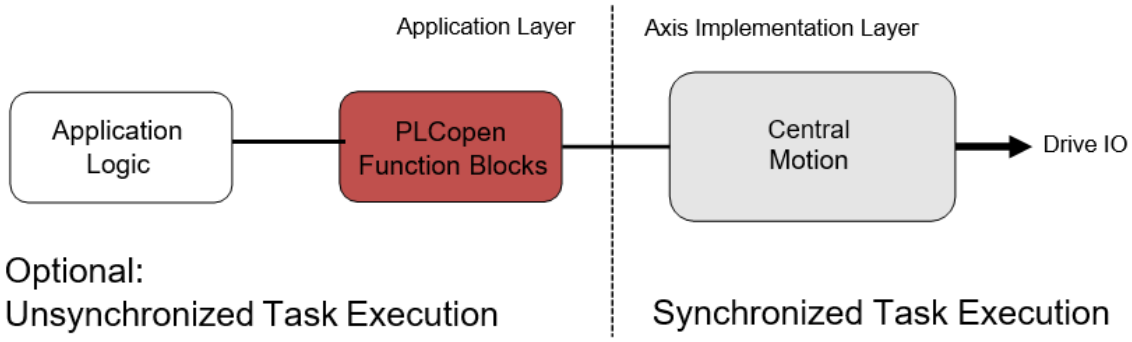
The function block CMC_Axis_Control_Parameter has to be used for the axis configuration.
↳ *Chapter 1.5.10.4.3 “Axis parameters” on page 2353*

The signal of the limits switches and the absolute switch should be connected to the elements of the data type CMC_Axis_IO. The signal of the absolute switch must be TRUE in case the axis hits the sensor. The signal of a corresponding limit switch has to be true when the axis leaves the area surrounded by the limit switches. If needed the signal has to be inverted before it is connected to the elements of the data type.

Task configuration

The kernel function block and the transfer of axis IO data should be processed in a cyclic task. This task should be as short and real-time as possible to achieve the best motion control performance. Always make sure Kernel function block is called at the highest priority task and other applications must be at a lower priority task.

In order to save PLC processing time the most PLCopen function blocks as well as the application logic can also be processed in a task which runs on a lower priority than the real-time task with the axis implementation as shown in the figure below.



All PLCopen function blocks which must be called in the same task than the kernel function block:

- MC_CombineAxes
- MCA_MoveByExternalReference

In case the position reference is transferred to the drive the task of the axis implementation should be synchronized to the fieldbus cycle. The following figures show an example for EtherCAT:

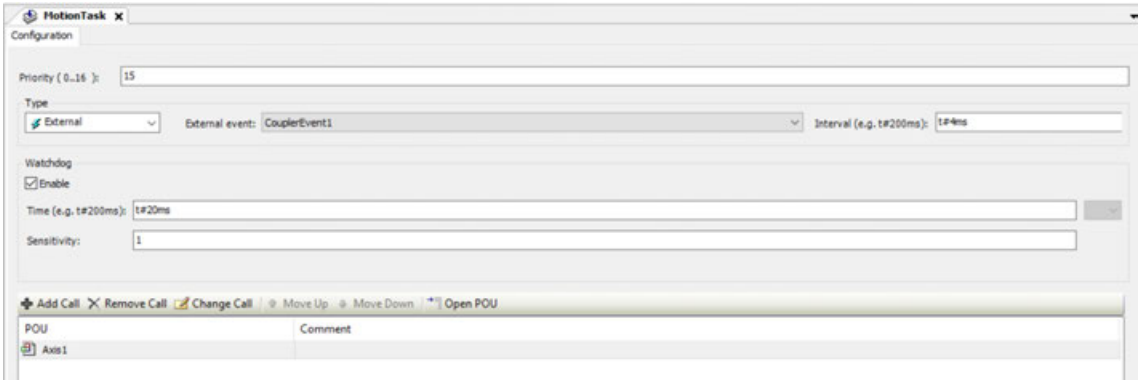


Fig. 87: Task of axis layer

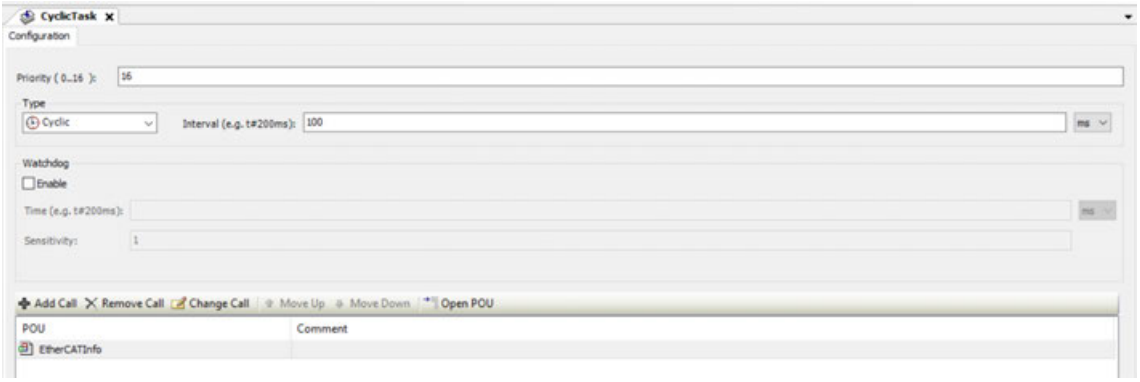
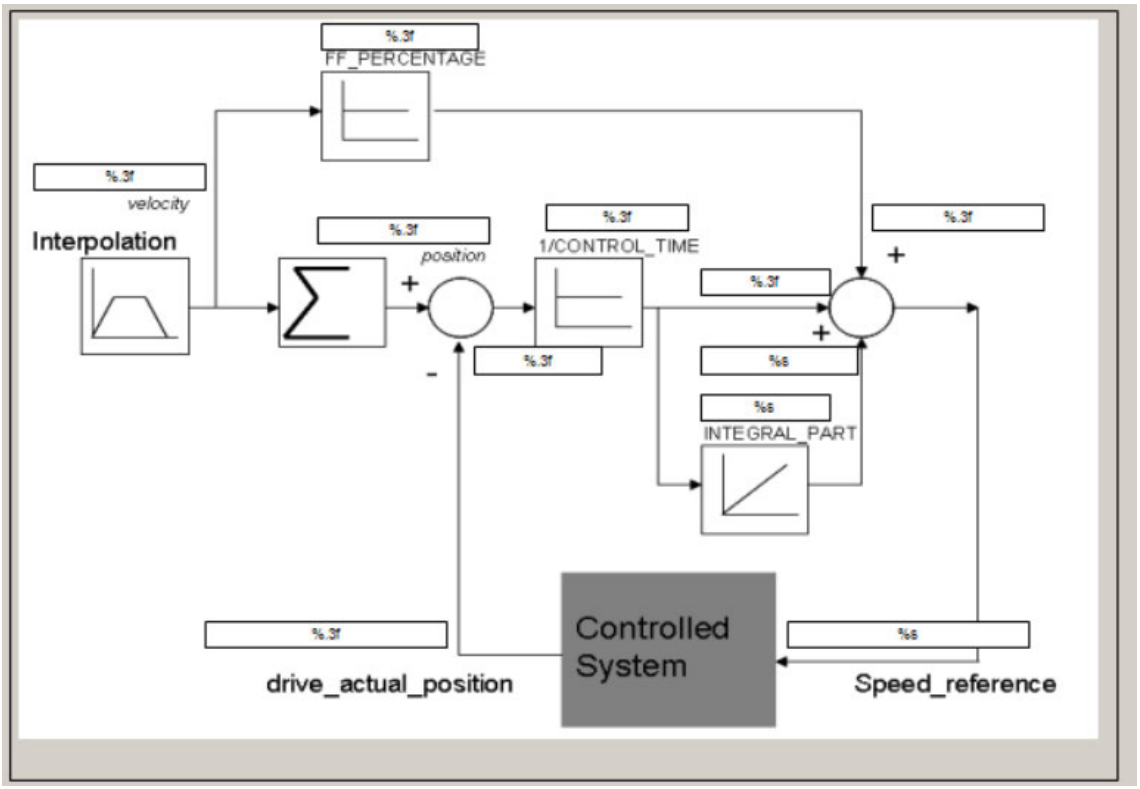


Fig. 88: Task of application implementation

1.5.10.4.5 Visualization

The structure of the position control loop is also as visualization element CMC_Visu_FB_Basic_Kernel. included in ABB_MotionControl_AC500.library. As placeholder, an instance of CMC_Basic_Kernel has to be used. The visualization shows all numbers as they are really used inside the block, the adjustment for different resolution or cycle times is already included.



1.5.10.4.6 ABB specific data structures

Not all data structures are defined by PLCopen. Some specific structures are described in the following chapter. In addition to the data in these arrays, the movement is modified by offset and scaling values at the respective function block. These offset and scaling values (except the time-scale) are transferred continuously. This will allow to follow a "Moving Target" by adjusting these values.

PositionPositionProfile

The data type MC_PProfile is used for CamTable. An array has to be defined and provided at MC_CamTableSelect. Several CamTables could be defined and the axis could change between them on the fly. There is no routine of smooth movement from one table to the next so the user has to take care just to switch on appropriate positions. Details are described in the documentation included with the library.

Declaration example CAM_table

```
ARRAY[1..3] OF MC_PProfile:=
    (Master_position:=    0    ,interpolation_point :=
0    ,Velocity_ratio:= 0    ,Acceleration_ratio:= 0 ),
(Master_position:=    50    ,interpolation_point :=
25    ,Velocity_ratio:= 0    ,Acceleration_ratio:= 0 ),
(Master_position:=    100    ,interpolation_point :=
0    ,Velocity_ratio:= 0    ,Acceleration_ratio:= 0 );
```

PositionTimeProfile

This structure is used for time based profiles, e.g. MC_PositionProfile:

Interpolation types for profiles

The curves defined by an array of MC_PProfile hold master position points and according slave positions. When the master position is between 2 points, the according position for the slave is interpolated. Different types of interpolation are possible. The type is defined in MC_ABB_iTypes_Enum . The master could be a real axis or some virtual axis which could be created by just writing values for position and velocity to the Axis_Master variable as shown in the example. The same interpolation types could be used on MC_TProfile.

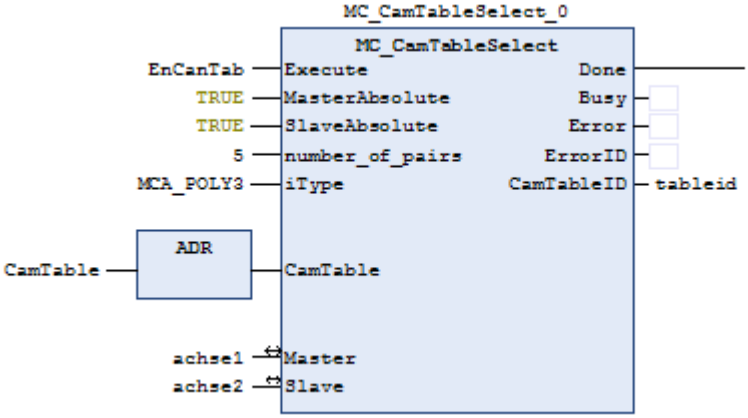


Table 396: Overview of different interpolations

Interpolation Types	Results in	Requires
MCA_LINEAR	Linear interpolation with constant velocity between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point
MCA_SPLINE_NATURAL	Cubic spline interpolation without jerk.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point

Interpolation Types	Results in	Requires
MCA_SPLINE_COMPLETE	Cubic spline interpolation without jerk, start and end of profile with velocity=0.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point
MCA_POLY3	Polynomial interpolation with linear velocity between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point, profile.MC_PProfile_Array[x].velocity_ratio
MCA_POLY5	Polynomial interpolation with linear acceleration between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point, profile.MC_PProfile_Array[x].velocity_ratio, profile.MC_PProfile_Array[x].acceleration_ratio

The interpolations allow to run on smooth curves without the need to define a large number of points. The following chapter shows the results with different interpolation modes for a sinus-curve with 10 interpolation points. The following table gives the mean deviation.

Interpolation Type	Mean deviation [ppm]
MCA_LINEAR	19686 =1.9%
MCA_SPLINE_NATURAL	151=0.0151%
MCA_SPLINE_COMPLETE	25510=2.5%
MCA_POLY3	131=0.0131%
MCA_POLY5	0.37

The original curve is represented by y_sinus for position and v_sinus for velocity. The diagrams show the result which is achieved by different interpolation types.

MCA_LINEAR

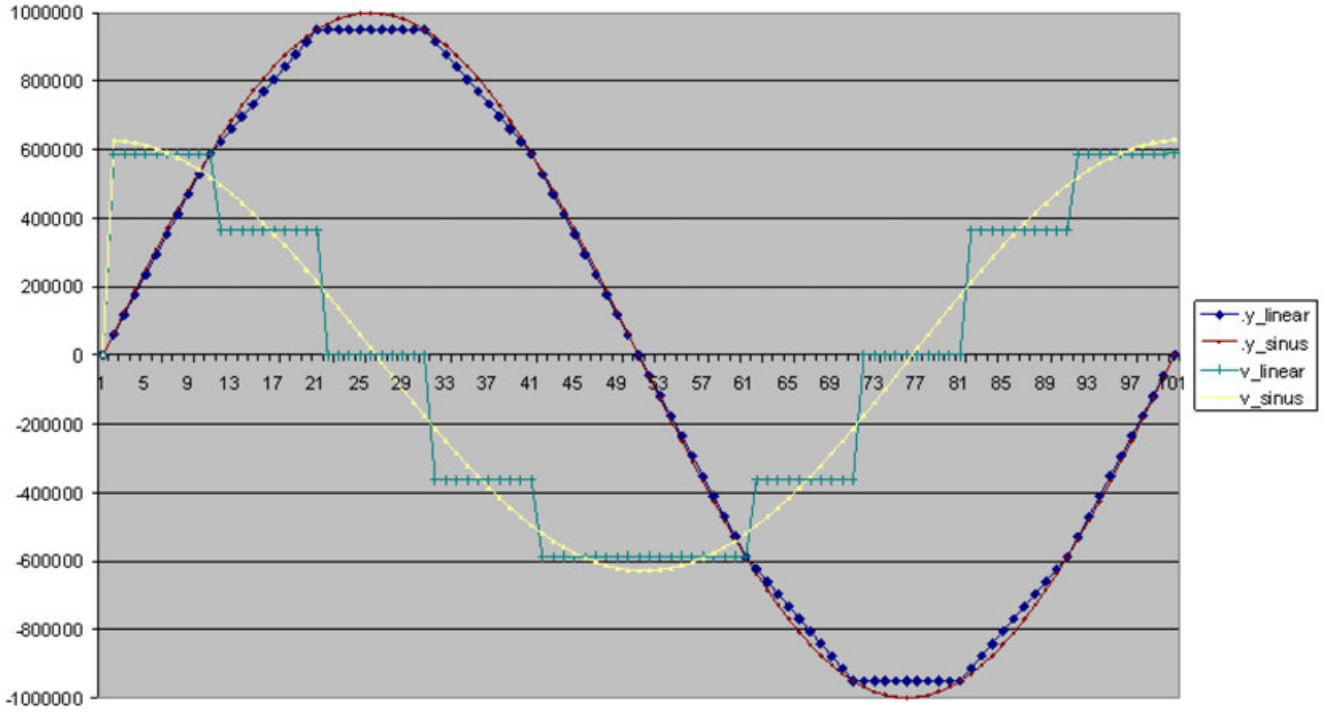


Fig. 89: Results from linear interpolation

The velocity is constant between the interpolation points.

MCA_POLY3

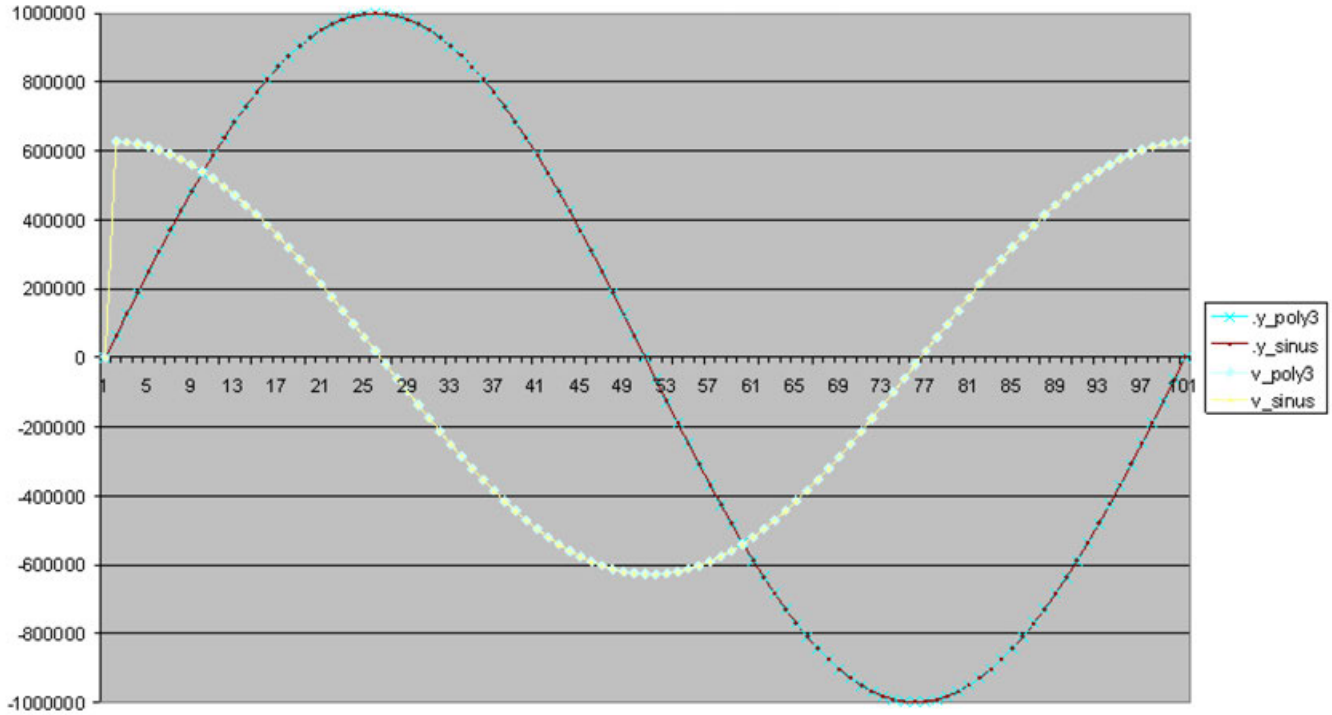


Fig. 90: Results from polynomial interpolation

The result looks almost identical to the original curve. The mean deviation shows that MCA_POLY3, MCA_POLY5 and MCA_SPLINE_NATURAL produce results which follow the original curve really good and are almost identical. The spline interpolation produces a jerk-free curve without the need of providing velocity values and acceleration values in advance.

MCA_COMPLETE

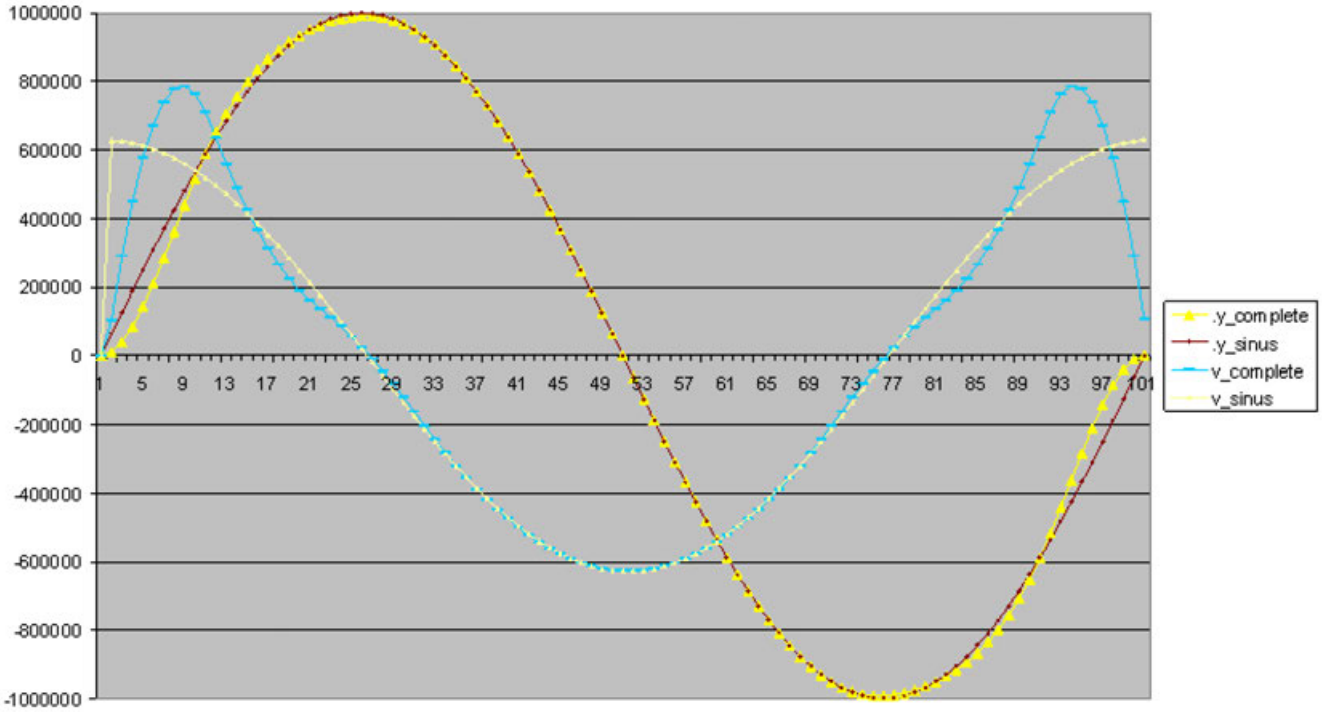


Fig. 91: Results from complete spline interpolation

In the beginning and the end, the curve does not follow the original curve. The reason is that it starts with velocity=0 and produces a jerk free result.

So the favoured result has to be considered in advance to choose the right interpolation method. With these different methods it is not necessary to provide a large number of interpolation points to get good results and smooth acceleration and deceleration ramps.

1.5.10.4.7 PLC-based motion control — Load control / fluid power extensions

The ABB_MotionControlLoad_AC500 library is an extension to ABB_MotionControl_AC500 library based on PLCopen part 6 called “fluid power” and basically can be used to implement load control as a simple form of torque profiling. It can be used together with all other motion control package libraries. The same structure and general rules are applied and all the above chapters in this document is relevant for ABB_MotionControlLoad_AC500 library as well. It is recommended to read through all the above chapters before start using the function blocks from this library. A difference is that the position control loop has to be closed inside the PLC as it is to be synchronized with the load control loop which is also realized. The implementation of Load function blocks is based on the PLCopen part 6 – Fluid power.

Overview of the defined extended function blocks:

Table 397: Overview of the defined function blocks

Administrative		**Motion**	
Single axis	**Multiple axis**	**Single axis**	**Multiple axis**
MC_LimitLoad	-	MC_LoadControl	-
MC_LimitMotion	-	MC_LoadSuperIm-posed	-
-	-	MC_LoadProfile	-
-	-	MC_TorqueControl	-

As per PLCopen MC_TorqueControl is a part 1 function block, however due to its implementation as a wrapper for the load control and limit load blocks this is added to ABB_MotionControlLoad_AC500 library.

The following state diagram is based on the version as defined in 'Part 1 – Function Blocks for Motion Control', Version 2.0.

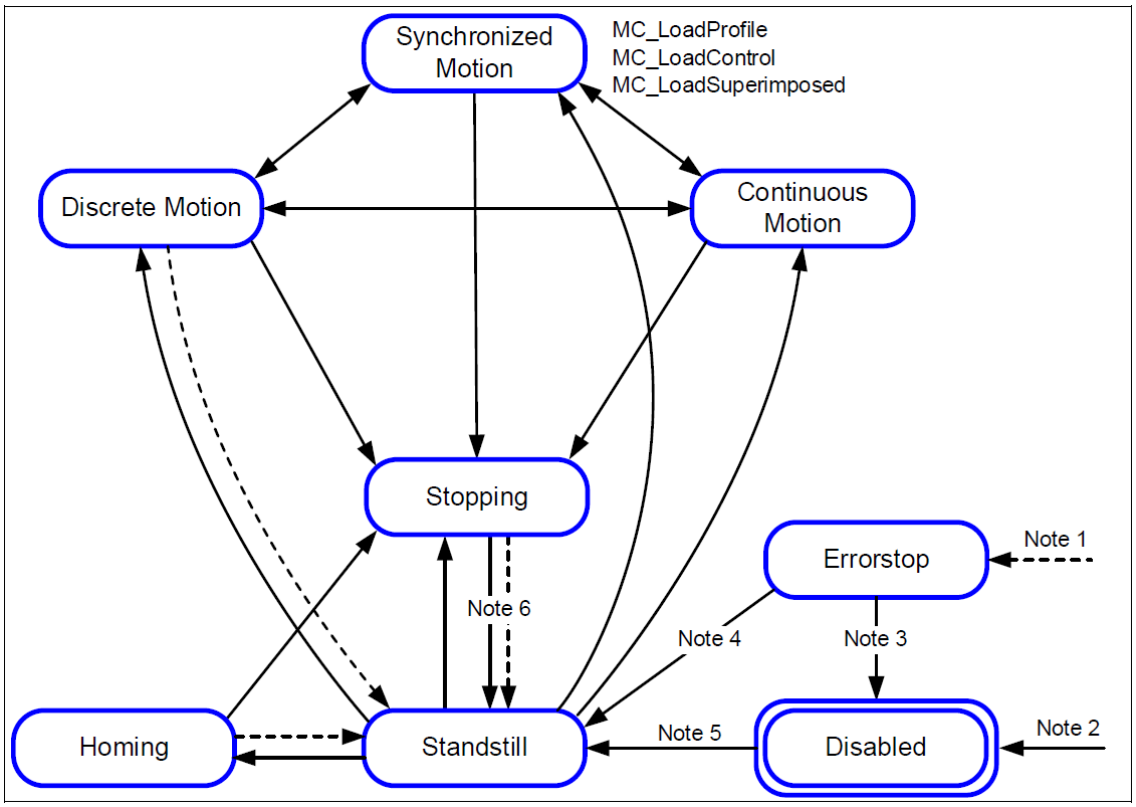
This specification adds three load function blocks to the state diagram:

- MC_LoadControl
- MC_LoadSuperImposed
- MC_LoadProfile

MC_TorqueControl function block also follows the same state diagram.

Function blocks not listed in the state diagram do not affect the state diagram, meaning that whenever they are called the state does not change.

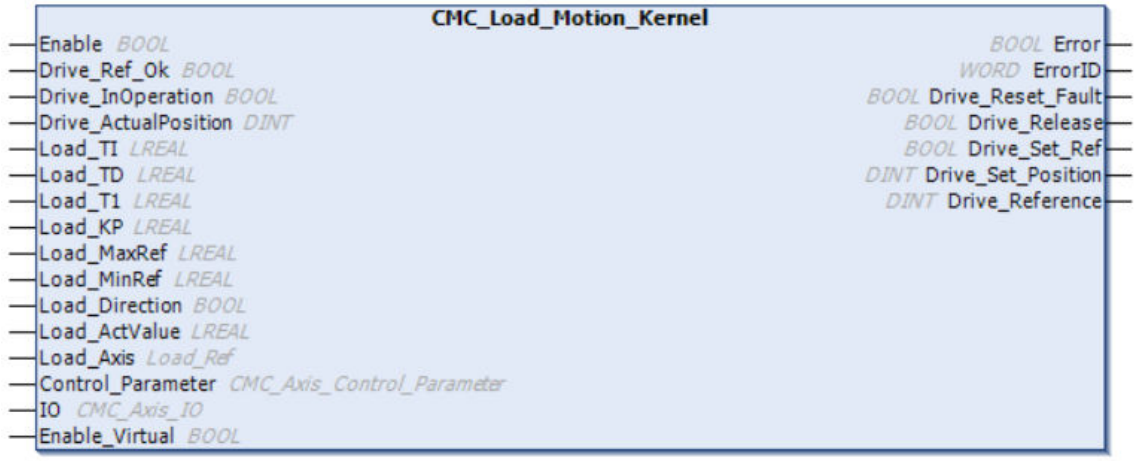
The state diagram shows synchronized motion because the position-axis follows the load, and the state is related to the position axis.

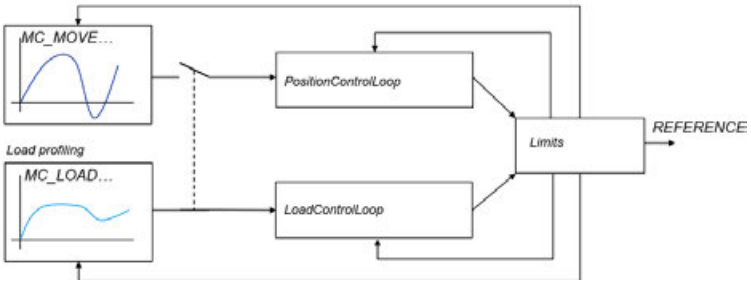


Note 1: From any state. An error in the axis occurred.
Note 2: From any state. MC_Power.Enable = FALSE and there is no error in the axis
Note 3: MC_Reset and MC_Power.Status = FALSE
Note 4: MC_Reset and MC_Power.Status = TRUE and MC_Power.Enable = TRUE
Note 5: MC_Power.Enable = TRUE and MC_Power.Status = TRUE
Note 6: MC_Stop.Done = TRUE and MC_Stop.Execute = FALSE

Kernel function block - Fluid power

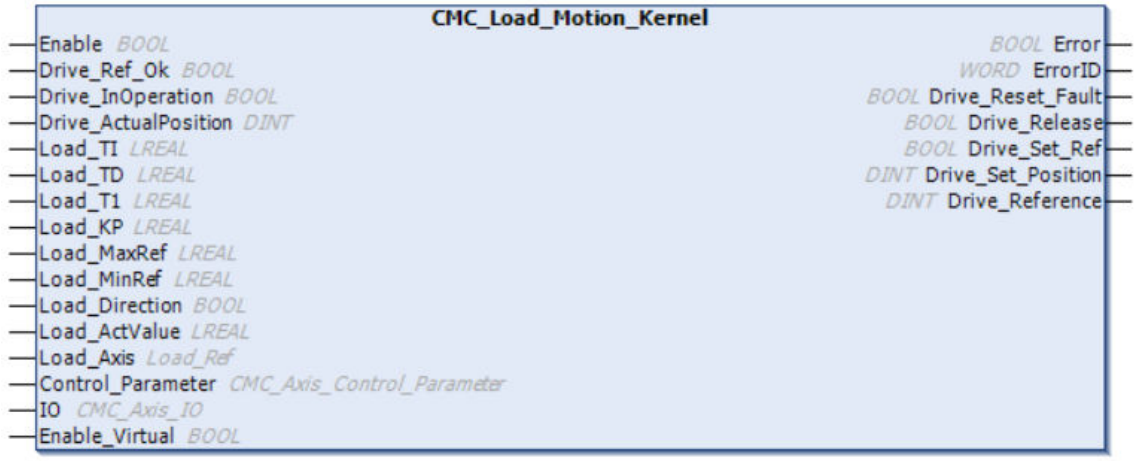
The basic block is the CMC_Load_Motion_Kernel. It has to be called every cycle and at least once before any MC… block is activated. It is used to combine the position and velocity functionality from CMC_Basic_Kernel with the load control functionality which is utilized by the MC_Load... blocks.





The reference which is used by the CMC_Load_Motion_Kernel is equivalent with the Speed_Reference at CMC_Basic_Kernel, as long as no LOAD-functionality is activated. The documentation from CMC_Basic_Kernel applies to the identical inputs and outputs. Some inputs and outputs are added to serve the load control functionality.

> *The Load_Ref is used instead of Axis_Ref for the MC_Loadxxx blocks. When the CMC_Load_Motion_Kernel is used, Load_Ref replaces Axis_Ref and user can use all PLCopen-Blocks.*



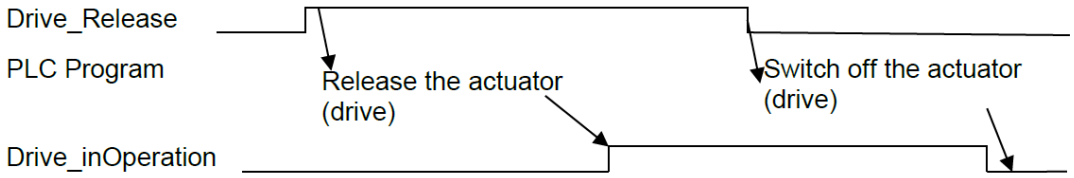
The actuator (drive) has to be accessed outside the CMC_Load_Motion_Kernel block. Actual values and reference values might be transferred by a synchronised bus or by I/Os.

- •All inputs and outputs of the function block which are named “DRIVE_xxxx” should be used to connect to the actuator (drive). It does not matter whether this connection is done by fieldbus or by conventional IOs.

  The Axis-structure is used to connect to the PLCopen blocks
- The Load_Axis structure is used to connect the fluid-power PLCopen blocks

- The control_parameter-structure is used for configuration of control loop.
- The IO-structure gives a connection to limit- or reference switches.

When the function block will take control (close loop) the output “Drive_Release“ is set. The PLC-Program should then start the actuator (actuator (drive)) and set “Drive_InOperation = TRUE” when successful. In case of actuator (actuator (drive)) problem, “Drive_InOperation” should be reset. The function block will then open the position control loop and Speed_Reference will be 0.



The homing is done with PLCopen-blocks. As the interface to the actual position is outside the CompactMotion, the bit “Drive_Set_Ref” is set when the state is reached to evaluate the zero-track. When the zero-track was found, Drive_ActualPosition has to be set to “Drive_Set_Position”, this has to be indicated by “Drive_Ref_Ok”.

The output “Drive Reference” should be send to the actuator (drive). This value is scaled with Max_Rpm and Max_Reference which means: when “Drive_Reference” equals Max_Reference, the motor is expected to run with Max_Rpm.

Load control

The function block holds a position control loop and a load control loop. The load control loop is a PIDT1-Block. Both control loops are alternately activated, depending if a MC_Load..block or a MC_Move… block is active. There is a bumbless transition realized between the different control loops.

The PIDT1 controller has a proportional, integral and derivative part. The integral and derivative part can be switched of by using a time value = 0.

Transfer function

$$F(s) = KP * \left(1 + \frac{1}{s * TN} + \frac{s * TV}{1 + (s\,T1)}\right)$$

Control algorithm: Simple rectangle rule:

$$Y = \frac{KP*XD}{100} + \frac{KP}{100} * \frac{XD}{TN\_TZ} + YI(z\text{-}1) + \frac{T1\_TZ}{1+(T1\_TZ)} * (YDTI(z\text{-}1) + \frac{1}{T1\_TZ} * \frac{TV}{TZ} * \frac{KP}{100} * (XD - XD(z\text{-}1)))$$

Where:
YI(z-1): The integral portion from the previous program cycle

(YDTI(z-1) The differential portion from the previous program cycle

XD(z-1): Control system difference from the previous program cycle

All 3 parts of the control loop are added up. The integral or derivative part could be disabled by setting the respective time constant to 0, so the following structures are possible:

- P
- PDT1
- PI
- PIDT1

The Load_MaxRef and Load_MinRef values will limit the controllers output Y and also apply to the controller’s internal integral part. I.e the integral part can only hold values between the high and low limits. If the manipulated variable Y reaches one of the two limits, the controller's integral part is no longer changed. This prevents the integral part from holding meaningless values and, in certain circumstances, not returning to the operating range for a long time. This behavior of a controller is also referred to as a "special anti-reset windup measure".

Example - Fluid power extensions

MC LimitLoad

In the diagram below, an example is explained. SFC is used here to distinguish between a movement where the MC_LimitLoad functionality has become ‘Active’ or not. In Step 2 there is a movement like ‘MoveAbsolute’, which is limited by the MC_LimitLoad functionality. If the absolute position is reached without MC_LimitLoad becoming active, the transition via done to step 3 is applicable. However, if the MC_LimitLoad becomes ‘Active’, the transition to the ‘Halt’ step is applicable, issuing a MC_Halt.

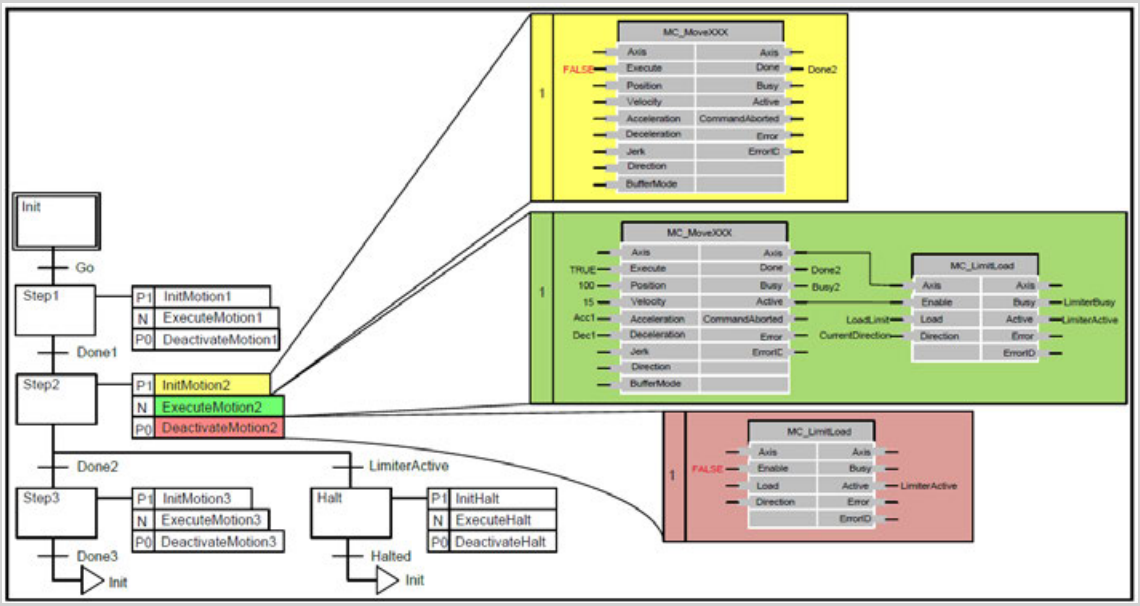


Fig. 92: MC_LimitLoad used in SFC

MC_LimitMotion e.g. force fitting

The function block is intended to be used in conjunction with a MC_LoadControl or MC_TorqueControl having primary control on the axis. The MC_LimitMotion should be enabled by the ‘Active’ output of the MC_LoadControl / MC_TorqueControl. If motion values on the axis exceed the given limit, appropriate measures are taken to keep to these limits, implying that the load/torque will not follow the programmed trajectory but depend on the external load conditions. However, the ‘Active’ output of the MC_LoadControl/MC_TorqueControl will stay TRUE in this case, following the modified PLCopen definition “The ‘Active’ output indicates, that the FB has control on the set-value generation of the axis”. This is despite the fact, that physically only the load-conditions or the movement of an axis can be controlled. With actual motion states below programmed limits, the programmed load/torque trajectory will proceed. Enabling the limiter block with activation of the MC_LoadControl/MC_TorqueControl ensures that limits are only supervised when the MC_LoadControl/MC_TorqueControl takes control on the axis for the first time. Disabling the limiter block with de-activation of the MC_LoadControl/MC_TorqueControl ensures that limits are no more supervised when the MC_LoadControl/MC_TorqueControl loses control on the axis by ‘CommandAborted’ or ‘Error’.

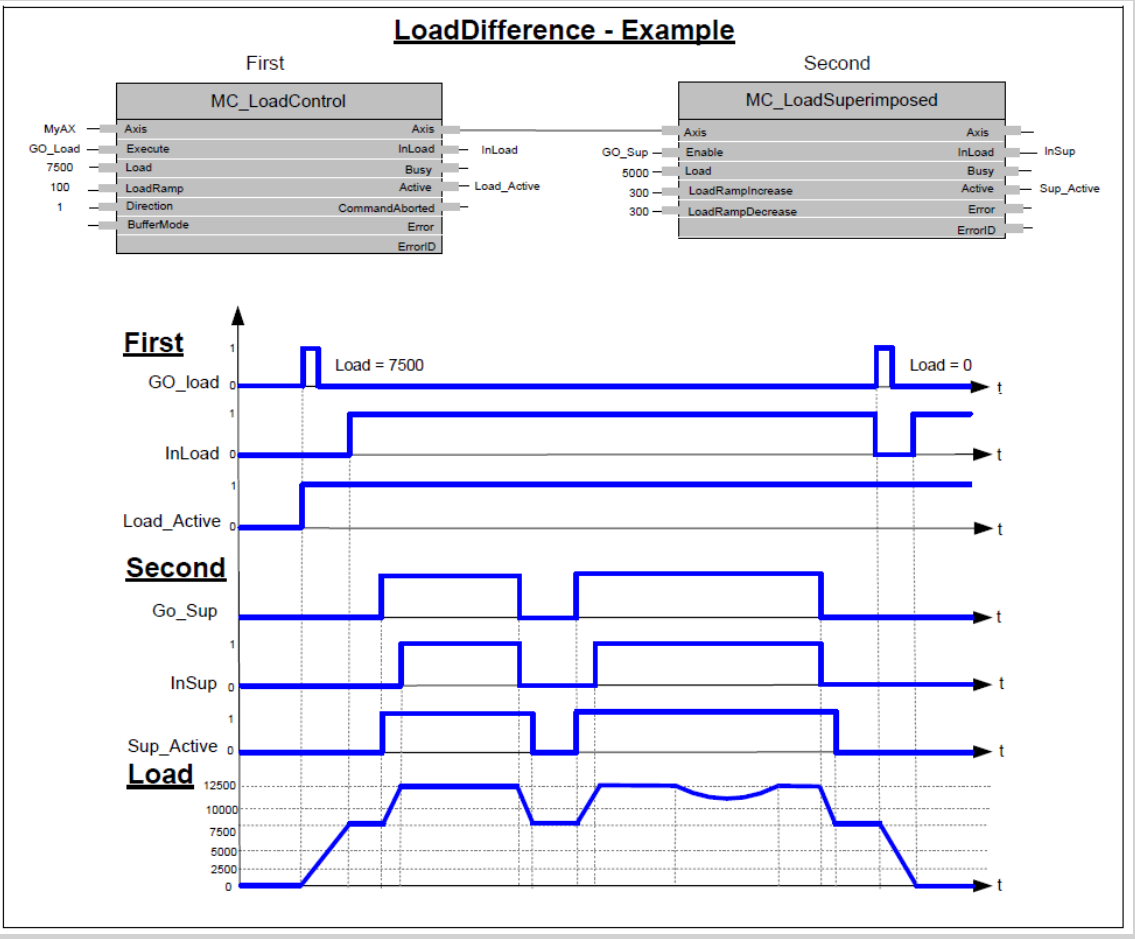
MC_ LoadSuperImposed

Possible Application: Actuator: hydraulic cylinder with fluid pressure sensor actuates the press of plastic injection molding machine in a continuous load operation.

Request: Prior to MC_LoadSuperImposed call, a MC_LoadControl block is ‘Active’ with a command of 7,500 kPa to press melted plastic into the mold. Once the MC_LoadControl ‘InLoad’ condition is achieved a superimposed pressure of 5,000 kPa is added several times to cause a hammering effect to relieve stresses in the plastic.

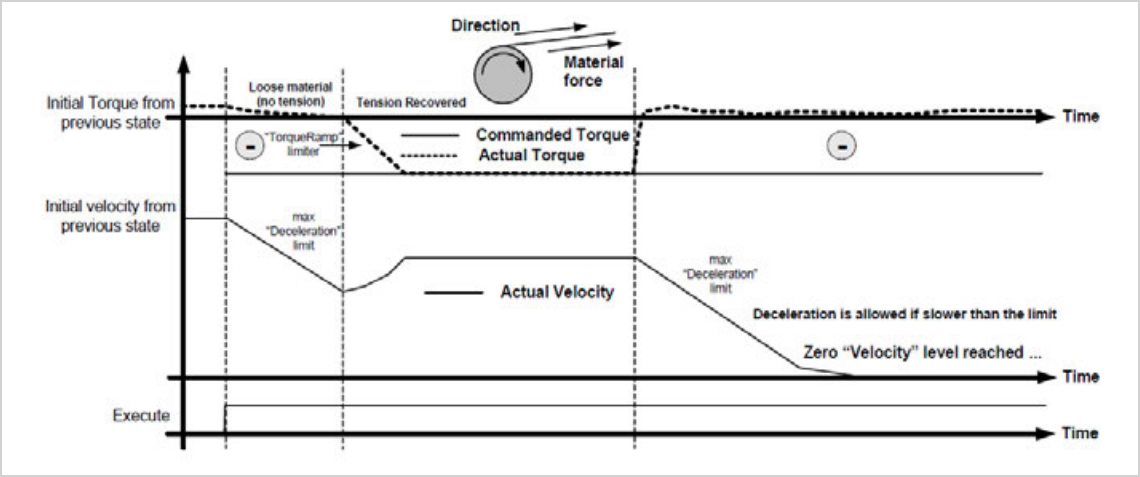
Result: the MC_LoadControl pressure command of 7,500 kPa is superimposed with a discrete pressure command of 5,000 kPa. Once the ‘LoadSuperImposed’ command is active the system pressure rises to 12,500 kPa.

When the superimposed pressure command has been achieved the MC_LoadSuperImposed block is done and the original command given by the MC_LoadControl resumes the original pressure command. The MC_LoadSuperImposed block is executed several times without affecting the original pressure command given by the MC_LoadControl block.



MC_Torque-Control

The example (below) opposite signs for ‘Direction’ & ‘Torque’ are used (e. g. Retention or brake control). (In the function block: +Direction –Torque). It is like an unwinding application with torque on the material, and a break in the material. When the material breaks, as shown in the middle of the figure this causes a drop in the real Torque value (in absolute terms): The velocity will decrease, limited by the fastest “deceleration” limit specified by the ‘Deceleration’ VAR_INPUT down to zero velocity (with no tension there is a risk of having shock breakings, so we have to limit to the fastest). In this case the torque setpoint might not be achieved.



In an unwinding application (derived from this brake control) material tension is the target, not motor torque. The instantaneous diameter of the roll should be taken into account to transform the “User tension setpoint”. Also, additional inertia compensation by modification of the torque setpoint for acceleration / deceleration is common from instantaneous weight data (weight is commonly estimated from diameter). Additionally, in unwinding applications, in the case of loose material (same condition as material break), a negative slow velocity reference is usually applied to “rewind” the loose material. In this case, this must be provided by external programming.

1.5.10.4.8 Appendix

List of all PLCopen and ABB specific function blocks in PS552-MC (for V2 PLC) and PS5611-Motion (for V3)

SNo	Funktion block type	Funktion block name	Motion Library V2 (PS552-MC)	Motion Library V3 (PS5611-Motion)
1	PLCopen	MC_Power	x	x
2	PLCopen	MC_Home	x	-
3	PLCopen	MC_Stop	x	x
4	PLCopen	MC_Halt	x	x
5	PLCopen	MC_MoveAbsolute	x	x
6	PLCopen	MC_MoveRelative	x	x
7	PLCopen	MC_MoveAdditive	x	x
8	PLCopen	MC_MoveSuperImposed	x	x

SNo	Funktion block type	Funktion block name	Motion Library V2 (PS552-MC)	Motion Library V3 (PS5611-Motion)
9	PLCopen	MC_HaltSuperImposed	x	x
10	PLCopen	MC_MoveVelocity	x	x
11	PLCopen	MC_MoveContinuousAb-solute	x	x
12	PLCopen	MC_MoveContinuousRela-tive	x	x
13	PLCopen	MC_PositionProfile	x	x
14	PLCopen	MC_VelocityProfile	x	x
15	PLCopen	MC_AccelerationProfile	x	x
16	PLCopen	MC_SetPosition	x	x
17	PLCopen	MC_SetOverride	x	x
18	PLCopen	MC_ReadParameter	x	x
19	PLCopen	MC_ReadBoolParameter	x	x
20	PLCopen	MC_WriteBoolParameter	x	x
21	PLCopen	MC_WriteParameter	x	x
22	PLCopen	MC_ReadActualPosition	x	x
23	PLCopen	MC_ReadActualVelocity	x	x
24	PLCopen	MC_ReadStatus	x	x
25	PLCopen	MC_ReadAxisError	x	x
26	PLCopen	MC_Reset	x	x
27	PLCopen	MC_CamTableSelect	x	x
28	PLCopen	MC_CamIn	x	x
29	PLCopen	MC_CamOut	x	x
30	PLCopen	MC_GearIn	x	x
31	PLCopen	MC_GearOut	x	x
32	PLCopen	MC_GearInPos	x	x
33	PLCopen	MC_PhasingAbsolute	x	x
34	PLCopen	MC_PhasingRelative	x	x
35	PLCopen	MC_HaltPhasing	-	x
36	PLCopen	MC_LoadControl	-	x
37	PLCopen	MC_LimitLoad	-	x
38	PLCopen	MC_LimitMotion	-	x
39	PLCopen	MC_LoadSuperImposed	-	x
40	PLCopen	MC_LoadProfile	-	x
41	PLCopen	MC_TorqueControl	-	x
42	ABB	MCA_CamInDirect	x	x
43	ABB	MCA_CamInfo	-	x
44	ABB	MCA_Cam_Extra	x	x
45	ABB	MCA_DriveBasedHome	x	x

SNo	Funktion block type	Funktion block name	Motion Library V2 (PS552-MC)	Motion Library V3 (PS5611-Motion)
46	ABB	MCA_GearInDirect M	x	x
47	ABB	CA_Indexing	x	x
48	ABB	MCA_JogAxis	x	x
49	ABB	MCA_MoveByExternalReference	x	x
50	ABB	MCA_MoveVelocityContinuous	x	x
51	ABB	MCA_MoveRelativeOpti	x	x
52	ABB	MCA_Parameter	x	x
53	ABB	MCA_PhasingbyMaster	-	x
54	ABB	MCA_ReadParameterList	x	x
55	ABB	MCA_SetOperatingMode	x	x
56	ABB	MCA_SetPositionContinuous	x	x
57	ABB	MCA_WriteParameterList	x	x
58	ABB	MCA_CamGetInterpolationPosition	-	x
59	ABB	MCA_Home	x	-
60	ABB	MCA_Power	x	-
61	ABB	ECAT_402Parameter-Homing_APP	x	x
62	ABB	ECAT_HomingOnTouchProbe_APP	x	x
63	ABB	ECAT_CiA402_TouchProbe_App	x	x

PLCopen Part 4 –Coordinated Motion is only available for V2 PLC and not yet available for V3 PLC.

1.5.10.5 Examples

Example projects for the libraries can be found in the folder:
\Users\Public\Documents\AutomationBuilder\Examples\PS5609-Log

1.5.11 MQTT client library

1.5.11.1 Structures and enumerations

MQTT_ERROR_ID (Enum)

Parameter	Value	Description
MQTT_ERR_NO_ERROR	0	No error.
MQTT_ERR_CONN_SERVICE_UNAVAIL	16#3001	The Network Connection has been made but the MQTT service is unavailable on the specified port.
MQTT_ERR_COMMUNICATION_TIMEOUT	16#3013	The timeout value for the communication has been exceeded.
MQTT_ERR_REC_PACKET_TOO_LONG	16#3017	Received topic is too long.
MQTT_ERR_PING_NO_ANSWER	16#301A	The MQTT broker did not answer the ping. MQTT client has passed the KeepAlive or MQTT broker is unreachable.
MQTT_ERR_CONN_CLIENT_ID_NOT_ALLOWED	16#301F	The Client identifier is correct UTF-8 but not allowed by the Server.
MQTT_ERR_CONN_REFUSED_PROTOCOL	16#3020	The Server does not support the level of the MQTT protocol requested by the Client.
MQTT_ERR_CONN_REFUSED_CONNECTION	16#3025	Connection refused, maybe the IP address is malformed.
MQTT_ERR_UNSPECIFIED_ERROR	16#302B	Internal library returned an unspecified error.
MQTT_ERR_NETWORK_ERROR	16#302D	General network error.
MQTT_ERR_CONN_AUTH_FAILED	16#3217	Authentication failed: Bad username, password OR client id.
MQTT_ERR_CONN_TLS_HANDSHAKE_FAILED	16#3230	Error on TLS handshake.
MQTT_ERR_CONN_SERVER_CERT_NOT_VALID	16#3231	Server certificate not valid. Check if PLC date has been set correctly.
MQTT_ERR_CONN_SERVER_CERT_NOT_PEM	16#3232	Server certificate format is not formatted as PEM.
MQTT_ERR_CONN_SERVER_CERT_EXPIRED	16#3233	Server certificate has expired.
MQTT_ERR_CONN_CLIENT_CERT_NOT_VALID	16#3234	Client certificate not valid. Check if PLC date has been set correctly.
MQTT_ERR_CONN_CLIENT_CERT_NOT_PEM	16#3235	Client certificate or client key format is not formatted as PEM.
MQTT_ERR_CONN_CLIENT_CERT_EXPIRED	16#3236	Client certificate has expired.

Parameter	Value	Description
MQTT_ERR_INPUT_02_0	16#4020	Function block Input 02 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Parameter Conn of function block was not set. ● MqttConnectWithCertFile (FB): Parameter Conn of function block was not set. ● MqttGetReceivedPacket (FB): Parameter Conn of function block was not set. ● MqttPublish (FB): Parameter Conn of function block was not set. ● MqttSubscribe (FB): Parameter Conn of function block was not set. ● MqttUnsubscribe (FB): Parameter Conn of function block was not set. MqttPing (FB): Parameter Conn of function block was not set.
MQTT_ERR_INPUT_03_0	16#4030	Function block Input 03 error (error case 0), specific error depends on used function block: ● MqttGetReceivedPacket (FB): Pointer payload not initialized. ● MqttPublish (FB): Publish topic name must not contain wildcard characters (+ or #). ● MqttSubscribe (FB): Topic is missing. ● MqttUnsubscribe (FB): Topic is missing.
MQTT_ERR_INPUT_03_1	16#4031	Function block Input 03 error (error case 1), specific error depends on used function block: ● MqttPublish (FB): Payload is not set in MQTT_MESSAGE.
MQTT_ERR_INPUT_04_0	16#4040	Function block Input 04 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Check if Port number has been set correctly (0 is not accepted). ● MqttConnectWithCertFile (FB): Check if Port number has been set correctly (0 is not accepted).
MQTT_ERR_INPUT_06_0	16#4060	Function block Input 06 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertFile (FB): Server certificate file was not found.
MQTT_ERR_INPUT_07_0	16#4070	Function block Input 07 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertFile (FB): Client certificate file was not found.

Parameter	Value	Description
MQTT_ERR_INPUT_08_0	16#4080	Function block Input 08 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertFile (FB): Client key file was not found.
MQTT_ERR_INPUT_12_0	16#4120	Function block Input 12 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Couldn't initialize Last Will message because the topic is not set. ● MqttConnectWithCertFile (FB): Couldn't initialize Last Will message because the payload is not set.
MQTT_ERR_INPUT_12_1	16#4121	Function block Input 12 error (error case 1), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Couldn't initialize Last Will message because the topic is not set. ● MqttConnectWithCertFile (FB): Couldn't initialize Last Will message because the payload is not set.
MQTT_ERR_FATAL_ERROR	16#5FFFF	Fatal error state machine.

MQTT_QOS (Enum)

Parameter	Value	Description
QOS_0	-	Fire and forget (At most once delivered).
QOS_1	-	Simple acknowledgement (At least once delivered).
QOS_2	-	Complex acknowledgement (Exactly once delivered).

MQTT_MESSAGE

This structure is used for messages which can be published or used for LastWill on MqttConnect(FB).

Variable name	Data type	Default value	Description
sTopic	STRING(MQTT_MAX_TOPIC_LEN)	Empty string	Topic where this message belongs to.
pbyPayload	POINTER TO BYTE	0	Payload which should be sent.
dwLen	DWORD	0	Length of the payload.
eQos	MQTT_QOS	QOS_0	Quality of Service level.
xRetainFlag	BOOL	FALSE	True = message must be stored by the server, False = server must not store this message.

MQTT_CONNECTION

Internal data required by the library to operate. This structure allocates memory and it is used to identify the MQTT connection you want to work with

Parameter	Data type	Range
abyConn	Array	MQTT_CLIENT_STRUCT_SIZE
abyTxBuf	Array	MQTT_TX_BUF_SIZE
abyRxBuf	Array	MQTT_RX_BUF_SIZE
abyMsgBuf	Array	MQTT_MSG_BUF_SIZE

1.5.11.2 Global variables

MQTT_CONSTANTS

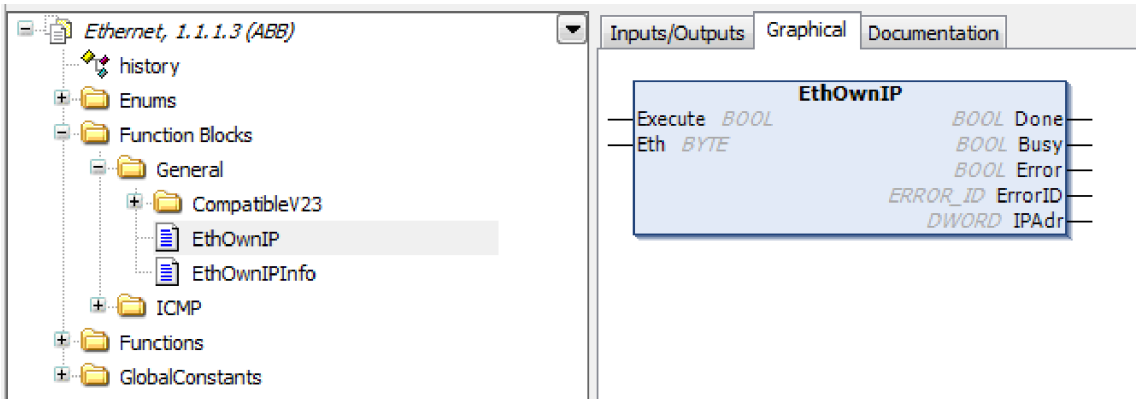
Parameter	Datatype	Value	Description
MQTT_MAX_IP_ADDRESS_LEN	Word	15	Maximum length of the IP address.
MQTT_MAX_PEM_KEY_LEN	Word	2048	Maximum length of the PEM key.
MQTT_MAX_PEM_CERT_LEN	Word	3072	Maximum length of the PEM certificate.
MQTT_MAX_FILE_PATH_LEN	Word	255	Maximum length of the file path to the certificate files.
MQTT_MAX_CLIENT_ID_LEN	Word	250	Maximum length of the client id.
MQTT_MAX_USERNAME_LEN	Word	250	Maximum length of the username.
MQTT_MAX_PASSWORD_LEN	Word	250	Maximum length of the password.
MQTT_MAX_TOPIC_LEN	Word	255	Maximum length of the topic.
MQTT_CLIENT_STRUCT_SIZE	Word	336	Size of the internal connection structure representing the connection state.
MQTT_TX_BUF_SIZE	Word	1024	Size of the internally used output buffer.
MQTT_RX_BUF_SIZE	Word	1024	Size of the internally used input buffer.
MQTT_MSG_BUF_SIZE	Word	2148	Size of the internally used message buffer.

1.5.12 PLCopen libraries

1.5.12.1 Common function block state machine

Most of the V3 function blocks follow the behavior model and style as recommended by PLC Open.

- Clear separation between *“Edge triggered FBs”* (*“Execute”*) or *“Level triggered FBs”* (*“Enable”*)
- Binary status outputs: *“Done”*, *“Busy”*, *“Error”* (exclusive)
- Standardized state machine
- CamelCase naming for function block and all inputs and outputs



Example: Edge_Triggerd_Function_Block_EthOwnIP according to PLCopen

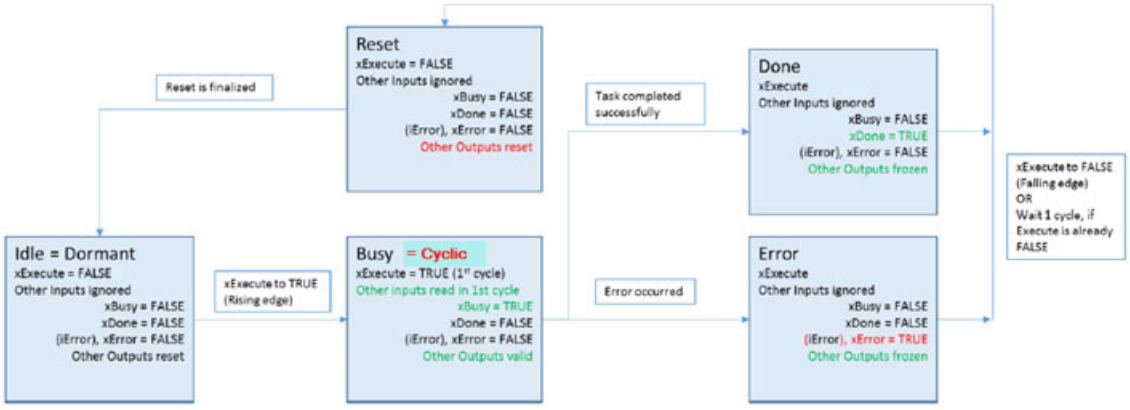
Currently the following *"function block state machines"* are used:

- *"Edge Triggered"* (Input *"Execute"*), for example EthOwnIP
- *"Level Controlled"* (Input *"Enable"*)
- *"Level Controlled Continuous"* (Input *"Enable"*, no *"Done"* output, for example PID)

> *In contrast to AC500 V2 POUs, either "Done" or "Error" is set, not both outputs at the same time in case of an error.*

The state machines are explained in the following chapters.

1.5.12.1.1 Edge triggered (AbbETrig)



After a rising edge at the input *"Execute"* the state goes from *"Dormant"* to *"Busy"*. In the first cycle all inputs are sampled and stored.

When the task is completed successfully the state goes from *"Busy"* to *"Done"*.

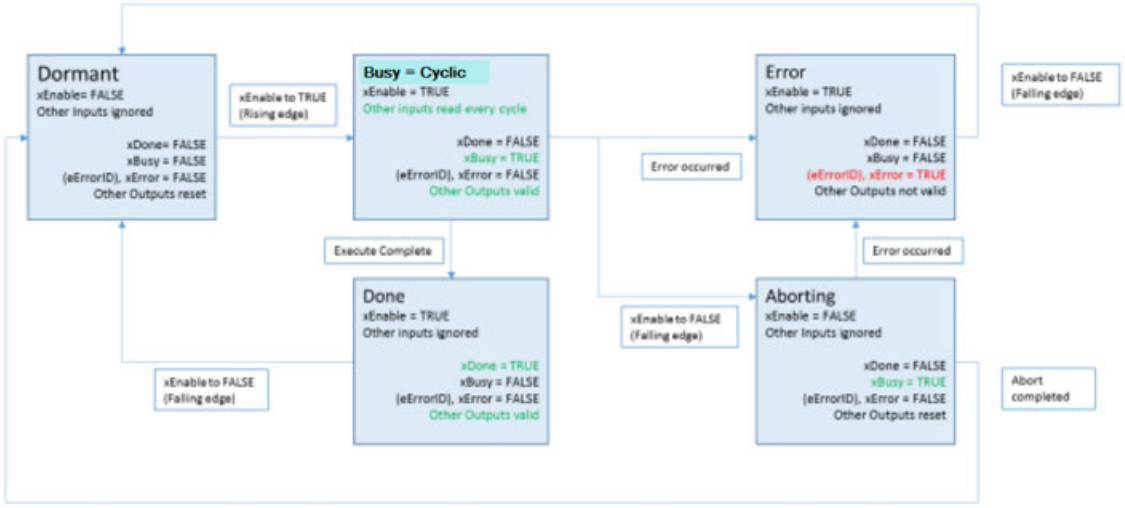
In case of an error the state goes to *"Error"*.

The states *"Done"* or *"Error"* are stable for minimum one cycle and as long as *"Execute"* is *"TRUE"*. With a falling edge of *"Execute"*, the state goes via Reset to *"Dormant"*.

Description of standard inputs and outputs:

- *"Execute"*
  A rising edge starts the operation, the output *"Busy"* goes to *"TRUE"*. In the first cycle all other inputs are read and stored, afterwards they are ignored.
  A falling edge does not stop the operation.
  After *"Done = TRUE"* or *"Error = TRUE"* and *"Execute = FALSE"* all outputs will be reset.
- *"Busy"*
  Operation is running (while outputs *"Done"* and *"Error"* are *"FALSE"*)
- *"Done"*
  Operation is completed without error (while outputs *"Busy"* and *"Error"* are *"FALSE"*).
  This output is *"TRUE"* for at least one cycle or until *"Execute"* is set to *"FALSE"*
- *"Error"*
  Operation is stopped with error (while outputs *"Busy"* and *"Done"* are *"FALSE"*).
  This output is *"TRUE"* for at least one cycle or until *"Execute"* is set to *"FALSE"*.
  The output *"ErrorID"* gives more details about the error.

1.5.12.1.2 Level controlled (AbbLCon)



After a rising edge at the input *"Enable"* the state goes *"Dormant"* to *"Busy"*. All inputs are sampled and considered continuously.

When the task is completed successfully the state goes from *"Busy"* to *"Done"*.

In case of an error the state goes to *"Error"*.

The states *"Done"* or *"Error"* are stable for minimum one cycle and as long as *"Enable is TRUE"*. With a falling edge of *"Enable"*, the state goes via Reset to *"Dormant"*.

The Busy state can be aborted from outside by setting the *"Enable"* input to *"FALSE"*.

After Aborting is done the state goes back to *"Dormant"*.

Description of standard inputs and outputs:

- *"Enable"*
  A rising edge (*"Enable = TRUE"*) starts the operation, the output *"Busy"* goes to *"TRUE"*. All other inputs are read and considered continuously. A falling edge (*"Enable = FALSE"*) aborts the operation.
  During Aborting the Busy is still *"TRUE"*. Afterward all outputs are reset.
- *"Busy"*
  Operation is running (while outputs *"Done"* and *"Error"* are *"FALSE"*)

- *"Done"*
  Operation is completed without error (while outputs *"Busy"* and *"Error"* are *"FALSE"*).
  This output is *"TRUE"* for at least one cycle or until *"Enable"* is set to *"FALSE"*
- *"Error"*
  Operation is stopped with error (while outputs *"Busy"* and *"Done"* are *"FALSE"*).
  This output is *"TRUE"* for at least one cycle or until *"Enable"* is set to *"FALSE"*.
  The output *"ErrorID"* gives more details about the error.

Level controlled continous (AbbLConC)

This state machine is a special case of *"Level Controlled"*. Only difference is that this function block type is never done, for example a PID which never stops.

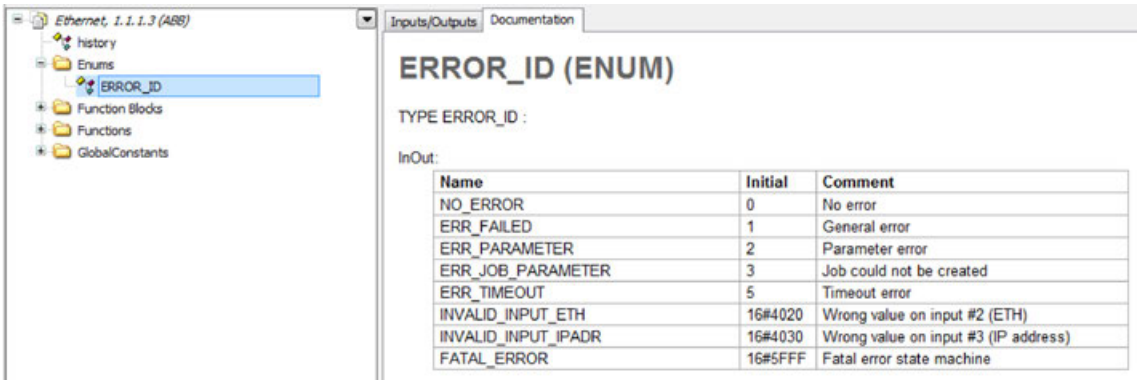
Therefore these function blocks have no *"Done"* output.

Description of standard inputs and outputs:

- *"Enable"*
  A rising edge (*"Enable = TRUE"*) starts the operation, the output *"Busy"* goes to *"TRUE"*. All other inputs are read and considered continuously.
  A falling edge (*"Enable = FALSE"*) aborts the operation.
  During Aborting the *"Busy"* is still *"TRUE"*. Afterward all outputs are reset.
- *"Busy"*
  Operation is running (while output *"Error is FALSE"*)
- *"Error"*
  Operation is stopped with error (while output *"Busy is FALSE"*).
  This output is *"TRUE"* for at least one cycle or until *"Enable"* is set to *"FALSE"*.
  The output *"ErrorID"* gives more details about the error.

1.5.12.1.3 Error_ID

Each library contains an enumeration *"ERROR_ID"*, which is valid for this library but not across all libraries

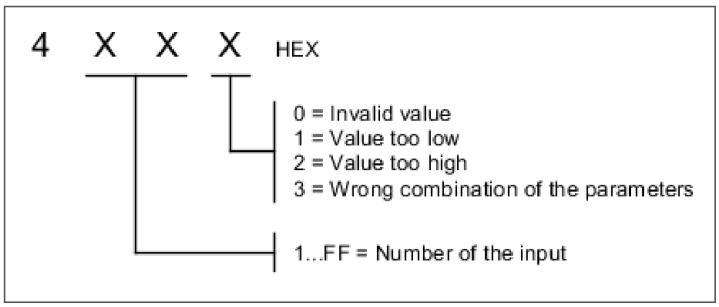


Only the following errors are unique:

- 16#5FFF FATAL_ERROR from state machine
- 16#4000 errors are used for input errors, same scheme like in V2:

4000hex...4FFFhex - Block Input Error

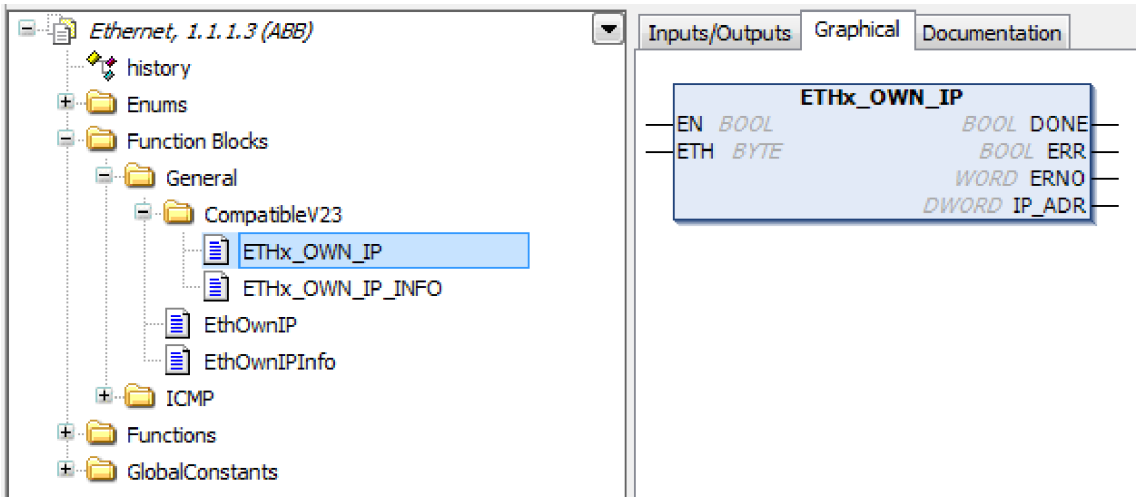
The error 4xxxhex is used in case of detected Function Block input parameter errors. The error is structured as follows:



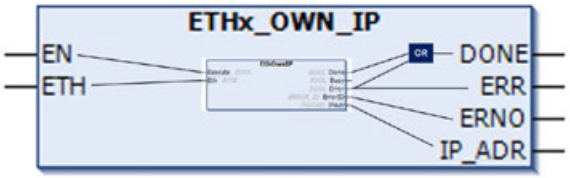
1.5.12.1.4 Compatibility with V2 function blocks

In order to ensure compatibility with V2 applications a lot of ABB function blocks are delivered together with a compatible version in classic style:

Names in CAPITAL letters, input *“EN”* and outputs *“DONE”*, *“ERR”* and *“ERNO”*:



The classic blocks internally use the PLCopen style function blocks. The inputs and outputs are mapped in the following way:

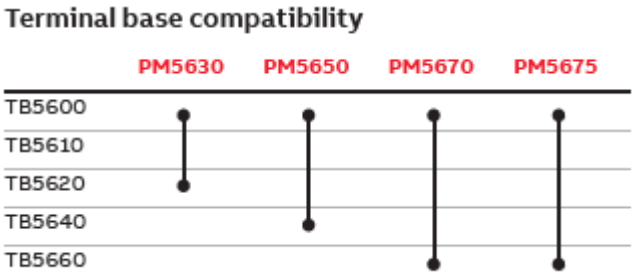


1.6 PLC integration (hardware)

1.6.1 Product overview and comparison

1.6.1.1 Comparison of AC500 V3 terminal bases

With the latest Automation Builder version the following terminal bases are compatible with the AC500 V3 processor modules:



The number of slots that are available on a terminal base for connecting communication modules or AC500-S modules differs within the terminal base range.

Table 398: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot

Processor module	PM5630	PM5650	PM5670	PM5675
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])
Remarks: The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base. [1]) PM567x must have an index ≥ C0.				

Supported devices

The AC500 V3 terminal bases can be equipped with the following supported devices:

Table 399: Comparison: TB56xx

Processor module		PM5630	PM5650	PM5670	PM5675
Max. number of variables allowed for each communication module supported					
	Input variables	4 kB	4 kB	5 kB	5 kB
	Output variables	4 kB	4 kB	5 kB	5 kB
Type of communication module supported					
	CM574-RS/RCOM - serial interface	No	No	No	No
	CM582-DP - PROFIBUS DP V0/V1 slave	No	No	No	No
	CM592-DP - PROFIBUS DP V0/V1 master	[1])	[1])	[1])	[1])
	CM579-ETHCAT - EtherCAT master	x	x	x	x
	CM579-PNIO - PROFINET IO RT controller	x	x	x	x
	CM589-PNIO - PROFINET IO RT device	[1])	[1])	[1])	[1])
	CM589-PNIO-4 - PROFINET IO RT with 4 devices	[1])	[1])	[1])	[1])
	CM597-ETH - Ethernet interface	No	No	No	No
	CM588-CN - CAN, CANopen slave	No	No	No	No
	CM598-CN - CAN, CANopen master	only CAN 2A/2B	only CAN 2A/2B	only CAN 2A/2B	only CAN 2A/2B
Type of AC500-S module supported					
	SM560-S - safety module	x	x	x	x
	SM560-S-FD-1 - safety module with F-Device functionality for 1 PROFIsafe network	[1])	[1])	[1])	[1])
	SM560-S -FD-4 - safety module with F-Device functionality for 1 PROFIsafe network	[1])	[1])	[1])	[1])
Remarks: [1]) in preparation					

Memory size and performance

Table 400: Comparison: PM56xx

Processor module		**PM5630**	**PM5650**	**PM5670**	**PM5675**
Total maximum downloadable application size [1)]		9 MB	84 MB	176 MB	176 MB
	Thereof user program code and data (dynamically allocated)	2 MB	8 MB	32 MB	32 MB
	Thereof user webserver data	7 MB	76 MB	144 MB	144 MB
	Remaining for all other usage (project save, infrastructure...)	30 MB	285 MB	643 MB	643 MB
Buffered (SRAM)		256 kB	256 kB	1.5 MB	1.5 MB
	Thereof VAR retain persistent	128 kB	128 kB	1024 kB	1024 kB
	Thereof %M memory (e.g. Modbus register)	128 kB	128 kB	512 kB	512 kB
Expandable memory		None	None	None	None
Integrated mass storage memory (FLASH)		None	None	None	8 GB
Slot for pluggable memory card		MC502	MC502	MC502	MC502
Processor type		TI ARM Cortex-A9 32-bit-RISC			
Processor speed		300 MHz	600 MHz	1 GHz	1 GHz
Cycle time for 1 instruction (minimum):					
	Binary	Min. 0.02 µs	Min. 0.01 µs	Min. 0.002 µs	Min. 0.002 µs
	Word	Min. 0.02 µs	Min. 0.01 µs	Min. 0.002 µs	Min. 0.002 µs
	Floating point	Min. 0.12 µs	Min. 0.01 µs	Min. 0.002 µs	Min. 0.002 µs
Mathematic co-processor		x	x	x	x
Motion capability					
	No. synchronized axis per 1 ms on EtherCAT CM typically	-	8*	16*	16*
	No. synchronized axis per 2 ms on EtherCAT CM typically	4*	16*	>32	>32
	No. synchronized axis per 4 ms on EtherCAT CM or CANopen onboard typically	8*	>32	>32	>32
	Min. bus cycle time for EtherCAT using external CM579	2 ms	1 ms	0,5 ms	0,5 ms
* in addition: 1 virtual axis					
Max. number of central inputs and outputs (10 exp. modules):					
	Digital inputs	320			
	Digital outputs	320			
	Analog inputs	160			

Processor module		**PM5630**	**PM5650**	**PM5670**	**PM5675**
	Analog outputs	160			
Number of decentralized inputs and outputs		Depends on the used fieldbus			
Data backup		Battery			
Data buffering time at 25 °C		Typ. 3 years			
Battery low indication		via application program			
Real-time clock:					
	With battery backup	x			
	Accuracy	Typ. ±2 s / day at 25 °C			
Program execution:					
	Cyclic	x			
	Time-controlled	x			
	Multitasking	x			
	Minimum cycle time configurable for cyclical task	1 ms	1 ms	0,5 ms	0,5 ms
User program protection by password		x (user management)			
Internal interfaces for communication:					
Serial interface COM1:					
	Physical link	Configurable for RS-232 or RS-485 (9.6 kb/s, 19.2 kb/s, 38.4 kb/s, 57.6 kb/s and 115.2 kb/s)			
	Connection	Pluggable terminal block, spring connection			
	Usage	Serial ASCII communication,Modbus RTU			
CAN interface:					
	Physical link	CAN 2A/2B (from 50 kb/s to 1 Mb/s)			
	Connection	Pluggable terminal block, spring connection			
	Usage	CANopen master communication, CAN 2A/2B, J1939 protocol, CAN sync			
	Max. number of variables allowed				
	Input variables	2 kB	4 kB	5 kB	5 kB
	Output variables	2 kB	4 kB	5 kB	5 kB
Network interface ETH1, ETH2:					
	Usage	Ethernet			
	Physical link	10/100 base-TX, configurable as internal switch or independent Interfaces			
	Connection	2x RJ45 socket, provided on TB56xx-2ETH			
LEDs, LCD display, function keys		RUN / STOP, status, diagnosis, settings			
Number of timers		Unlimited			
Number of counters		Unlimited			

Processor module		PM5630	PM5650	PM5670	PM5675
Programming languages:					
	Structured Text ST	x			
	Instruction list IL	x			
	Function Block Diagram FBD	x			
	Ladder Diagram LD	x			
	Sequential function chart SFC	x			
	Continuous function chart (CFC)	x			

Remarks:

[1]: The values are for information only and cannot be fulfilled altogether. The available resources are limited at the end by the maximal downloadable application size for each CPU.

1.6.1.2 Comparison of features and protocols

Communication and onboard protocols

Table 401: OPC UA server / OPC DA server

Processor module		PM5630	PM5650	PM5670	PM5675
OPC UA server		x	x	x	x
	Number of free tags + additional license for extension [1])	1.000	5.000	30.000	30.000
	Number of connections	10	20	50	50
	Min. sampling rate (limit)	500 ms	100 ms	50 ms	50 ms
OPC DA server AE		x	x	x	x
	Number of connections	8	8	8	8
Remarks: [1]) in preparation					

Table 402: Modbus, Telecontrol

Processor module		PM5630	PM5650	PM5670	PM5675
Modbus TCP client / server		x	x	x	x
	Number of Modbus clients ModMast in parallel on a CPU master (server)	30	50	120	120
	Number of Modbus server in parallel (e.g. for SCADA access)	15	25	50	50
IEC 60870-5-104 telecontrol protocol		x	x	x	x
	Number of free tags + additional license for extension [1])	1.000	5.000	10.000	10.000
	Control station (number of connections)	5	10	20	20

Processor module	PM5630	PM5650	PM5670	PM5675
Sub-station (number of connections)	5	10	20	20
Remarks: [1]) in preparation				

1.6.1.3 Ethernet protocols and ports for AC500 V3 products

Supported as of Automation Builder V 2.1

Description	PM5630 -2ETH	PM5650 -2ETH	PM5670 -2ETH	PM567 5-2ETH	≥ CPU firm-ware
ABB netConfig	x	x	x	x	V3.0.0
Online access with driver 3S TCP/IP BlkDrvTcp	x	x	x	x	V3.0.0
Modbus TCP server	x	x	x	x	V3.0.3
Modbus TCP client with POU ETHx_MOD_MAST	x	x	x	x	V3.0.1
UDP out of user program with library netBaseService.lib	x	x	x	x	V3.0.0
UDP data exchange, Network variables	x	x	x	x	V3.0.0
TCP/IP out of user program with library netBaseService.lib	x	x	x	x	V3.0.0
Web server on PLC with web visualization	x	x	x	x	V3.0.0
NTP/SNTP ((Simple) Network Time Protocol) client with 3S licenced store package SNTPService.package. Library container: SNTPService	x	x	x	x	V3.0.0
IEC60870-5-104 control station incl. 2nd connection and 2nd port	x	x	x	x	V3.0.0
IEC60870-5-104 substation incl. 2nd port	x	x	x	x	V3.0.0
FTP server (See *Chapter 1.6.6.3.5.1 "Configuration of FTP server" on page 3917*)	x	x	x	x	V3.0.0
CODESYS network variables	x	x	x	x	V3.0.0
OPC DA server	x	x	x	x	V3.0.0
OPC UA server	x	x	x	x	V3.0.0
ICMP – ping out of user project with POU *ETHx_ICMP_PING*	x	x	x	x	V3.0.0
DHCP client	x	x	x	x	V3.1.0
NTP/SNTP ((Simple) Network Time Protocol) client system solution (See *Chapter 1.6.6.3.4.2.1 "(S)NTP client configuration" on page 3913*)	x	x	x	x	V3.1.0
NTP/SNTP ((Simple) Network Time Protocol) server system solution (See *Chapter 1.6.6.3.4.2.2 "(S)NTP server configuration" on page 3916*)	x	x	x	x	V3.1.0

Description	PM5630 -2ETH	PM5650 -2ETH	PM5670 -2ETH	PM567 5-2ETH	≥ CPU firm- ware
Maximum number of Input/output allowed variable on Ethernet for the protocol	2 kB /2 kB	4 kB /4 kB	5 kB /5 kB	5 kB /5 kB	V3.4.0
IEC 61850 (MMS server, GOOSE) [2])	x	x	x	x	V3.1.0
EthernetIP Scanner [1, 2])	x	x	x	x	AB 2.4.1/ FW 3.4.1
EthernetIP Adapter [1, 2])	x	x	x	x	AB 2.4.1/ FW 3.4.1
KNX - Building communication [2])	x	x	x	x	V3.2.x
BACnet-BC - Infrastructure communication [2])	x	x	x	x	V3.3.1
HTTPS – secure web server on PLC with CODESYS web visualization (See *Chapter 1.6.6.3.7.3.2 "Secure web server" on page 3922*)	x	x	x	x	V3.1.0
WebVisu for data visualisation on web server HTML5	x	x	x	x	V3.0.0
FTPS – secure FTP (See *Chapter 1.6.6.3.7.3.3 "Secure FTP" on page 3923*)	x	x	x	x	V3.1.0
Secure online access with driver 3S UDP BlkDrvUdp	x	x	x	x	V3.1.0
Secure online access with driver 3S TCP/IP BlkDrvTcp	x	x	x	x	V3.1.0
ICMP – ping out of user project with POU ETHx_ICMP_PING or EthIcmpPing (PLCopen style)	x	x	x	x	V3.1.0
Modbus TCP client (master) with POU ETHx_MOD_MAST or ModTcpMast (PLCopen style)	x	x	x	x	V3.1.0
RTV (Remote Target Visualization)	x	x	x	x	V3.1.0
Remarks: [1]): in preparation [2]): feature is licensed					

1.6.1.3.1 Default open Ethernet ports of PM56xx-2ETH

After startup without a PLC project the PM56xx-2ETH contains the following Ethernet ports and sockets:

Protocol	Port
ABB NetConfig [1])	UDP 24576
Online access with driver 3S UDP BlkDrvUdp (with scan)	UDP 1740
Online access with driver 3S Tcp/Ip BlkDrvTcp (no scan)	TCP 11740
OPC UA server [2])	TCP 4840
Remarks: [1]): The port 24576 for ABB NetConfig protocol can be disabled via PLC configuration by deleting the protocol node from configuration tree of Ethernet interfaces ETH1 and ETH2. [2]): The port 4840 for OPC UA server is closed by default as of SystemFW V3.1.0.	

All other ports are closed by default.

1.6.1.3.2 Overview of protocols, sockets and ports

Protocol	Port	Sockets
ABB netConfig	24576	1 permanent socket per interface
3S gateway client (e.g. CODESYS) to gateway server	1217	1 permanent socket
Online access with driver 3S UDP BlkDrvUdp (with scan)	1740	1 socket per connection + 4 listen
Online access with driver 3S block driver TCP/IP (no scan)	11740	1 socket per connection + 1 listen
Modbus TCP server	502 or configurable	1 socket listen + 1 socket per server connection, number of server connections is configurable in AB
Modbus TCP client with POU *ETHx_MOD_MAST*	Random	1 socket per connection with POU *ETHx_MOD_MAST*
UDP out of user program with library SysLibSockets.lib	1 ... 65535	1 socket per connection
TCP/IP out of user program with library SysLibSockets.lib	1 ... 65535	1 socket per connection
Web server on PLC with web visualization	80	1 listen and 1 per connection
NTP/SNTP client	123	1 permanent socket
IEC60870-5-104 control station	Random	1 per connection
IEC60870-5-104 substation	2404	1 per connection
FTP server (See ↪ *Chapter 1.6.6.3.5.1 "Configuration of FTP server" on page 3917*)	Command port = 21 Data active mode = 20 Data passive mode = random	1 per session, max. 4 allowed
CODESYS network variables	1202	(UDP broadcast)
OPC DA server (default 3S block driver)	UDP = 1740 or TCP/IP =11740	1 socket per connection

Protocol	Port	Sockets
OPC UA server	4840	1 permanent socket
ICMP – ping out of user project with POU ETHx_ICMP_PING DHCP	none	No socket
DHCP	67	1 socket during startup
NTP/SNTP ((Simple) Network Time Protocol) client system solution (See *Chapter 1.6.6.3.4.2.1 "(S)NTP client configuration" on page 3913*)	123	1 permanent socket
NTP/SNTP ((Simple) Network Time Protocol) server system solution (See *Chapter 1.6.6.3.4.2.2 "(S)NTP server configuration" on page 3916*)	123	1 permanent socket
HTTPS – secure web server on PLC with CODESYS web visualization (See *Chapter 1.6.6.3.7.3.2 "Secure web server" on page 3922*)	443	1 listen and 1 per connection
FTPS – secure FTP (See *Chapter 1.6.6.3.7.3.3 "Secure FTP" on page 3923*)	Command port = 21 Data active mode = 20 Data passive mode = random	1 per session, max. 4 allowed
Secure online access with driver 3S UDP BlkDrvUdp	1740	1 socket per connection + 1 listen
Secure online access with driver 3S TCP/IP BlkDrvTcp	11740	1 socket per connection + 1 listen
ICMP – ping out of user project with POU *ETHx_ICMP_PING* or EthIcmpPing (PLCopen style)	None	No socket
Modbus TCP client (master) with POU *ETHx_MOD_MAST* or *ModTcpMast* (PLCopen style)	Random	1 socket per connection with POU *ETHx_MOD_MAST* or *ModTcpMast*

1.6.1.3.3 Limitation of connections per protocol

Protocol	PM5630-2ETH	PM5650-2ETH	PM5670-2ETH	PM5675-2ETH	≥ CPU firmware
Modbus TCP server (e.g. for SCADA access)	30	100	100	100	3.0.3
	40	40	40	40	3.1.0
	15	25	50	50	3.1.3
Modbus TCP client with POU *ETHx_MOD_MAST*	n/a	100	n/a	n/a	3.0.1
	40	40	40	40	3.1.0
	30	50	120	120	3.1.3
Modbus TCP client with POU *ETHx_MOD_MAST* or *ModTcpMast* (PLCopen style)	30	100	100	100	3.1.0
	30	50	120	120	3.1.3

Protocol		PM5630-2ETH	PM5650-2ETH	PM5670-2ETH	PM5675-2ETH	≥ CPU firm-ware
IEC60870-5-104 control station incl. 2nd connection and 2nd port		10 5	10 10	10 20	10 20	3.1.0 3.4.0
IEC60870-5-104 substation incl. 2nd port		10 5	10 10	10 20	10 20	3.1.0 3.4.0
IEC60870-5-104: No. of free tags + additional license for extension [1])		1.000	5.000	10.000	10.000	3.4.0
FTP server		4	4	4	4	3.1.0
Online access with driver 3S UDP BlkDrvUdp		n/a 8	4 8	n/a 8	n/a 8	3.0.0 3.1.0
Online access with driver 3S TCP/IP BlkDrvTcp		n/a 8	4 8	n/a 8	n/a 8	3.0.0 3.1.0
OPC DA server (number of connections)		n/a 8	4 8	n/a 8	n/a 8	3.0.0 3.1.0
OPC UA server (number of connections)		50 10	50 20	50 50	50 50	3.1.0 3.4.0
	No. of free tags + additional license for extension [1])	1.000	5.000	30.000	30.000	3.4.0
	min sampling rate (limit)	500 ms	100 ms	50 ms	50 ms	3.4.0
Secure online access with driver 3S UDP BlkDrvUdp		8	8	8	8	3.1.0
Secure online access with driver 3S TCP/IP BlkDrvTcp		8	8	8	8	3.1.0
FTPS - secure FTP server		4	4	4	4	3.1.0
RTV (Remote Target Visualization)		5	5	5	5	3.1.0
Remarks: [1]): in preparation						

The PLC types PM5630-2ETH, PM5670-2ETH and PM5675-2ETH are available as of SystemFW 3.1.0.

1.6.1.3.4 Ethernet configuration

Default Ethernet configuration

Module	IP Address	Netmask	Comment
PM5xx2-x-ETH	ETH: 192.168.0.10	255.255.255.0	
PM5072-T-2ETH	ETH1: 192.168.0.10 ETH2: 192.168.1.10	255.255.255.0	The Ethernet ports must be configured in different sub networks.
PM56xx-2ETH	ETH1: 192.168.0.10 ETH2: 192.168.1.10	255.255.255.0	The Ethernet ports must be configured in different sub networks.

For changing the default addresses or the description of the function keys CFG see:

↳ *Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675*

↳ *Chapter 1.6.5.1.6.5 “Description of the function keys” on page 3491*.

1.6.1.3.5 Online access

Preferred driver for online access: 3S UDP block driver BlkDrvUdp. This driver allows to scan and select the connected PLC's.

Alternative: 3S TCP/IP block driver. This driver requires at least 2 sockets:

- 1x driver "BlkDrvTcp" on port 11740
- 1x listen on port 11740 if PLC has established online connection

Online access can be established from:

- *Automation Builder command 'Login' Chapter 1.4.1.20.3.6.2 "Command 'Login'" on page 1028*
- *CODESYS OPC DA server*
- *Panel CP600 series*

Each established connection needs one socket. In addition one socket on port 11740 is listening.

1. Startup the PLC.
   ⇨ One socket on port 11740 (listen).
2. Login from Automation Builder via driver "BlKDrvTcp".
   ⇨ 2 sockets on port 11740 (1x online, 1x listen)
3. Additional login out of OPC server with the same driver.
   ⇨ 3 sockets on port 11740 (2x online, 1x listen)
4. Additional connect CP600 via driver "BlkDrvTcp".
   ⇨ 4 sockets on port 11740 (3x online, 1x listen)

1.6.2 PLC introduction

1.6.2.1 Safety instructions

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variants and requirements associated with any particular installation, ABB cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by ABB with respect to use of information, circuits, equipment or software described in this manual. No liability is assumed for the direct or indirect consequences of the improper use, improper application or inadequate maintenance of these devices. In no event will ABB be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

PLC specific safety notices

The product family AC500 control system is designed according to EN 61131-2 IEC 61131-2 standards. Data, different from IEC 61131, are caused by the higher requirements of Maritime Services. Other differences are described in the technical data description of the devices.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

NOTICE!

PLC damage due to operation conditions

Protect the devices from dampness, dirt and damage during transport, storage and operation!

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

Cleaning instruction

Do not use cleaning agent for cleaning the device.

Use a damp cloth instead.

Connection plans and user software must be created so that all technical safety aspects, legal regulations and standards are observed. In practice, possible shortcircuits and breakages must not be able to lead to dangerous situations. The extent of resulting errors must be kept to a minimum.

Do not operate devices outside of the specified, technical data!

Trouble-free functioning cannot be guaranteed outside of the specified data.

NOTICE!

PLC damage due to missing grounding

- Ensure to earth the devices.
- The grounding (switch cabinet grounding, PE) is supplied both by the mains connection (or 24 V supply voltage) and via DIN rail. The DIN rail must be connected to the ground before the device is subjected to any power. The grounding may be removed only if it is certain that no more power is being supplied to the control system.

In the description for the devices (operating manual or AC500 system description), reference is made at several points to grounding, galvanic isolation and EMC measures. One of the EMC measures consists of discharging interference voltages into the grounding via Y-type capacitors. Capacitor discharge currents must basically be able to flow off to the grounding (in this respect, see also VBG 4 and the relevant VDE regulations).

CAUTION!

Do not obstruct the ventilation for cooling!

The ventilation slots on the upper and lower side of the devices must not be covered.

CAUTION!

Run signal and power wiring separately!

Signal and supply lines (power cables) must be laid out so that no malfunctions due to capacitive and inductive interference can occur (EMC).

WARNING!

Labels on or inside the device alert people that dangerous voltage may be present or that surfaces may have dangerous temperatures.

WARNING!

Splaying of strands can cause hazards!

During wiring of terminals with stranded conductors, splaying of strands shall be avoided.

– Ferrules can be used to prevent splaying.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

Information on batteries

CAUTION!

Use only ABB approved lithium battery modules!

At the end of the battery's lifetime, always replace it only with a genuine battery module.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

Environment considerations

Recycle exhausted batteries. Dispose batteries in an environmentally conscious manner, in accordance to local-authority regulations.

Environment and enclosure information

This equipment is intended for use in a Pollution Degree 2 industrial environment, in overvoltage Category II applications (as defined in IEC publication 60664-1), at altitudes up to 2.000 meters without derating.

This equipment is considered Group 1, Class A industrial equipment according to IEC/CISPR Publication 11. Without appropriate precautions, there may be potential difficulties ensuring electromagnetic compatibility in other environments due to conducted as well as radiated disturbance.

This equipment is supplied as "open type" equipment. It must be mounted within an enclosure that is suitably designed for those specific environmental conditions that will be present and appropriately designed to prevent personal injury resulting from accessibility to live parts. The interior of the enclosure must be accessible only by the use of a tool. Subsequent sections of this publication may contain additional information regarding specific enclosure type ratings that are required to comply with certain product safety certifications.

Refer to NEMA Standards publication 250 and IEC publication 60529, as applicable, for explanations of the degrees of protection provided by different types of enclosure. Also see the appropriate sections in this manual.

1.6.2.2 Cyber security

Cyber security disclaimer

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

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

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

Security related deployment guidelines for industrial automation

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

Signed firmware updates

The firmware update files for the AC500 V3 PLC are digitally signed releases by ABB. During the update process, these signatures are validated by a hardware security component in the PLC. This way, the AC500 V3 PLC will only update with valid, authentic firmware, signed by ABB.

Open ports and services

As part of the ABB security concept the AC500 V3 PLC comes with minimal services opened by default. Only the services needed for initial setup and programming are open before any user application is downloaded ↬ *Chapter 1.6.1.3 “Ethernet protocols and ports for AC500 V3 products” on page 2389*.

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

Secure communication

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

Secure shell access for ABB service

The AC500 V3 PLC contains a secure shell service to access core logging data in case of problems which need a deeper analysis. This service is inactive by default, which means that no one can access this privileged shell in the normal operating state.

To activate this service, local access to the PLC is necessary and activation is only valid until the next power cycle of the PLC. Once activated, the service run on TCP port 22. Each PLC also protects the secure shell access by an individual password.

Frequently asked questions

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

1.6.2.2.1 Defense in depth

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

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

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

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

References: *CODESYS Security Whitepaper*

Security zones

IT resources vary in the extent to which they can be trusted. A common security architecture is therefore based on a layered approach that uses zones of trust to provide increasing levels of security according to increasing security needs. Less-trusted zones contain more-trusted zones and connections between the zones are only possible through secure interconnections such as firewallsFig. 93. All resources in the same zone must have the same minimum level of trust. The inner layers, where communication interaction needs to flow freely between nodes, must have the highest level of trust. This is the approach described in the IEC 62443 series of standards.

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

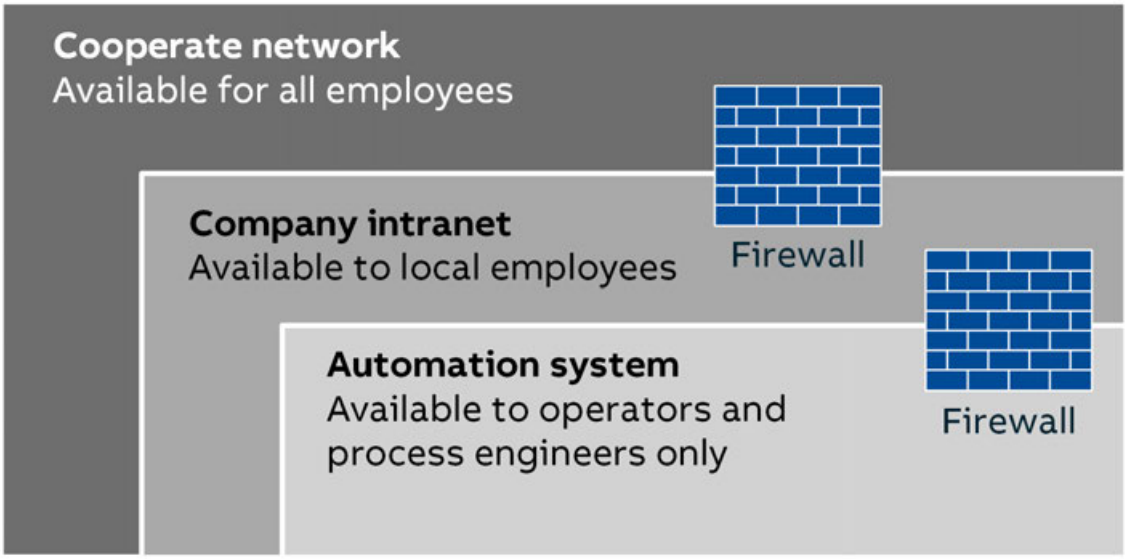


Fig. 93: Security zones

Fig. 93 shows three security zones, but the number of zones does not have to be as many or as few as three. The use of multiple zones allows access between zones of different trust levels to be controlled to protect a trusted resource from attack by a less trusted one.

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

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

References: *Security for Industrial Automation and Control Systems*

1.6.2.2.2 Secure operation

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

A protected environment can be:

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

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

Follow these rules for the protected environment:

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

- Define and maintain plans for incident response, including how to recover from potential disasters.
- Regularly review the organization as well as technical systems and installations with respect to compliance with security policies, procedures and practices.

A protected local control cabinet could look like in figure 94, page 2402. This network is not connected to any external network. Security is primarily a matter of physically protecting the automation system and preventing unauthorized users from accessing the system and from connecting or installing unauthorized hardware and software.

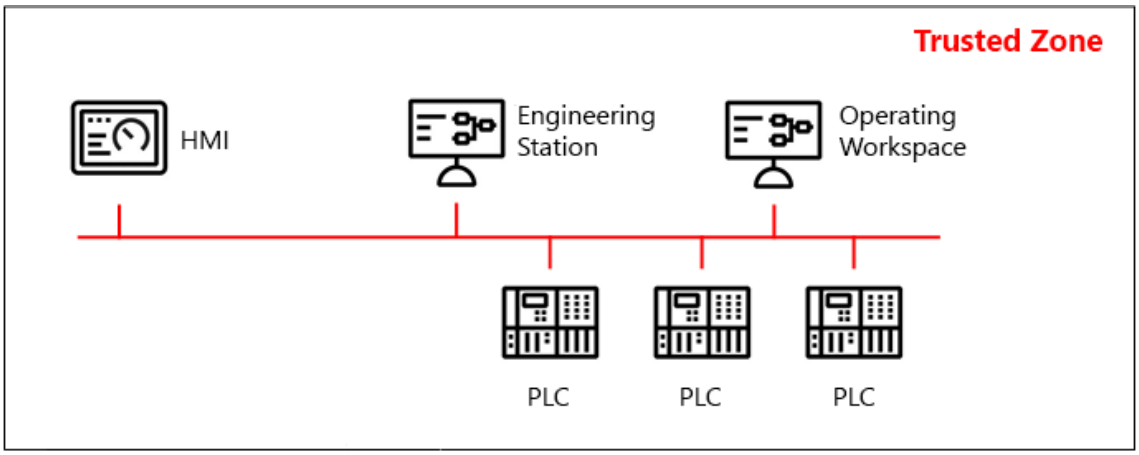


Fig. 94: Isolated automation system

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

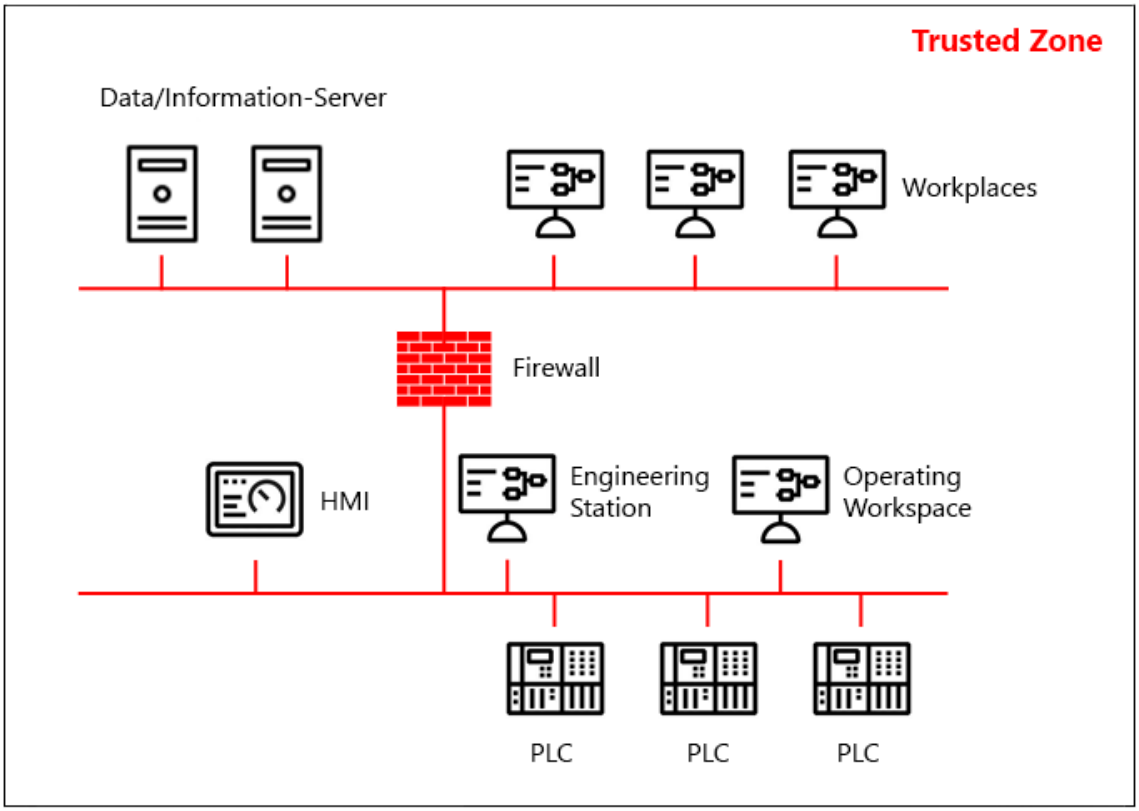


Fig. 95: Plant information network connected to an automation system

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

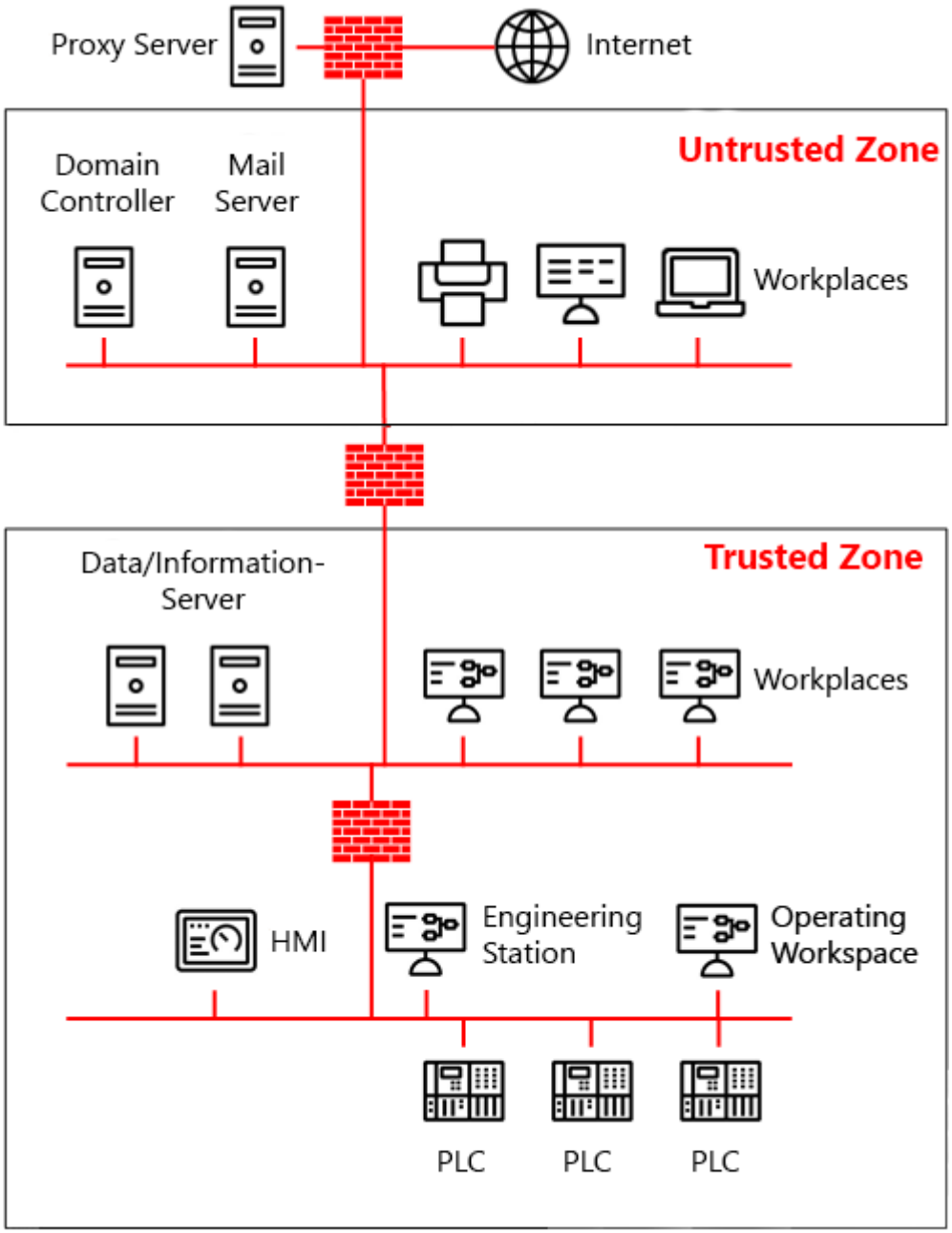


Fig. 96: Automation system and IS network

1.6.2.2.3 Hardening

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

Commissioning phase

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

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

Verification phase

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

Operation phase

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

Decommissioning phase

- Delete all credentials stored in the device like certificates and user data *Chapter 1.6.4.4.6 “Decommissioning” on page 3351*.

References: *Hardening in Wikipedia (2021)*

1.6.2.2.4 Open Ports and Services

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

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

1.6.2.3 License and third party information

Information on Automation Builder licensing and Third Party software can be found in the "About" window of the Automation Builder Installation Manager.

Further information on licensing.

1.6.2.4 Regulations

Appropriate system setup

The following regulations have to be taken into due consideration:

- DIN VDE 0100: "Regulations for the Setting up of Power Installations"
- DIN VDE 0110 Part 1 and Part 2: "The Rating of Creepage Distances and Clearances"
- DIN VDE 0160 and DIN VDE 0660 Part 500: "The Equipment of Power Installations with Electrical Components"

To ensure project success and proper installation of all systems, customers must be familiar and proficient with the following standards and must comply with their directives:

- DIN VDE 0113 Part 1 & Part 200: "Working & Process Machinery"
- DIN VDE 0106 Part 100: "Close proximity to dangerous voltages"
- DIN VDE 0160, DIN VDE 0110 Part 1: "Protection against direct contact"

The user has to guarantee that the devices and the components are mounted following these regulations. For operating the machines and installations, other national and international relevant regulations, concerning prevention of accidents and using technical working means, also have to be met.

AC500 devices are designed according to IEC 1131 Part 2 under overvoltage category II per DIN VDE 0110 Part 2.

For direct connection of AC Category III overvoltages provide protection measures for overvoltage category II according to IEC-Report 664/1980 and DIN VDE 0110 Part 1.

Equivalent standards:

- DIN VDE 0110 Part 1 ↔ IEC 664
- DIN VDE 0113 Part 1 ↔ EN 60204 Part 1
- DIN VDE 0660 Part 500 ↔ EN 60439-1 ↔ IEC 439-1

All rights reserved to change design, size, weight, etc.

Qualified personnel

Both the control system AC500 and other components in the vicinity are operated with dangerous contact voltages. Touching parts, which are under such voltages, can cause grave damage to health.

In order to avoid such risks and the occurrence of material damage, persons involved with the assembly, starting up and servicing must possess pertinent knowledge of the following:

- Automation technology sector
- Dealing with dangerous voltages
- Using standards and regulations, in particular VDE, accident prevention regulations and regulations concerning special ambient conditions (e.g. areas potentially endangered by explosive materials, heavy pollution or corrosive influences).

1.6.2.5 Definitions: PLC system start-up

Cold start

> *The AC500-eCo V3 does not use a battery for buffering the operand areas specified below, hence the “cold start” mode does not exist in this product.*

- A cold start is performed by switching power OFF/ON if no battery is connected.
- All RAM memory modules are checked and erased (see *Chapter 1.4.1.20.3.6.10 “Command 'Reset Cold'” on page 1038*).
- If no user program is stored in the Flash EPROM, the default values (as set on delivery) are applied to the interfaces.
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

Warm start

- A warm start is performed by switching power OFF/ON with a battery connected.
- All RAM memory modules are checked and erased except of the buffered operand areas and the RETAIN variables (see *Chapter 1.4.1.20.3.6.11 “Command 'Reset Warm'” on page 1038*).
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

RUN -> STOP

- RUN -> STOP means pressing the RUN function key on the PLC while the PLC is in run mode (AC500 PLC display "run", AC500-eCo PLC "RUN LED" is ON).
- If a user program is loaded into RAM, execution is stopped.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP", AC500-eCo "RUN LED" changes from ON to OFF.

START -> STOP

- START -> STOP means stopping the execution of the user program in the PLC's RAM using the menu item "Online/Stop" in the programming system.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP".

Reset

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (exception: RETAIN variables) are set to their initialization values.
- Reset is performed using the menu item "Online/Reset" in the programming system or pressing the function key RUN for $\geq$ 5 s in STOP mode.

Reset (cold)

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (also RETAIN variables) are set to their initialization values.
- Reset (cold) is performed using the menu item "Online/Reset (cold)" in the programming system.

Reset (original)

- Resets the controller to its original state (deletion of Flash, SRAM (%M, area, %R area, RETAIN, RETAIN PERSISTENT), Communication Module configurations and user program!).
- Reset (original) is performed using the menu item "Online/Reset (original)" in the programming system.

STOP -> RUN

- STOP -> RUN means short pressing the RUN function key on the PLC while the PLC is in STOP mode (AC500 PLC display "StoP", AC500-eCo "RUN LED" is ON). "RUN LED" is OFF of the toggle switch of an AC500-eCo CPU.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo "RUN LED" changes from OFF to ON.

STOP -> START

- STOP -> START means continuing the execution of the user program in the PLC's RAM using the menu item "Online/Start" in the programming system.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo PLC "RUN LED" changes from OFF to ON.

Download

- Download means loading the complete user program into the PLC's RAM. This process is started by selecting the menu item "Online/Download" in the programming system or after confirming a corresponding system message when switching to online mode (menu item "Online/Login").
- Execution of the user program is stopped.
- In order to store the user program to the Flash memory, the menu item "Online/Create boot project" must be called after downloading the program.
- Variables are set to their initialization values according to the initialization table.
- RETAIN variables can have wrong values as they can be allocated to other memory addresses in the new project!
- A download is forced by the following:
  - changed PLC configuration
  - changed task configuration
  - changed library management
  - changed compile-specific settings (segment sizes)
  - execution of the commands "Project/Clean all" and "Project/Rebuild All".

Online change

- After a project has changed, only these changes are compiled when pressing the key <F11> or calling the menu item "Project/Build". The changed program parts are marked with a blue arrow in the block list.
- The term Online Change means loading the changes made in the user program into the PLC's RAM using the programming system (after confirming a corresponding system message when switching to online mode, menu item "Online/Login").
- Execution of the user program is not stopped. After downloading the program changes, the program is re-organized. During re-organization, no further online change command is allowed. The storage of the user program to the Flash memory using the command "Online/Create boot project" cannot be initiated until re-organization is completed.
- Online Change is not possible after:
  - changes in the PLC configuration
  - changes in the task configuration
  - changes in the library management
  - changed compile-specific settings (segment sizes)
  - performing the commands "Project/Clean all" and "Project/Rebuild All".

Data buffering

- Data buffering, i.e., maintaining data after power ON/OFF, is only possible, if a battery is connected for AC500 CPU and the buffering will take place in FLASH with AC500-eCo V3 CPU. The following data can be buffered completely or in parts:
  - Data in the addressable flag area (%M area)
  - RETAIN variable
  - PERSISTENT variable (number is limited, no structured variables)
  - PERSISTENT area (%R area)
- In order to buffer particular data, the data must be excluded from the initialization process (see *Chapter 1.6.5.1.1 “Handling of remanent variables for AC500 V3 products” on page 3456*).

1.6.2.6 Device lists

1.6.2.6.1 Device list: Terminal bases

Terminal bases for AC500 (Standard):

V3 products

Type	Description
TB5600-2ETH ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5600-2ETH, terminal base AC500, slots: 1 processor module, no communication module, 2 Ethernet RJ45 connector, 1 CAN connector
TB5600-2ETH-XC ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5600-2ETH-XC, terminal base AC500, slots: 1 processor module, no communication module, 2 Ethernet RJ45 connector, 1 CAN connector, XC version
TB5610-2ETH ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5610-2ETH, terminal base AC500, slots: 1 processor module, 1 communication module, 2 Ethernet RJ45 connector, 1 CAN connector
TB5610-2ETH-XC ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5610-2ETH-XC, terminal base AC500, slots: 1 processor module, 1 communication module, 2 Ethernet RJ45 connector, 1 CAN connector, XC version
TB5620-2ETH ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5620-2ETH, terminal base AC500, slots: 1 processor module, 2 communication modules, 2 Ethernet RJ45 connector, 1 CAN connector
TB5620-2ETH-XC ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5620-2ETH-XC, terminal base AC500, slots: 1 processor module, 2 communication modules, 2 Ethernet RJ45 connector, 1 CAN connector, XC version
TB5640-2ETH ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5640-2ETH, terminal base AC500, slots: 1 processor module, 4 communication modules, 2 Ethernet RJ45 connector, 1 CAN connector
TB5640-2ETH-XC ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5640-2ETH-XC, terminal base AC500, slots: 1 processor module, 4 communication modules, 2 Ethernet RJ45 connector, 1 CAN connector, XC version
TB5660-2ETH ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5660-2ETH, terminal base AC500, slots: 1 processor module, 6 communication modules, 2 Ethernet RJ45 connector, 1 CAN connector
TB5660-2ETH-XC ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*	TB5660-2ETH-XC, terminal base AC500, slots: 1 processor module, 6 communication modules, 2 Ethernet RJ45 connector, 1 CAN connector, XC version

1.6.2.6.2 Device list: Processor modules (CPUs)

Processor modules for AC500-eCo

V3 products

Type	Description
PM5012-T-ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5012-T-ETH, processor module, programmable logic controller 1 MB, 6DI/4DO-Transistor, Ethernet, 24 V DC, option slot
PM5012-R-ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5012-R-ETH, processor module, programmable logic controller 1 MB, 6DI/4DO-Relay, Ethernet, 24 V DC, option slot
PM5032-T-ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5032-T-ETH, processor module, programmable logic controller 2 MB, 12DI/8DO-Transistor/2DC, Ethernet, 24 V DC, 2 option slots
PM5032-R-ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5032-R-ETH, processor module, programmable logic controller 2 MB, 12DI/6DO-Relay/2DC, Ethernet, 24 V DC, 2 option slots
PM5052-T-ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5052-T-ETH, processor module, programmable logic controller 4 MB, 12DI/8DO-Transistor/2DC, Ethernet, 24 V DC, 3 option slots
PM5052-R-ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5052-R-ETH, processor module, programmable logic controller 4 MB, 12DI/6DO-Relay/2DC, Ethernet, 24 V DC, 3 option slots
PM5072-T-2ETH ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5072-T-2ETH, processor module, programmable logic controller 8 MB, 12DI/8DO-Transistor/2DC, 2 Ethernet, 24 V DC, 3 option slots
PM5072-T-2ETHW ↳ *Chapter 1.6.3.3.1.1 “PM50xx” on page 2440*	PM5072-T-2ETHW, processor module, programmable logic controller 8 MB, 12DI/8DO-Transistor/2DC, 2 Ethernet, 24 V DC, 3 option slots, wide temperature

Option boards for AC500-eCo V3 processor modules

Type	Description
TA5101-4DI ↳ *Chapter 1.6.3.3.1.2.1 “TA5101-4DI - Option board for digital I/O extension” on page 2478*	TA5101-4DI: AC500-eCo V3, option board for digital I/O extension, 4DI 24 V DC, spring/cable front terminal 3.50 mm pitch
TA5105-4DOT ↳ *Chapter 1.6.3.3.1.2.2 “TA5105-4DOT - Option board for digital I/O extension” on page 2484*	TA5105-4DOT: AC500-eCo V3, option board for digital I/O extension, 4DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch

Type	Description
TA5110-2DI2DOT ↳ *Chapter 1.6.3.3.1.2.3 "TA5110-2DI2DOT - Option board for digital I/O extension" on page 2490*	TA5110-2DI2DOT: AC500-eCo V3, option board for digital I/O extension, 2DI 24 V DC, 2DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch
TA5130-KNXP ↳ *Chapter 1.6.3.3.1.2.4 "TA5130-KNXPB - Option board KNX adress push button" on page 2498*	TA5130-KNXPB: AC500-eCo V3, option board KNX adress push button
TA5131-RTC ↳ *Chapter 1.6.3.3.1.2.5 "TA5131-RTC - Option board for real-time clock" on page 2500*	TA5131-RTC:AC500-eCo V3, real-time clock without battery, option board for AC500-eCo V3 Basic CPU only
TA5141-RS232I ↳ *Chapter 1.6.3.3.1.2.6 "TA5141-RS232I - Option board for COMx serial communication" on page 2502*	TA5141-RS232I: AC500-eCo V3, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
TA5142-RS485I ↳ *Chapter 1.6.3.3.1.2.7 "TA5142-RS485I - Option board for COMx serial communication" on page 2504*	TA5142-RS485I: AC500-eCo V3, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
TA5142-RS485 ↳ *Chapter 1.6.3.3.1.2.8 "TA5142-RS485 - Option board for COMx serial communication" on page 2510*	TA5142-RS485: AC500-eCo V3, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch

Processor modules for AC500 (Standard)

V3 products

Type	Description
PM5630-2ETH ↳ *Chapter 1.6.3.3.2.1 "PM56xx-2ETH for AC500 V3 products" on page 2516*	PM5630-2ETH, processor module, memory 8 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols
PM5630-2ETH-XC ↳ *Chapter 1.6.3.3.2.1 "PM56xx-2ETH for AC500 V3 products" on page 2516*	PM5630-2ETH-XC, processor module, memory 8 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version

Type	Description
PM5650-2ETH ↳ *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*	PM5650-2ETH, processor module, memory 80 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols
PM5650-2ETH-XC ↳ *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*	PM5650-2ETH-XC, processor module, memory 80 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version
PM5670-2ETH ↳ *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*	PM5670-2ETH, processor module, memory 160 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols
PM5670-2ETH-XC ↳ *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*	PM5670-2ETH-XC, processor module, memory 160 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version
PM5675-2ETH ↳ *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*	PM5675-2ETH, processor module, memory 160 MB, 8 GB flash disk, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols
PM5675-2ETH-XC ↳ *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*	PM5675-2ETH-XC, processor module, memory 160 MB, 8 GB flash disk, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version

1.6.2.6.3 Device list: Communication modules

Type	Description
CM579-ETHCAT ↳ *Chapter 1.6.3.4.4.1 “CM579-ETHCAT - EtherCAT master” on page 2539*	CM579-ETHCAT, EtherCAT communication module
CM579-PNIO ↳ *Chapter 1.6.3.4.5.1 “CM579-PNIO - PROFINET IO RT controller” on page 2543*	CM579-PNIO, PROFINET communication module
CM579-PNIO-XC ↳ *Chapter 1.6.3.4.5.1 “CM579-PNIO - PROFINET IO RT controller” on page 2543*	CM579-PNIO-XC, PROFINET communication module, XC version

Type	Description
CM598-CN *Chapter 1.6.3.4.3.1 “CM598-CN - CANopen master” on page 2532*	CM598-CN, communication module, CANopen master
CM598-CN-XC *Chapter 1.6.3.4.3.1 “CM598-CN - CANopen master” on page 2532*	CM598-CN-XC, communication module, CANopen master, XC version

1.6.2.6.4 Device list: Terminal units

Type	Description
TU507-ETH *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*	TU507-ETH, Ethernet terminal unit, 24 V DC, screw terminals
TU508-ETH *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*	TU508-ETH, Ethernet terminal unit, 24 V DC, spring terminals
TU508-ETH-XC *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*	TU508-ETH-XC, Ethernet terminal unit, 24 V DC, spring terminals, XC version
TU515 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU515, I/O terminal unit, 24 V DC, screw terminals
TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU516, I/O terminal unit, 24 V DC, spring terminals
TU516-XC *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU516-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
TU516-H *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU516-H, I/O terminal unit, hot swap, 24 V DC, spring terminals

Type	Description
TU516-H-XC ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU516-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version
TU517 ↳ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559*	TU517, terminal unit for communication interface modules, 24 V DC, screw terminals
TU518 ↳ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559*	TU518, terminal unit for communication interface modules, 24 V DC, spring terminals
TU518-XC ↳ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559*	TU518-XC, terminal unit for communication interface modules, 24 V DC, spring terminals, XC version
TU531 ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*	TU531, I/O terminal unit, 230 V AC, relays, screw terminals
TU532 ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*	TU532, I/O terminal unit, 230 V AC, relays, spring terminals
TU532-XC ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*	TU532-XC, I/O terminal unit, 230 V AC, relays, spring terminals, XC version
TU532-H ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*	TU532-H, I/O terminal unit, hot swap, 230 V AC, relays, spring terminals
TU532-H-XC ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*	TU532-H-XC, I/O terminal unit, hot swap, 230 V AC, relays, spring terminals, XC version
TU541 ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU541, I/O terminal unit, 24 V DC, screw terminals
TU542 ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU542, I/O terminal unit, 24 V DC, spring terminals

Type	Description
TU542-XC *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
TU542-H *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU542-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
TU542-H-XC *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*	TU542-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version

1.6.2.6.5 Device list: S500-eCo I/O modules

Type	Description
AI561 *Chapter 1.6.3.6.2.1.1 “AI561 - Analog input module” on page 2776*	AI561, analog input module, 4 AI, U/I
AI562 *Chapter 1.6.3.6.2.1.2 “AI562 - Analog input module” on page 2787*	AI562, analog input module, 2 AI, RTD
AI563 *Chapter 1.6.3.6.2.1.3 “AI563 - Analog input module” on page 2798*	AI563, analog input module, 4 AI, thermocouple
AO561 *Chapter 1.6.3.6.2.1.4 “AO561 - Analog output module” on page 2810*	AO561, analog output module, 2 AO, U/I
AX561 *Chapter 1.6.3.6.2.1.5 “AX561 - Analog input/output module” on page 2819*	AX561, analog input/output module, 4 AI, 2AO, U/I
DC561 *Chapter 1.6.3.6.1.1.1 “DC561 - Digital input/output module” on page 2569*	DC561, digital input/output module, 16 configurable inputs/outputs, transistor output, interfast connection
DC562 *Chapter 1.6.3.6.1.1.2 “DC562 - Digital input/output module” on page 2577*	DC562, digital input/output module, 16 configurable inputs/outputs
DI561 *Chapter 1.6.3.6.1.1.3 “DI561 - Digital input module” on page 2588*	DI561, digital input module, 8 DI, 24 V DC

Type	Description
DI562 ↳ *Chapter 1.6.3.6.1.1.4 “DI562 - Digital input module” on page 2594*	DI562, digital input module, 16 DI, 24 V DC
DI571 ↳ *Chapter 1.6.3.6.1.1.5 “DI571 - Digital input module” on page 2603*	DI571, digital input module, 8 DI, 120 V AC...240 V AC
DI572 ↳ *Chapter 1.6.3.6.1.1.6 “DI572 - Digital input module” on page 2611*	DI572, digital input module, 16 DI, 100 V AC...240 V AC
DO561 ↳ *Chapter 1.6.3.6.1.1.7 “DO561 - Digital output module” on page 2620*	DO561, digital output module, 8 DO, transistor output
DO562 ↳ *Chapter 1.6.3.6.1.1.8 “DO562 - Digital output module” on page 2629*	DO562, digital output module, 16 DO, transistor output
DO571 ↳ *Chapter 1.6.3.6.1.1.9 “DO571 - Digital output module” on page 2638*	DO571, digital output module, 8 DO, relay output
DO572 ↳ *Chapter 1.6.3.6.1.1.10 “DO572 - Digital output module” on page 2648*	DO572, digital output module, 8 DO, triac output
DO573 ↳ *Chapter 1.6.3.6.1.1.11 “DO573 - Digital output module” on page 2658*	DO573, digital output module, 16 DO, relay output
DX561 ↳ *Chapter 1.6.3.6.1.1.12 “DX561 - Digital input/output module” on page 2670*	DX561, digital input/output module, 8 DI 24 V DC, 8 DO 24 V DC, transistor output
DX571 ↳ *Chapter 1.6.3.6.1.1.13 “DX571 - Digital input/output module” on page 2682*	DX571, digital input/output module, 8 DI 24 V DC, 8 DO, relay output

1.6.2.6.6 Device list: S500 I/O modules

Type	Description
AI523 ↳ *Chapter 1.6.3.6.2.2.2 “AI523 - Analog input module” on page 2858*	AI523, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires
AI523-XC ↳ *Chapter 1.6.3.6.2.2.2 “AI523 - Analog input module” on page 2858*	AI523-XC, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires, XC version

Type	Description
AI531 ↳ *Chapter 1.6.3.6.2.2.3 “AI531 - Analog input module” on page 2880*	AI531, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires
AI531-XC ↳ *Chapter 1.6.3.6.2.2.3 “AI531 - Analog input module” on page 2880*	AI531-XC, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires, XC version
AO523 ↳ *Chapter 1.6.3.6.2.2.4 “AO523 - Analog output module” on page 2912*	AO523, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires
AO523-XC ↳ *Chapter 1.6.3.6.2.2.4 “AO523 - Analog output module” on page 2912*	AO523-XC, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires, XC version
AX521 ↳ *Chapter 1.6.3.6.2.2.5 “AX521 - Analog input/output module” on page 2927*	AX521, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires
AX521-XC ↳ *Chapter 1.6.3.6.2.2.5 “AX521 - Analog input/output module” on page 2927*	AX521-XC, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version
AX522 ↳ *Chapter 1.6.3.6.2.2.6 “AX522 - Analog input/output module” on page 2950*	AX522, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires
AX522-XC ↳ *Chapter 1.6.3.6.2.2.6 “AX522 - Analog input/output module” on page 2950*	AX522-XC, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version
DA501 ↳ *Chapter 1.6.3.6.3.1.1 “DA501 - Digital/Analog input/output module” on page 2975*	DA501, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO
DA501-XC ↳ *Chapter 1.6.3.6.3.1.1 “DA501 - Digital/Analog input/output module” on page 2975*	DA501-XC, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO, XC version
DA502 ↳ *Chapter 1.6.3.6.3.1.2 “DA502 - Digital/Analog input/output module” on page 3010*	DA502, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO
DA502-XC ↳ *Chapter 1.6.3.6.3.1.2 “DA502 - Digital/Analog input/output module” on page 3010*	DA502-XC, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO, XC version
DC522 ↳ *Chapter 1.6.3.6.1.2.1 “DC522 - Digital input/output module” on page 2696*	DC522, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires

Type	Description
DC522-XC *Chapter 1.6.3.6.1.2.1 “DC522 - Digital input/output module” on page 2696*	DC522-XC, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires, XC version
DC523 *Chapter 1.6.3.6.1.2.2 “DC523 - Digital input/output module” on page 2706*	DC523, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire
DC523-XC *Chapter 1.6.3.6.1.2.2 “DC523 - Digital input/output module” on page 2706*	DC523-XC, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire, XC version
DC532 *Chapter 1.6.3.6.1.2.3 “DC532 - Digital input/output module” on page 2717*	DC532, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire
DC532-XC *Chapter 1.6.3.6.1.2.3 “DC532 - Digital input/output module” on page 2717*	DC532-XC, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire, XC version
DI524 *Chapter 1.6.3.6.1.2.4 “DI524 - Digital input module” on page 2729*	DI524, digital input module, 32 DI, 24 V DC, 1-wire
DI524-XC *Chapter 1.6.3.6.1.2.4 “DI524 - Digital input module” on page 2729*	DI524-XC, digital input module, 32 DI, 24 V DC, 1-wire, XC version
DO524 *Chapter 1.6.3.6.1.2.5 “DO524 - Digital output module” on page 2737*	DO524, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire
DO524-XC *Chapter 1.6.3.6.1.2.5 “DO524 - Digital output module” on page 2737*	DO524-XC, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire, XC version
DO526 *Chapter 1.6.3.6.1.2.6 “DO526 - Digital output module” on page 2745*	DO526, digital output module, 8 DO, 24 V DC / 2 A, 1-wire
DO526-XC *Chapter 1.6.3.6.1.2.6 “DO526 - Digital output module” on page 2745*	DO526-XC, digital output module, 8 DO, 24 V DC / 2 A, 1-wire, XC version
DX522 *Chapter 1.6.3.6.1.2.7 “DX522 - Digital input/output module” on page 2754*	DX522, digital input/output module, 8 DI, 24 V DC, 8 DO relays

Type	Description
DX522-XC *Chapter 1.6.3.6.1.2.7 “DX522 - Digital input/output module” on page 2754*	DX522-XC, digital input/output module, 8 DI, 24 V DC, 8 DO relays, XC version
DX531 *Chapter 1.6.3.6.1.2.8 “DX531 - Digital input/output module” on page 2766*	DX531, digital input/output module, 8 DI, 230 V AC, 4 DO relay, 2-wires

1.6.2.6.7 Device list: Communication interface modules

Table 403: CANopen

Type	Description
CI581-CN *Chapter 1.6.3.7.2.2 “CI581-CN” on page 3046*	CI581-CN, CANopen communication interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI581-CN-XC *Chapter 1.6.3.7.2.2 “CI581-CN” on page 3046*	CI581-CN-XC, CANopen communication interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version
CI582-CN *Chapter 1.6.3.7.2.3 “CI582-CN” on page 3084*	CI582-CN, CANopen communication interface module, 8 DI, 8 DO and 8 DC
CI582-CN-XC *Chapter 1.6.3.7.2.3 “CI582-CN” on page 3084*	CI582-CN-XC, CANopen communication interface module, 8 DI, 8 DO and 8 DC, XC version

Table 404: EtherCAT

Type	Description
CI511-ETHCAT *Chapter 1.6.3.7.3.1 “CI511-ETHCAT” on page 3106*	CI511-ETHCAT, EtherCAT communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI512-ETHCAT *Chapter 1.6.3.7.3.2 “CI512-ETHCAT” on page 3138*	CI512-ETHCAT, EtherCAT communucation interface module, 8 DI, 8 DO and 8 DC

Table 405: Modbus

Type	Description
CI521-MODTCP *Chapter 1.6.3.7.4.1 “CI521-MODTCP” on page 3156*	CI521-MODTCP, Modbus TCP communucation interface module, 4 AI, 2 AO, 8 DI and 8 DO
CI521-MODTCP-XC *Chapter 1.6.3.7.4.1 “CI521-MODTCP” on page 3156*	CI521-MODTCP-XC, Modbus TCP communucation interface module, 4 AI, 2 AO, 8 DI and 8 DO, XC version

Type	Description
CI522-MODTCP ↳ *Chapter 1.6.3.7.4.2 “CI522-MODTCP” on page 3196*	CI522-MODTCP, Modbus TCP communucation interface module, 8 DC, 8 DI and 8 DO
CI522-MODTCP-XC ↳ *Chapter 1.6.3.7.4.2 “CI522-MODTCP” on page 3196*	CI522-MODTCP-XC, Modbus TCP communucation interface module, 8 DC, 8 DI and 8 DO, XC version

Table 406: PROFINET

Type	Description
CI501-PNIO (V3) ↳ *Chapter 1.6.3.7.5.2 “CI501-PNIO” on page 3224*	CI501-PNIO (V3), PROFINET communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI501-PNIO-XC (V3) ↳ *Chapter 1.6.3.7.5.2 “CI501-PNIO” on page 3224*	CI501-PNIO-XC (V3), PROFINET communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version
CI502-PNIO ↳ *Chapter 1.6.3.7.5.3 “CI502-PNIO” on page 3263*	CI502-PNIO, PROFINET communucation interface module, 8 DI, 8 DO and 8 DC
CI502-PNIO-XC ↳ *Chapter 1.6.3.7.5.3 “CI502-PNIO” on page 3263*	CI502-PNIO-XC, PROFINET communucation interface module, 8 DI, 8 DO and 8 DC, XC version

1.6.2.6.8 Device list: Accessories

Type	Description
Automation Builder	DM-TOOL, Automation Builder software suite, programming software (multilanguage) *www.abb.com/automationbuilder*
MC502 ↳ *Chapter 1.6.3.8.2.1 “MC502 - Memory card” on page 3311*	MC502, memory card
MC5102 ↳ *Chapter 1.6.3.8.1.1 “MC5102 - Micro memory card with micro memory card adapter” on page 3288*	MC5102, micro memory card with micro memory card adapter
MC5141 ↳ *Chapter 1.6.3.8.2.3 “MC5141 - Memory card” on page 3320*	MC5141, memory card
TA521 ↳ *Chapter 1.6.3.8.2.4 “TA521 - Battery” on page 3324*	TA521, lithium battery

Type	Description
TA523 *Chapter 1.6.3.8.3.1 “TA523 - Pluggable label mounting” on page 3329*	TA523, pluggable label mounting (10 pcs)
TA524 *Chapter 1.6.3.8.2.5 “TA524 - Dummy communication module” on page 3328*	TA524, dummy communication module
TA525 *Chapter 1.6.3.8.3.2 “TA525 - Plastic labels” on page 3331*	TA525, set of 10 white plastic markers
TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*	TA526, wall mounting accessory, 10 pcs
TA535 *Chapter 1.6.3.8.3.4 “TA535 - Protective caps for XC devices” on page 3333*	TA535, protective caps for XC devices
TA543 *Chapter 1.6.4.5.5.5 “TA543 - Screw mounting accessory” on page 3396*	TA543, screw mounting accessory for AC500-eCo V3 processor modules PM50xx without DIN rail
TA566 *“Mounting I/O modules on a metal plate” on page 3367*	TA566, wall mounting accessory for S500-eCo I/O modules without DIN rail
TA5400-SIM *Chapter 1.6.3.8.1.4 “TA5400-SIM - Input simulator” on page 3307*	TA5400-SIM, input simulator for PM50xx

1.6.2.7 PLC system description

1.6.2.7.1 AC500 product family

AC500 programmable logic controllers (PLCs)

The AC500 (Standard), AC500-eCo, AC500-S and AC500-XC scalable PLC ranges provide solutions for small, middle and high-end applications. Our AC500 platform offers different performance levels and is the ideal choice for high availability, extreme environments or safety solutions. Our AC500 PLC platform offers interoperability and compatibility in hardware and software from compact PLCs up to high-end and safety PLCs.

Due to the flexible combinations of AC500 devices and components, AC500 PLCs can be used for controlling a wide variety of applications to fulfill your automation needs.

Features of AC500 PLCs

- Scalable and consistently expandable system
- Different performance classes of processor modules (CPUs) available
- Several field busses available
- Parallel connection to several field busses which can be combined arbitrarily

The AC500 product family consists of the product groups:

- AC500 (standard):
  AC500 standard PLCs offer a wide range of performance levels and scalability. The PLCs are highly capable of communication and extension for flexible application.
- AC500-eCo:
  AC500-eCo PLCs are cost-effective, high-performance compact PLCs that offer total interoperability with the core AC500 range and provide battery-free data buffering. All I/O modules can be freely connected in a simple, stable and reliable manner.
- AC500-S:
  AC500-S PLCs are designed for safety applications involved in factory, process or machinery automation area.
- AC500-XC:
  AC500 (standard) and AC500-S provide devices with -XC extension as a product variant. These variants operate according to their product group and can, in addition, be operated under extreme conditions. AC500-XC PLCs can be used at high altitudes, extended operating temperature and in humid condition. Further, the devices provide immunity to vibration and hazardous gases. The AC500-XC series is consistent with standard devices in the overall dimensions, control function and software compatibility. *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*.

The AC500 product family is characterized by functional modularity. As the complete AC500 product family shares the same hardware platform and programming software tool, the devices of the AC500 product groups can be flexibly combined.

S500 devices represent the I/O modules of the product group AC500 (standard), whereas S500-eCo devices represent the I/O modules of the product group AC500-eCo. Both S500 and S500-eCo devices can be combined with devices of the AC500 product family in a flexible way.

Extensions in the product name

AC500 devices support different protocols and technologies (e.g. Ethernet, PROFIBUS etc.) in variable number. AC500 devices with onboard interfaces for support of a certain protocol or technology can be identified easily by the extension in the product name of the AC500 device. For example the AC500 Communication Module PM592**-DP** provides onboard support for PROFIBUS DP, the AC500 processor module PM595**-4ETH** provides onboard support for four provided Ethernet interfaces.

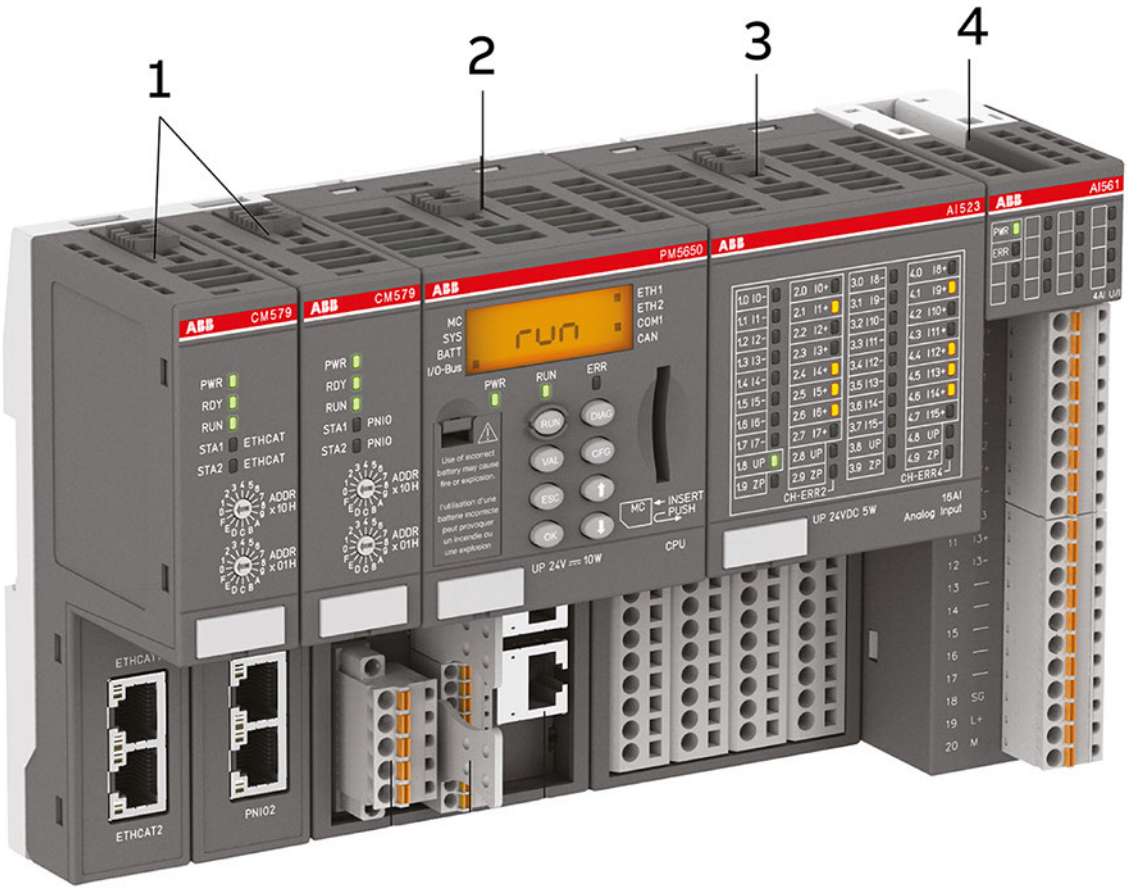
Further extensions in AC500 device names:

- -ETH: Ethernet
- -ARC: ARCNET
- -DP: PROFIBUS DP
- -CAN: CAN/CANopen
- -ETHCAT: EtherCAT
- -PNIO: PROFINET
- -RCOM: RCOM/RCOM+
- -RS: Serial interface

1.6.2.7.2 AC500/S500 system structure

The AC500 product family provides a variety of modules and pluggable components for expanding the capabilities of the CPU with additional I/Os or other communication protocols. Depending on the features and functions of the processor module (CPU) compatible components can be added to a complete AC500 PLC system.

Example of an AC500 PLC system:



1 Plug-in communication module
2 Processor module
3 Plug-in I/O module
4 Plug-in function module (AC500-eCo)
- Plug-in communication module (AC500-S), not displayed
- Plug-in I/O module (AC500-S), not displayed

Centralized I/O extension

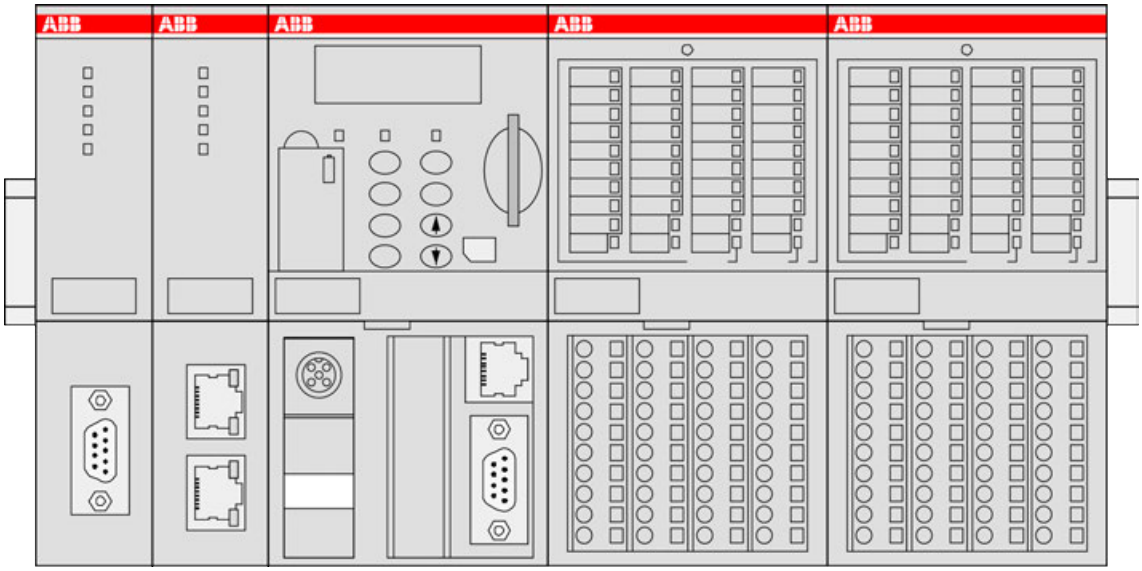


Fig. 97: S500 I/O modules directly connected to a processor module

Decentralized I/O extension

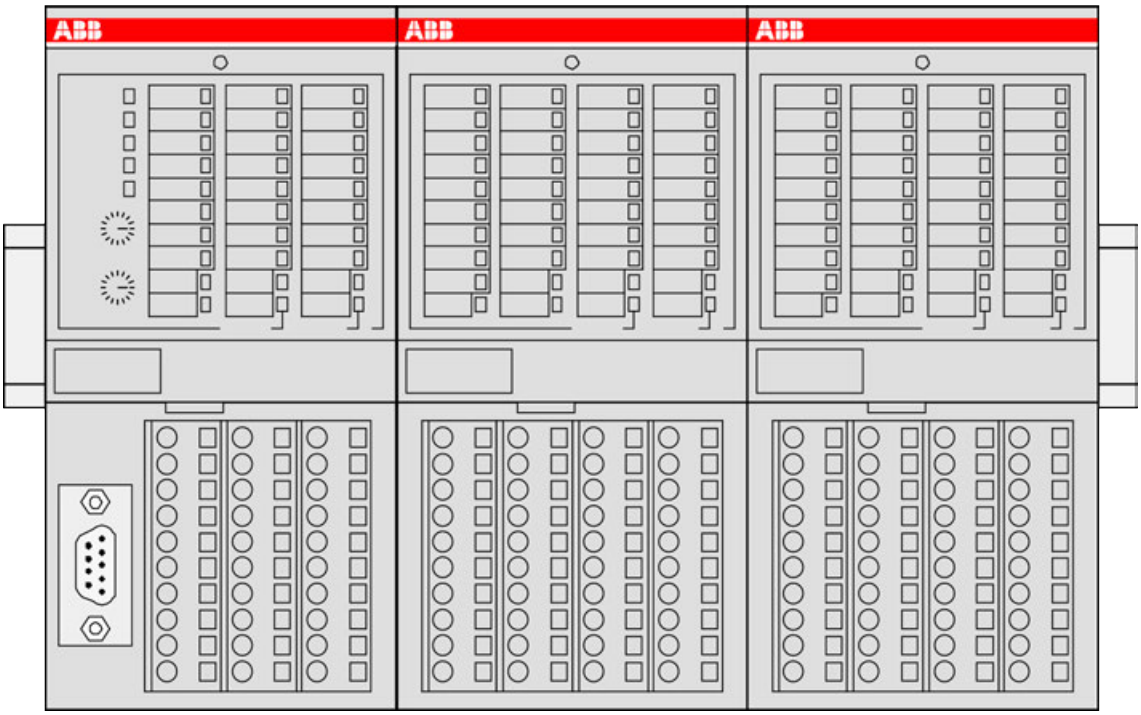
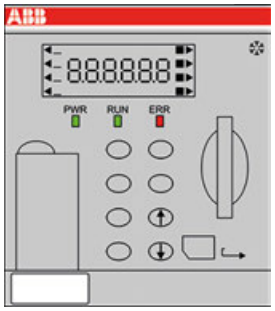


Fig. 98: S500 I/O modules connected via communication interface module

1.6.2.7.3 AC500/S500: Short description hardware

Processor modules



AC500 processor modules contains the CPU with the core component of the PLC. The CPU is connected with the user memory, input and output module, communication port and other units via system bus and performs tasks by means of system programs preset in the system memory. The CPU adopts the function preset by the system program to command the PLC for operation.

Its functions include:

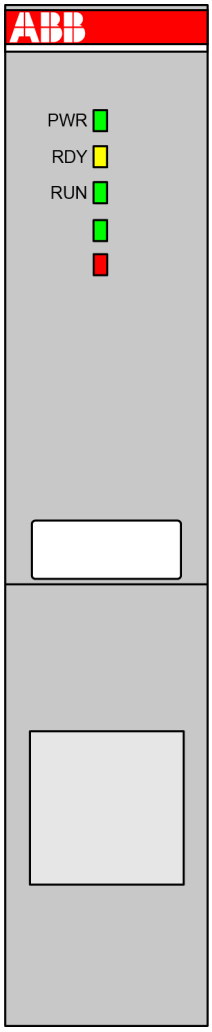
- To receive user program and data entered
- To diagnose work faults of the power supply and PLC circuit as well as syntax error in programming
- To receive the state or data of the site via the input interface and save it into the shadow register or data register
- To read the user program in the memory one by one and execute it after interpretation
- To update the state of related flag bits and output shadow register contents according to execution results and realize output control by means of output unit.

Processor modules are available in different performance classes. Only one processor module is required for a valid system architecture.

There are different types of processor module available that differ in the features and functions they provide, e.g. performance, LED display etc.

If required, processor modules are also available with an integrated Ethernet communication module (TCP/IP).

Communication modules



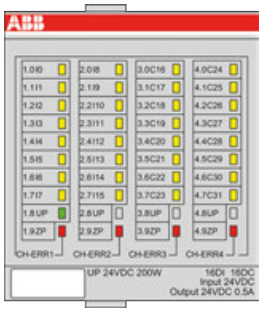
AC500 communication modules are required for

- a connection to standard field bus systems and
- for integration into existing networks.

AC500 communication modules

- enable communication on different field buses.
- are mounted on the left side of the processor module on the same terminal base.
- are directly powered via the internal communication module bus of the terminal base. A separate voltage source is not required.

I/O modules



The I/O modules are the input / output unit which connects the PLC with the industrial production site. The PLC can detect controlled object data via the input interface and the data is taken as the basis for PLC control on the controlled object. In addition, the PLC sends processing results via the output interface to the controlled object to realize the control purpose.

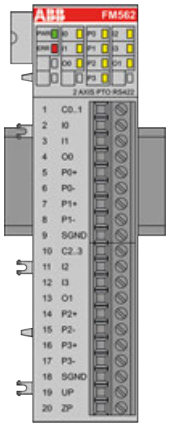
External input equipment and output equipment need various signal levels while the information processed by the CPU in the PLC only can be the standard level. In order to realize such conversion, the I/O interface generally shall perform optical isolation and filtering to improve interference immunity of the PLC. In addition, the I/O interface generally can indicate the working state to facilitate maintenance.

The PLC provides multiple I/O interface for operation level and drive capability to users for selection such as digital input, digital output, analog input, analog output, etc. I/O interfaces of the PLC have the number of input / output signals taken as the number of PLC I/O points. The number of I/O points is an important basis for PLC selection. If the system is insufficient in the I/O points, it can be expanded via the I/O extension interface of the PLC.

The I/O modules for digital and/or analog inputs and outputs are available in different versions and allow flexible use thanks to configurable channels.

The modules can be simply plugged onto a terminal unit for a centralized I/O extension or for a decentralized I/O extension via communication interface modules.

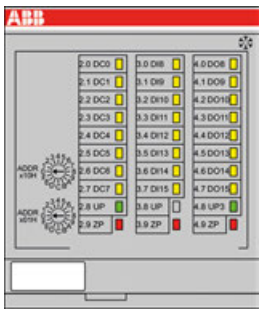
Function modules



Function modules extend the PLC system to perform special task control. Those modules often provide independent components such as a CPU, system programs, memory and interfaces connected with the PLC system bus.

It is connected with the PLC via the I/O bus to exchange data and independently work under cooperative management of the PLC.

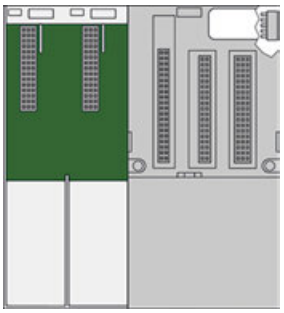
Communication interface modules



Communication interface modules enable a decentralized I/O station. It contains embedded digital I/Os and a fieldbus interface.

Communication interface modules act as I/O slave devices within a master-slave-arrangement.

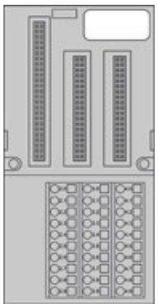
Terminal bases



On a terminal base the processor module and the communication modules are plugged.

> *For AC500-eCo processor modules and special AC500 (Standard) processor modules the terminal base cannot be removed.*

Terminal units



On a terminal unit the I/O modules are plugged.

Terminal units enable simple prewiring without electronics and are available for 24 V DC and 230 V AC, optionally for spring or screw-type terminals.

Memory

In the PLC, the memory is mainly used for saving system programs, user programs and work data. The following memory types can be distinguished:

- Volatile memory:
  All saved data will be lost after power failure of the memory but the memory can provide high access rate and unlimited programming cycles. Common volatile memories mainly include SRAM and DRAM (including common memories such as SDRAM).
- Nonvolatile memory:
  All saved data will not be lost after power failure of the memory, but the memory is subject to low read-write rate and limited rewrite cycles. Common nonvolatile memories mainly include NORflash, NANDflash, EEPROM, memory card, etc.

AC500 PLCs store all user programs in the nonvolatile memory to get protected from power failure. The programs are exported to the volatile memory under operation of the PLC to ensure high-speed and efficient operation. If user program debugging is finished, the programs can be fixed in the nonvolatile memory when they need no change. The work data is subject to frequent change and access in the PLC operation. It is saved in the volatile memory to meet the requirements for random access.

The work data memory of PLC has the memory area for input and output relay, auxiliary relay, timer, counter and other logic devices. The state of these devices depends on initial setting and operation of the user programs. Some data maintains existing state by using built-in supercapacitors or backup batteries in case of power failure. The memory area for data saving in case of power failure is called the data retention area.

Power supply

The PLC is equipped with a switch power supply for internal circuit. Compared with ordinary power supply, the PLC power supply has the higher stability and interference immunity. A number of PLC products provide 24 V DC stabilized voltage supply to meet external sensors.

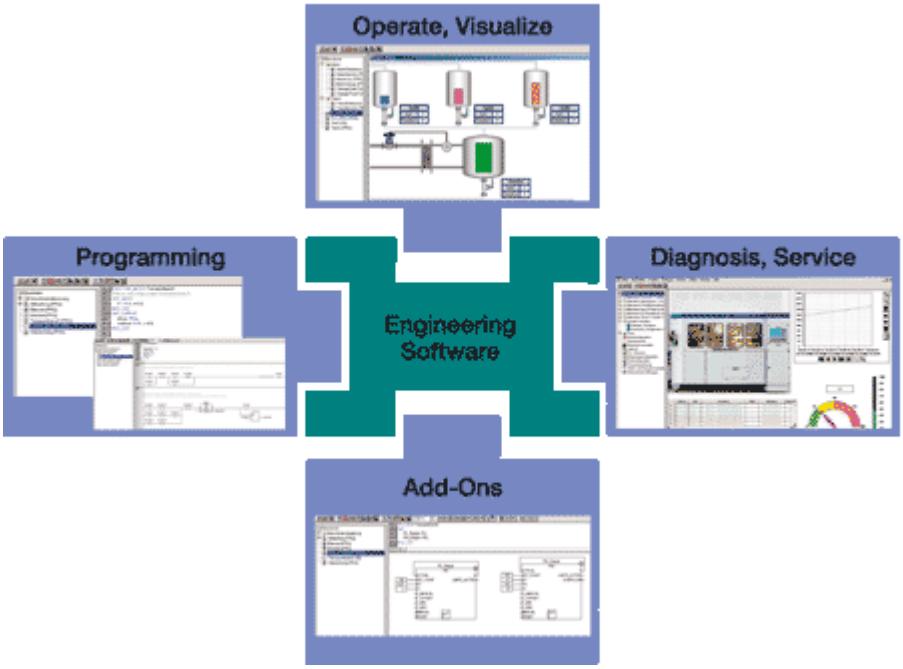
1.6.2.7.4 Short description software

Configuration and programming

Configuration and programming of all AC500 control systems (CPUs) is done by using Automation Builder software.

Features:

- Standardized programming according to IEC 61131-3 - five programming languages (Structured Text (ST), Function Block Diagram (FBD), Instruction List (IL), Ladder Diagram (LD), Sequential Function Chart (SFC)), free graphical function chart (CFC), debugging functions for program test
- Application programming in C/C++
- Online diagnosis
- Debugging functions for the program test: Single step, Single cycle, Breakpoint
- Offline simulation - simulate commands without PLC being connected
- Sampling trace - timing diagrams for process variables
- Recipe management and watch lists
- Visualization
- Configuration of the communication interface modules (for PROFINET, EtherCAT, CANopen, Ethernet, Modbus)
- Programming - serial or via Ethernet networks
- Comprehensive libraries
- Export and import interfaces for devices, signals, applications, visualization, etc.
- Multi-user support and project compare
- Project scripting



Offline simulation

IEC 61131-3 commands can be simulated without a PLC being connected, including the relevant malfunctions. After the program test, the application can be downloaded to the control system.

Sampling trace

Timing diagrams for process variables and storage of data in a ring buffer with event trigger.

Recipe management and watch lists Values of selected variables are displayed. Pre-defined values can be assigned to variables which can then be downloaded to the control system all at once ("Write recipe"). Actual values from the control system can also be pre-assigned for reading into the Watch and Recipe Manager, and stored in memory there ("Read recipe). These functions are also helpful, for example, for setting and entering control parameters.

Visualization Includes color change, moving elements, bitmaps, text display, allows input of setpoint values and display of process variables read from the PLC, dynamic bar diagrams, alarm and event management, function keys and ActiveX elements.

Programming Serial or via Ethernet networks.

Engineering interface Provides access from the programming system to an external project database in which the program source code of one or several automation projects is managed. Optionally, a version control system, such as Visual Source Safe, can be used in order to ensure data consistency of the program code for several different users and projects.

1.6.2.7.5 CP600 control panels (HMI)

ABB control panels offer a wide range of features and functionalities for maximum operability. The panels are distinguished by their robustness and easy usability, providing all the relevant information from production plants and machines at a single touch.

HMI - human control and operation of machines and processes.

Individual solutions for each application - this enables an operator at any time to have an overview on a profitable production and intervene manually if necessary.

Control panels with TFT graphical display and touch screen.

Available in various resolutions.

1.6.2.8 AC500-S

The AC500-S safety user manual must be read and understood before using safety configuration and programming tools of Automation Builder / PS501 Control Builder Plus. Only qualified personnel shall be allowed to work with AC500-S safety PLCs.

In order to have always the latest version and due to a different lifecycle compared to Automation Builder help, the *AC500-S safety user manual* is only available on our website.

The AC500-S safety PLC includes the following safety-relevant hardware components.

- SM560-S / SM560-S-FD-1 / SM560-S-FD-4
- DI581-S
- DX581-S
- AI581-S
- TU582-S

1.6.2.9 Converting an AC500 V2 project to an AC500 V3 project

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

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

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

1.6.3 Device specifications

1.6.3.1 Status LEDs, display and control elements

Depending on the device type, various operating elements provided on the front panel can be used to control the devices of the PLC system and/or to change the operating mode.

Operating elements:

- Status LEDs:
  Indicates the availability of devices/components such as communication modules, communication interface modules or function modules. Functionality and diagnosis of the status LEDs depends on the specific module and is described in the device description of the appropriate module. Possible status: on/off/blinking
- I/O LEDs:
  Displays the status of the the inputs and outputs.
- LED display:
  Available for some processor modules. It can be used for simple configurations and for reading out diagnosis information.
  ↳ *Chapter 1.6.5.1.6 “LEDs, display and function keys on the front panel” on page 3486*
  ↳ *Chapter 1.7.1.2 “Diagnosis in CPU display” on page 4013*
- Function keys and switches:
  Allows to change the current operating modes/status manually ↳ *Chapter 1.6.5.1.6.5 “Description of the function keys” on page 3491*.

1.6.3.2 Terminal bases (AC500 standard)

AC500-eCo V3 processor modules do not have a separate terminal base

1.6.3.2.1 TB56xx for AC500 V3 products

- TB5600-2ETH: 1 processor module, with network interface 2 Ethernet RJ45, 1 CAN and 1 COM1
- TB5610-2ETH: 1 processor module, 1 communication module, with network interface 2 Ethernet RJ45, 1 CAN and 1 COM1

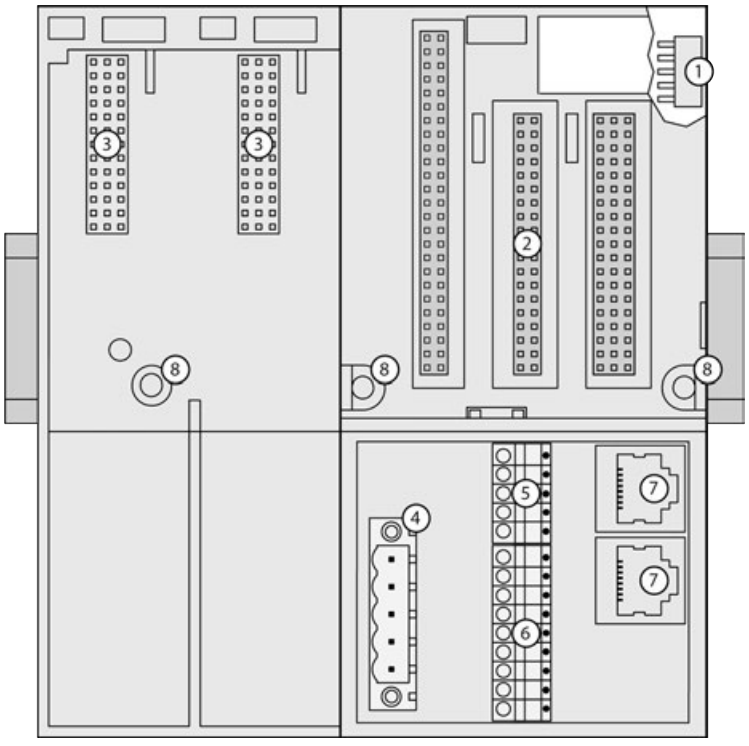
- TB5620-2ETH: 1 processor module, 2 communication modules, with network interface 2 Ethernet RJ45, 1 CAN and 1 COM1
- TB5640-2ETH: 1 processor module, 4 communication modules, with network interface 2 Ethernet RJ45, 1 CAN and 1 COM1
- TB5660-2ETH: 1 processor module, 6 communication modules, with network interface 2 Ethernet RJ45, 1 CAN and 1 COM1
- XC version for use in extreme ambient conditions available

Terminal bases TB56xx-2ETH can only be used with processor modules PM56xx-2ETH.

Table 407: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])
Remarks: The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base. [1]) PM567x must have an index ≥ C0.				

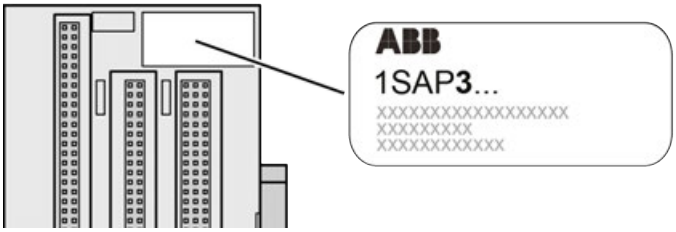
The following figure shows the TB5620-2ETH as example.



1 I/O bus (10-pin, female) to connect the I/O terminal units
2 One available slot for the processor module
3 Slots for communication modules
4 Interface for CAN (5-pin terminal block, removable)
5 Power supply (5-pin terminal block, removable)

6 Serial interface COM1 (9-pin terminal block, removable)
7 RJ45 female connector for Ethernet connection
8 Holes for screw mounting

XC version **XC** = e**X**treme **C**onditions



Extreme conditions

Terminal bases for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **3** in the Part no. 1SAP**3**... (label) identifies the XC version.*

Short description

Terminal bases TB56xx are used as sockets for processor modules PM56xx and communication modules.

Up to 10 I/O terminal units for I/O expansion modules can be added to these terminal bases.

The terminal bases have slots for one processor module and for communication modules as well as terminals and interfaces for power supply, expansion and networking.

Table 408: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])

Remarks:

The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base.

[1]) PM567x must have an index ≥ C0.

NOTICE!

Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↳ *Chapter 1.6.3.8.2.5 “TA524 - Dummy communication module” on page 3328*.
– I/O bus connectors must not be touched during operation.

Connections

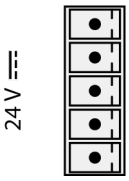
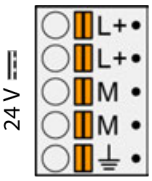
I/O bus

The I/O bus is the I/O data bus for the I/O modules. Through this bus, I/O and diagnosis data are transferred between the processor module and the I/O modules. Up to 10 I/O modules can be added (see description for I/O bus in the system asembly chapter *Chapter 1.6.4.4.1 “Serial I/O bus” on page 3338*).

Power supply

The supply voltage of 24 V DC is connected to a removable 5-pin terminal block. L+/M exist twice. It is therefore possible to feed e.g. external sensors (up to 8 A max. with 1.5 mm^2 conductor) via these terminals, when the ambient temperature never exceeds 60 °C.

Pin assignment

Pin Assignment		Label	Function	Description
 Terminal block removed	 Terminal block inserted	L+	+24 V DC	Positive pin of the power supply voltage
		L+	+24 V DC	Positive pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		⏚	FE	Functional earth

Faulty wiring on power supply terminals

NOTICE!

Risk of damaging the processor module and terminal base!

Exceeding the maximum voltage could lead to unrecoverable damage to the system.

The system might be destroyed.

NOTICE!

Risk of malfunction!

To ensure reliability and proper functionality of processor modules below index C0, the supply voltage must ramp-up from 0 V to 24 V within max. 2.5 s.

NOTICE!

Risk of damaging the terminal base and power supply!

Short circuits might damage the terminal base and power supply.

Make sure that the four clamps L+ and M (two of each) are not wrongly connected (e. g. +/- of power supply is connected to both L+/L+ or both M/M).

NOTICE!

Risk of damaging the terminal base!

Terminal base can be damaged by connecting the power supply terminal block (L+/M) to COM1.

Make sure that the COM1 terminal block is always connected to the terminal base even if you do not use COM1 to prevent this.

NOTICE!

Risk of damaging the terminal base!

Excessive current might damage the clamp and terminal base.

Make sure that the current flowing through the removable clamps never exceeds 8 A (with 1.5 mm^2 conductor).

NOTICE!

For applications using XC versions!

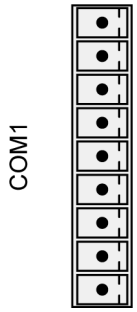
To ensure reliability and proper function, make sure the ambient temperature never exceeds 60 °C when the current flowing through the removable clamps is 8 A (with 1.5 mm^2 conductor).

Serial interface COM1

The serial interface COM1 is connected to a removable 9-pin terminal block.

From firmware version V3.1 it is configurable for RS-232 or RS-485 (V3.0 RS-232 only).

Pin assignment (RS-485 / RS-232)



Pin	Signal	Interface	Description
1	Terminator P	RS-485	Terminator P
2	RxD/TxD-P	RS-485	Receive/Transmit, positive
3	RxD/TxD-N	RS-485	Receive/Transmit, negative
4	Terminator N	RS-485	Terminator N
5	RTS	RS-232	Request to send (output)
6	TxD	RS-232	Transmit data (output)
7	SGND	Signal Ground	Signal Ground
8	RxD	RS-232	Receive data (input)
9	CTS	RS-232	Clear to send (input)

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

For further information on connection and wiring please refer to .

Ethernet interface

This interface is the connection to a processor module with onboard Ethernet e.g. PM56xx-2ETH.

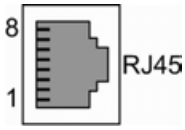
TB56xx-2ETH for processor modules PM56xx-2ETH provide 2 independent Ethernet interfaces.

The two Ethernet interfaces can be configured as independent interfaces or with switch functionality.

In case of two independent interfaces they must be configured to different subnets.

For structured Ethernet cabling only use cables according to TIA/EIA-568-A, ISO/IEC 11801 or EN 50173.

Pin assignment

Interface	Pin	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used
	6	RxD-	Receive data -
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.3.8.3.4 “TA535 - Protective caps for XC devices” on page 3333*

See supported protocols and used Ethernet ports for AC500 V3 products: *Chapter 1.6.1.3 “Ethernet protocols and ports for AC500 V3 products” on page 2389*.

See communication via Modbus for AC500 V3 products: *Chapter 1.6.5.1.11 “Communication with Modbus TCP/IP” on page 3558*.

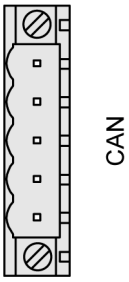
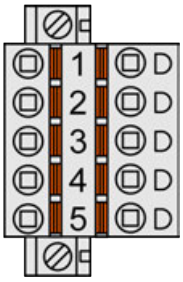
See communication via Modbus for AC500 V3 products: *Chapter 1.6.5.1.10 “Communication with Modbus RTU” on page 3542*.

CAN interface

This interface is the connection to a processor module with onboard CAN e.g. PM56xx-2ETH.

Interface socket	COMBICON, 5-pin, female, removable plug with spring terminals
Transmission standard	ISO 11898, potential-free
Transmission protocol	CANopen (CAN), 1 Mbaud max.
Transfer rate (transmission rate)	50 kbit/s, 100 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s,

Pin assignment

Interface		PIN	Signal	Description
 Terminal block removed	 Terminal block inserted	1	CAN_GND	CAN reference potential
		2	CAN_L	Bus line, receive/transmit line, LOW
		3	CAN_SHLD	Shield of the bus line
		4	CAN_H	Bus line, receive/transmit line, HIGH
		5	NC	Not connected

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
50 kbit/s	1000 m

Types of bus cables

Only bus cables with characteristics as recommended in ISO 11898 are to be used. The requirements for the bus cables depend on the length of the bus segment. See *Chapter 1.6.4.6.4.6 “CANopen field bus” on page 3422*.

Bus terminating resistors

Both ends of the CAN bus have to be terminated with a 120 Ω (≥ 1/4 W, ≤ 5 %) bus terminating resistor, to minimize signal reflection. The bus terminating resistor should be connected directly at the bus connector between the CAN signals (CAN_H and CAN_L). See *Chapter 1.6.4.6.4.6 “CANopen field bus” on page 3422*.

Technical data

The system data of AC500 and S500 are applicable to the standard version. *Chapter 1.6.4.6.1 “System data AC500” on page 3398*

The system data of AC500-XC are applicable to the XC version. *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Connection of the supply voltage 24 V DC at the terminal base of the processor module	Removable 5-pin terminal block spring type
Max. current consumption from 24 V DC	TB5600: 0.25 A [1]) TB5610: 0.35 A [1]) TB5620: 0.4 A [1]) TB5640: 0.6 A [1]) TB5660: 0.8 A [1])
Melting integral of a fuse at 24 V DC	Min. 1 A^2s [2])
Peak inrush current from 24 V DC	55 A [2])
Number of slots for processor modules	1 (on all terminal bases)
Processor module interfaces at TB56xx	I/O bus, ETH1, ETH2, CAN, COM1
Net weight (terminal base without processor module)	TB5600: 155 g TB5610: 180 g TB5620: 210 g TB5640: 260 g TB5660: 310 g
Mounting position	Horizontal or vertical

[1]) Including processor modules, communication modules and communication interface modules

[2]) The inrush current and the melting integral depends on the internal power supply of the processor module and the number and type of communication modules and I/O modules connected to the I/O bus.

Table 409: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])
Remarks: The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base. [1]) PM567x must have an index ≥ C0.				

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 110 300 R0278	TB5600-2ETH, terminal base AC500, slots: 1 processor module, 2 Ethernet RJ45, 1 CAN connector	Active
1SAP 310 300 R0278	TB5600-2ETH-XC, terminal base AC500, slots: 1 processor module, 2 Ethernet RJ45, 1 CAN connector, XC version	Active
1SAP 111 300 R0278	TB5610-2ETH, terminal base AC500, slots: 1 processor module, 1 communication module, 2 Ethernet RJ45, 1 CAN connector	Active
1SAP 311 300 R0278	TB5610-2ETH-XC, terminal base AC500, slots: 1 processor module, 1 communication module, 2 Ethernet RJ45, 1 CAN connector, XC version	Active
1SAP 112 300 R0278	TB5620-2ETH, terminal base AC500, slots: 1 processor module, 2 communication modules, 2 Ethernet RJ45, 1 CAN connector	Active
1SAP 312 300 R0278	TB5620-2ETH-XC, terminal base AC500, slots: 1 processor module, 2 communication modules, 2 Ethernet RJ45, 1 CAN connector, XC version	Active
1SAP 114 300 R0278	TB5640-2ETH, terminal base AC500, slots: 1 processor module, 4 communication modules, 2 Ethernet RJ45, 1 CAN connector	Active
1SAP 314 300 R0278	TB5640-2ETH-XC, terminal base AC500, slots: 1 processor module, 4 communication modules, 2 Ethernet RJ45, 1 CAN connector, XC version	Active
1SAP 116 300 R0278	TB5660-2ETH, terminal base AC500, slots: 1 processor module, 6 communication modules, 2 Ethernet RJ45, 1 CAN connector	Active
1SAP 316 300 R0278	TB5660-2ETH-XC, terminal base AC500, slots: 1 processor module, 6 communication modules, 2 Ethernet RJ45, 1 CAN connector, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 410: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots

Processor module	PM5630	PM5650	PM5670	PM5675
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])
Remarks: The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base. [1]) PM567x must have an index ≥ C0.				

Table 411: Accessories

Part no.	Description
1SAP 180 800 R0001	TA526, wall mounting accessory

1.6.3.3 Processor modules

The AC500 product family consists of the product groups:

- AC500 (standard):
  AC500 standard PLCs offer a wide range of performance levels and scalability. The PLCs are highly capable of communication and extension for flexible application.
- AC500-eCo:
  AC500-eCo PLCs are cost-effective, high-performance compact PLCs that offer total interoperability with the core AC500 range and provide battery-free data buffering. All I/O modules can be freely connected in a simple, stable and reliable manner.
- AC500-S:
  AC500-S PLCs are designed for safety applications involved in factory, process or machinery automation area.
- AC500-XC:
  AC500 (standard) and AC500-S provide devices with -XC extension as a product variant. These variants operate according to their product group and can, in addition, be operated under extreme conditions. AC500-XC PLCs can be used at high altitudes, extended operating temperature and in humid condition. Further, the devices provide immunity to vibration and hazardous gases. The AC500-XC series is consistent with standard devices in the overall dimensions, control function and software compatibility. ↳ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*.

The AC500 product family is characterized by functional modularity. As the complete AC500 product family shares the same hardware platform and programming software tool, the devices of the AC500 product groups can be flexibly combined.

S500 devices represent the I/O modules of the product group AC500 (standard), whereas S500-eCo devices represent the I/O modules of the product group AC500-eCo. Both S500 and S500-eCo devices can be combined with devices of the AC500 product family in a flexible way.

1.6.3.3.1 AC500-eCo

PM50xx

The following table lists all AC500-eCo V3 CPUs with their most important properties.

Processor modules	Global user memory	Configurable input/output	Digital inputs	Digital outputs	Power supply	Ethernet interfaces	Option board slots
Basic CPUs							
PM5012-T-ETH	1 MB thereof 256 kB for user program code and data dynamically allocated	-	6	4 (Transistor)	24 V DC	1	1
PM5012-R-ETH	1 MB thereof 256 kB for user program code and data dynamically allocated	-	6	4 (Relay)	24 V DC	1	1
Standard CPUs							
PM5032-T-ETH	2 MB thereof 512 kB for user program code and data dynamically allocated	2 (Transistor)	12	8 (Transistor)	24 V DC	1	2
PM5032-R-ETH	2 MB thereof 512 kB for user program code and data dynamically allocated	2 (Transistor)	12	6 (Relay)	24 V DC	1	2
PM5052-T-ETH	4 MB thereof 768 kB for user program code and data dynamically allocated	2 (Transistor)	12	8 (Transistor)	24 V DC	1	3
PM5052-R-ETH	4 MB thereof 768 kB for user program code and data dynamically allocated	2 (Transistor)	12	6 (Relay)	24 V DC	1	3
Pro CPUs							

Processor modules	Global user memory	Configurable input/output	Digital inputs	Digital outputs	Power supply	Ethernet interfaces	Option board slots
PM5072-T-2ETH	8 MB thereof 1 MB for user program code and data dynamically allocated	2 (Transistor)	12	8 (Transistor)	24 V DC	2	3
PM5072-T-2ETHW *)	8 MB thereof 1 MB for user program code and data dynamically allocated	2 (Transistor)	12	8 (Transistor)	24 V DC	2	3

*) W = wide temperature

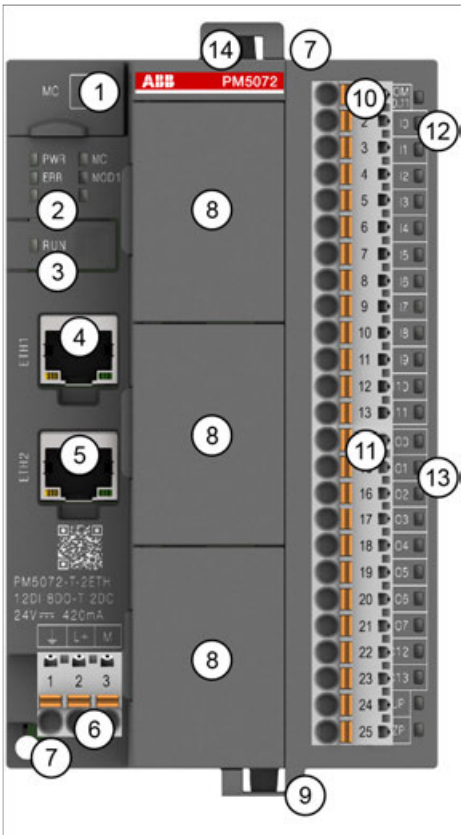


Fig. 99: Example: PM5072-T-2ETH

1 Micro memory card slot
2 5 LEDs to display the states of the processor module (Power, Error, Run, MC, MOD1)
3 RUN button
4 RJ45 female connector for Ethernet1 connection
5 RJ45 female connector for Ethernet2 connection (available for PM5072-T-2ETH(W))
6 3-pin terminal block for power supply 24 V DC
7 2 holes for screw mounting
8 Option board slot cover for option board slot (the number of available slots varies according to the CPU type)
9 Cable fixing

10 13-pin terminal block for onboard I/Os
11 12-pin terminal block for onboard I/Os (not available on PM5012-x-ETH)
12 12 LEDs to display the states of the signals
13 10 LEDs to display the states of the signals
14 Cable fixing accessory TA5301-CFA on the top of the housing (optional)

The processor module is shown with pluggable terminal blocks. These terminal blocks must be ordered separately.

The cable fixing accessory on the top of the housing is optional.

Please use TA5301-CFA cable fixing accessory to provide strain relief.

It can also be used for AC500-eCo I/O modules.

The PM50x2 processor modules are supplied with option board slot covers as standard.

There are various TA51xx option boards for the processor modules that can be ordered separately.

Which and how many option boards can be plugged, depends on the respective processor module.

Short description

The processor modules PM50xx series are the central units of AC500-eCo V3 PLC. Their main characteristics are:

- Power supply 24 V DC
- I/O bus (not for PM5012-x-ETH)
- Real-time clock (PM5012-x-ETH needs additional RTC option board)
- Option board slots for extension on the CPU (1 for PM5012-x-ETH, 2 for PM5032-x-ETH, 3 for PM5052-x-ETH and PM5072-T-2ETH)
- 6 digital inputs (PM5012-x-ETH), 12 digital inputs (PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH)
- 4 transistor outputs (PM5012-T-ETH), 8 transistor outputs (PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH)
- 4 relay outputs (PM5012-R-ETH), 6 relay outputs (PM5032-R-ETH, PM5052-R-ETH)
- 2 configurable digital inputs/outputs (not for PM5012-x-ETH)

The various processor module variants differ in the following characteristics:

- Type of the digital outputs (transistor or relays)
- Ethernet interface one or two independent interfaces

All processor module variants include a micro memory card slot.

Details and technical data are provided in the technical data section *Chapter 1.6.3.3.1.1.8 “Technical data” on page 2472*.

Assortment

Processor module	Total max-imum download-able application size	Allocated global user memory for user program code and data	Cycle time for 1000 instructions [ns]	Numer digital inputs	Number digital outputs	Type of digital outputs	Config-urable digital inputs/ outputs	Number of option board slots	Max. number of I/O modules on I/O bus
PM5012-T-ETH	1 MB	256 kB	Binary: 20 Word: 50 Floating: 600	6	4	Tran-sistor	-	1	-
PM5012-R-ETH	1 MB	256 kB		6	4	Relay	-	1	-
PM5032-T-ETH	2 MB	512 kB		12	8	Tran-sistor	2	2	10 with max. 128 Bytes inputs/ 128 Bytes outputs variables
PM5032-R-ETH	2 MB	512 kB		12	6	Relay	2	2	10 with max. 128 Bytes inputs/ 128 Bytes outputs variables
PM5052-T-ETH	4 MB	768 kB		12	8	Tran-sistor	2	3	10
PM5052-R-ETH	4 MB	768 kB		12	6	Relay	2	3	10
PM5072-T-2ETH	8 MB	1 MB		12	8	Tran-sistor	2	3	10
PM5072-T-2ETHW	8 MB	1 MB		12	8	Tran-sistor	2	3	10

Connections and interfaces

I/O bus

The I/O bus is not available for PM5012-T-ETH and PM5012-R-ETH. I/O channel extension using option board slot only.

The I/O bus is the I/O data bus for the I/O modules. Through this bus, I/O and diagnosis data are transferred between the processor module and the I/O modules. Up to 10 I/O modules for PM5032-x-ETH (but with a limit of 128 Bytes input/ 128 Bytes output variables) and 10 I/O modules for PM5052-x-ETH and PM5072-T-2ETH can be added.

Option board slot interface

Depending on the processor module variants, an additional option board can be connected to the option board slot to extend the feature of the processor module ↳ *Chapter 1.6.2.6.2.1.1 “Option boards for AC500-eCo V3 processor modules” on page 2410* .

Serial interface

RS-232 communication interface is available by using option board:

- TA5141-RS232I (isolated)
  ↳ *Chapter 1.6.3.3.1.2.6 “TA5141-RS232I - Option board for COMx serial communication” on page 2502*

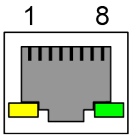
RS-485 communication interface is available by using option boards:

- TA5142-RS485I (isolated)
  ↳ *Chapter 1.6.3.3.1.2.7 “TA5142-RS485I - Option board for COMx serial communication” on page 2504*
- TA5142-RS485 (non isolated)
  ↳ *Chapter 1.6.3.3.1.2.8 “TA5142-RS485 - Option board for COMx serial communication” on page 2510*

Ethernet interface

The Ethernet interface is carried out via a RJ45 jack.

Table 412: Pin assignment of the Ethernet interface

Interface	Pin	Description	
	1	Tx+	Transmit data +
	2	Tx-	Transmit data -
	3	Rx+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	Rx-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

Power supply

The processor modules PM50x2 can be connected to the 24 V DC supply voltage via a removable 3-pin spring terminal block or a 3-pin screw terminal block.

Table 413: Removable terminal block for the supply voltage 24 V DC

3-pin spring terminal block	3-pin screw terminal block
	

The terminal block is available as a set for AC500-eCo V3 processor modules.

Basic CPU (PM5012)		Standard CPUs (PM5032, PM5052) and Pro CPUs (PM5072)	
Spring type	**Screw type**	**Spring type**	**Screw type**
TA5211-TSPF-B	TA5211-TSCL-B	TA5212-TSPF	TA5212-TSCL

Further information on the terminal blocks concerning power supply and onboard inputs/outputs are provided under pluggable connectors for screw and spring connection *Chapter 1.6.3.8.1.2 “TA52xx(-x) - Terminal block sets” on page 3293*.

Pin assignment

Pin Assignment	Pin	Label	Function	Description
 Terminal block inserted	1	⏚	FE	Functional earth
	2	L+	+24 V DC	Positive pin of the power supply voltage
	3	M	0 V	Negative pin of the power supply voltage

Faulty wiring on power supply terminals

CAUTION!

Risk of damaging the AC500-eCo V3 processor module and the connected modules!

Voltages > 30 V DC might damage the processor module and the connected modules.

Make sure that the supply voltage never exceeds 30 V DC.

State LEDs and operating elements

RUN/STOP button

The processor modules, PM50xx series, have a RUN/STOP button. By pressing the RUN/STOP button, the processer modules switch between RUN mode and STOP mode. By long-pressing RUN/STOP button during the processor module power on phase, the processor module will be in MOD1.

State LEDs

The processor modules PM50xx indicate their states of operation via 5 LEDs located on the upper left side of the processor module.

LED	State	Color	LED = ON	LED = OFF	LED flashing
PWR	Power supply	Green	Power supply present	Power supply missing	-
MC	Micro memory card indication	Yellow	Micro memory card is in the socket	Micro memory card is not in the socket	Micro memory card is in read/ write state: any file on card is opened, means activity on card

LED	State	Color	LED = ON	LED = OFF	LED flashing
ERR	Error indication	Red	An error occurred	No errors or only warnings encountered (E4 errors). The LED behavior for the error classes 2 to 4 is configurable.	Fast flashing (4 Hz) displays together with the RUN LED a currently running firmware-upgrade or writing data to the Flash-EPROM. Slow flashing (1 Hz) alone displays shutdown of Request To Send. Medium flashing (2 Hz) alone displays at start of PLC if reboot after watchdog.
MOD1	Mode 1 indication	Yellow	Processor module is in mode 1 state	Processor module is not in mode 1 state	-
RUN	RUN/STOP state	Green	Processor module is in state RUN	Processor module is in state STOP	Fast flashing (4 Hz): The processor module is reading/writing data from/to the memory card. If the ERR-LED is also flashing, data is being written to the Flash-EPROM.

LED	State	Color	LED = ON	LED = OFF	LED flashing
					Slow flashing (1 Hz): The firmware update from the memory card has been completed successfully or Boot project is being updated. Slow flashing (0.5 Hz) together with MOD1 LED ON: Mode1: Boot project is not loaded.
Two LEDs below “ERR” and “MOD1”	Configurable	Yellow	Configurable	Configurable	Additional two LEDs are reserved and can be controlled from IEC user code with FB PmLedSet

User configurable LEDs

The AC500-eCo V3 processor module also provides 2 LEDs below the state LEDs which can be used by user and driven by an application.

The LEDs can be used into a project and controlled using special function blocks which are contained in the PM AC500 library. The POU is PmLedSet located in folder LED control.

I/O LEDs

The processor module provides up to 10 LEDs (PM5012-x-ETH), 20 LEDs (PM5032-R-ETH, PM5052-R-ETH), or 22 LEDs (PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH) to display the states of the inputs and outputs.

Processor module	LED	State	Color	LED = ON	LED = OFF
PM5012-x-ETH	I0..I5	Digital input	Yellow	Input is ON	Input is OFF
	O0..O3	Transistor output	Yellow	Output is ON	Output is OFF
	NO0..NO3	Relay output	Yellow	Output is ON	Output is OFF
PM5032-x-ETH PM5052-x-ETH	I0..I11	Digital input	Yellow	Input is ON	Input is OFF
	O0..O7	Transistor output	Yellow	Output is ON	Output is OFF
	NO0..NO5	Relay output	Yellow	Output is ON	Output is OFF
	C12, C13	Digital configurable input/ output	Yellow	Input/Output is ON	Input/Output is OFF
PM5072-T-2ETH	I0..I11	Digital input	Yellow	Input is ON	Input is OFF

Processor module	LED	State	Color	LED = ON	LED = OFF
PM5072-T-2ETHW	O0..O7	Transistor output	Yellow	Output is ON	Output is OFF
	C12, C13	Digital configurable input/ output	Yellow	Input/Output is ON	Input/Output is OFF

Ethernet state LEDs

Table 414: State LEDs at Ethernet connector

LED	Color	OFF	ON	Flashing
Activity	Yellow	No activity	---	Activity
Link	Green	No link	Link	---

Diagnosis

The AC500 processor module can display various errors according to the error classes. The following error classes are possible. The reaction of the processor module is different for each type of error.

Error class	Type	Description	Example
E1 ERR-LED is ON	Fatal error	A safe function of the operating system is no longer guaranteed.	Checksum error in the system Flash or RAM error
E2 ERR-LED is ON	Severe error	The operating system is functioning without problems, but the error-free processing of the user program is no longer guaranteed.	Checksum error in the user Flash, independent of the task duration
E3 ERR-LED is ON/OFF *)	Light error	It depends on the application if the user program should be stopped by the operating system or not. The user should determine which reaction is necessary.	Flash could not be programmed, I/O module has failed
E4 ERR-LED is ON/OFF *)	Warning	Error in the periphery (e.g. I/O) which may show an impact in the future. The user should determine which reaction is necessary.	Short-circuit at an I/O module, the battery is run down or not inserted
*) The behaviour if the ERR-LED lights up at error classes E3 or E4 is configurable.			

Occurred errors can be displayed with the commands diagshow all in the PLC-Browser of Automation Builder software.

Onboard I/Os

The AC500-eCo V3 processor modules have onboard I/Os which provide several functionalities. According to the CPU type, the number or the functionality of the onboard I/Os can be different.

Intended purpose

Table 415: Numbers and types of the onboard I/Os

Processor module	No. and type of digital inputs	No. and type of digital outputs	No. and type of configurable inputs/outputs
PM5012-T-ETH	6 24 V DC (one isolation group)	4 0.5 A max., transistor (one isolation group)	None
PM5012-R-ETH	6 24 V DC (one isolation group)	4 2 A max., relay (two isolation groups)	None
PM5032-T-ETH	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5032-R-ETH	12 24 V DC (one isolation group)	6 2 A max., relay (two isolation groups)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5052-T-ETH	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5052-R-ETH	12 24 V DC (one isolation group)	6 2 A max., relay (two isolation groups)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5072-T-2ETH	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5072-T-2ETHW	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)

Functionality

<table>
<tr><th rowspan="2">Parameter</th><th colspan="4">Value</th></tr>
<tr><th>PM5012-T-ETH</th><th>PM5012-R-ETH</th><th>PM5032-T-ETH
PM5052-T-ETH
PM5072-T-2ETH(W)</th><th>PM5032-R-ETH
PM5052-R-ETH</th></tr>
<tr><td>Digital inputs</td><td colspan="2">6</td><td colspan="2">12</td></tr>
<tr><td>Functionality of digital inputs
(encoder, fast counter, counter, interrupt)</td><td colspan="2" rowspan="3">6 DI fast input 24 V DC (max. 5 kHz)
usable as
● 6 DI 24 V DC standard
● 2 channel 5 kHz encoder with frequency measurement or
● 2 channel 5 kHz encoder with frequency measurement and with touch/reset using standard DI or
● 2 fast counter (5 kHz)
● 4 DI as interrupt input with 1 dedicated interrupt task and input information</td><td colspan="2">4 DI fast input 24 V DC (max. 200 kHz)
usable as
● 4 DI 24 V DC standard or
● 4 fast counter (100 kHz) or
● 2 A/B encoder (200 kHz) with frequency measurement or
● 2 full A/B encoders 0 and 1 (200 kHz) with frequency measurement and with touch/reset using standard highspeed (5 kHz) DI
● 1 full A/B encoder 0 (200 kHz) with frequency measurement and optional with touch/reset using 2 touch/sync inputs with A/B encoder 0</td></tr>
<tr><td colspan="2">4 DI fast input 24 V DC (5 kHz)
usable as
● 4 DI 24 V DC standard or
● 4 DI as interrupt input with 1 dedicated interrupt task and input information
● 4 touch/sync inputs with A/B encoder 0 or 1</td></tr>
<tr><td colspan="2">4 standard DI 24 V DC</td></tr>
<tr><td>Digital outputs</td><td colspan="2">4</td><td>8</td><td>6</td></tr>
</table>

Parameter	Value			
	PM5012-T-ETH	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH(W)**	**PM5032-R-ETH** **PM5052-R-ETH**
Functionality of digital outputs	**4 fast output DO-T** 24 V DC/0.5 A (max. 5 kHz) usable as ● 4 DO-T 24 V DC/0.5 A or ● 4 PWM Note: The speed must be limited below 100 Hz. The low speed PWM can be used for heating control. ● 4 limit switch	**4 DO-R** 24 V DC / 240 V AC 2A in 2 groups	**4 fast output DO-T** 24 V DC (100 kHz) usable as ● 4 DO-T 24 V DC/0.5 A ● 4 limit/ switch outputs for encoder/ counter or ● 4 PWM (30 kHz, 2 µs accuracy and maximum duty 95 %) or ● 2 PTO (200 kHz) CW/CCW or Pulse/Direc-tion ● 4 PTO (PWM) 100 kHz Pulse/ Direction using standard output **4 fast output DO-T** 24 V DC/0.5 A (5 kHz) (max. 5 kHz) usable as ● 4 DO-T 24 V DC/0.5 A ● 4 limit/ switch outputs for encoder/ counter or ● 4 PWM Note: The speed must be limited below 100 Hz. The low speed PWM can be used for heating control.	**6 DO-R** 24 V DC / 240 V AC 2A in 2 groups

Parameter	Value			
	PM5012-T-ETH	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH(W)**	**PM5032-R-ETH** **PM5052-R-ETH**
Digital inputs/ outputs, configurable	-	-	2	2
Functionality of digital inputs/ outputs, configurable	-	-	**2 DC 24 V DC** • 2 standard I/Os configurable	**2 DC 24 V DC** usable as • 2 DC standard (DI 24 V DC or DO-T) or • 2 PWM (30 kHz) or • 1 PTO (200 kHz) as Pulse/Direction or CW/CCW
LED displays	For signal states			
Internal power supply	Via processor module			
External power supply	Via UP and ZP terminal			

Connections

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

When replacing a processor module, it is recommended to mark each wire connected to the onboard I/O terminal block before disconnecting it. This should make sure that the wires can be reconnected in the same order.

The connection is carried out by using removable 12-pin and 13-pin terminal blocks.

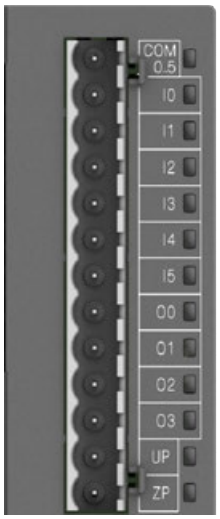


Table 416: Assignment of the terminals for PM5012-T-ETH:

Terminal	Signal	Description
1	COM 0..5	Input common for digital input signals I0 to I5
2	I0	Digital input signal I0 (5 kHz)
3	I1	Digital input signal I1 (5 kHz)
4	I2	Digital input signal I2 (5 kHz)
5	I3	Digital input signal I3 (5 kHz)
6	I4	Digital input signal I4 (5 kHz)
7	I5	Digital input signal I5 (5 kHz)
8	O0	Digital output signal O0 (5 kHz)
9	O1	Digital output signal O1 (5 kHz)
10	O2	Digital output signal O2 (5 kHz)
11	O3	Digital output signal O3 (5 kHz)
12	UP	Process supply voltage UP +24 V DC
13	ZP	Process supply voltage ZP 0 V DC

Table 417: Assignment of the terminals for PM5012-R-ETH:

Terminal	Signal	Description
1	COM 0..5	Input common for digital input signals I0 to I5
2	I0	Digital input signal I0 (5 kHz)
3	I1	Digital input signal I1 (5 kHz)
4	I2	Digital input signal I2 (5 kHz)
5	I3	Digital input signal I3 (5 kHz)
6	I4	Digital input signal I4 (5 kHz)
7	I5	Digital input signal I5 (5 kHz)
8	NO0	Normally-open relay contact of the output NO0
9	NO1	Normally-open relay contact of the output NO1
10	R0..1	Output common for signals NO0 to NO1

Terminal	Signal	Description
11	NO2	Normally-open relay contact of the output NO2
12	NO3	Normally-open relay contact of the output NO3
13	R2..3	Output common for signals NO2 to NO3

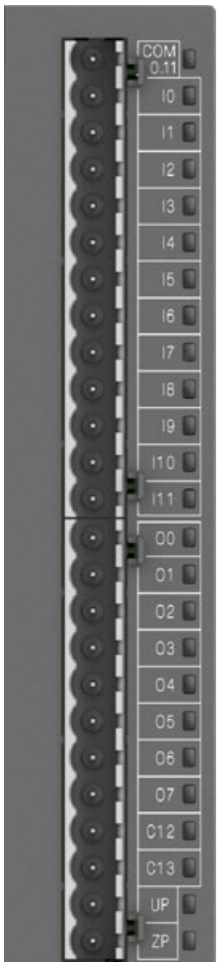


Table 418: Assignment of the terminals for PM5032-T-ETH, PM5052-T-ETH and PM5072-T-2ETH(W):

Terminal	Signal	Description
1	COM 0..11	Input common for digital input signals I0 to I11
2	I0	Digital input signal I0 (max. 5 kHz)
3	I1	Digital input signal I1 (max. 5 kHz)
4	I2	Digital input signal I2 (max. 5 kHz)
5	I3	Digital input signal I3 (max. 5 kHz)
6	I4	Digital input signal I4 Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
7	I5	Digital input signal I5 (100 kHz) Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
8	I6	Digital input signal I6 (100 kHz) Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
9	I7	Digital input signal I7 (100 kHz) Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
10	I8	Digital input signal I8
11	I9	Digital input signal I9
12	I10	Digital input signal I10
13	I11	Digital input signal I11
14	O0	Digital output signal O0 (max. 5 kHz)
15	O1	Digital output signal O1 (max. 5 kHz)
16	O2	Digital output signal O2 (max. 5 kHz)
17	O3	Digital output signal O3 (max. 5 kHz)
18	O4	Digital output signal O4 PWM (max. 100 kHz), PTO (max. 200 kHz)
19	O5	Digital output signal O5 PWM (max. 100 kHz), PTO (max. 200 kHz)
20	O6	Digital output signal O6 PWM (max. 100 kHz), PTO (max. 200 kHz)
21	O7	Digital output signal O7 PWM (max. 100 kHz), PTO (max. 200 kHz)
22	C12	Digital input/output signal configurable C12
23	C13	Digital input/output signal configurable C13

Terminal	Signal	Description
24	UP	Process supply voltage UP +24 V DC
25	ZP	Process supply voltage ZP 0 V DC

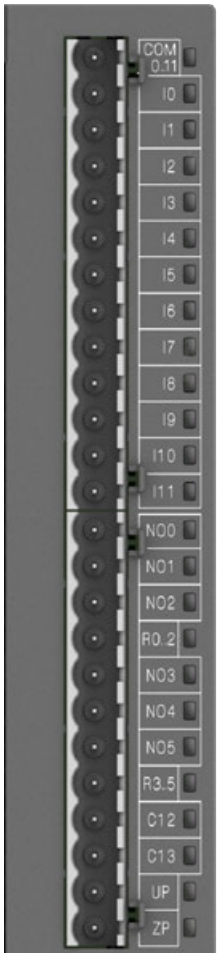
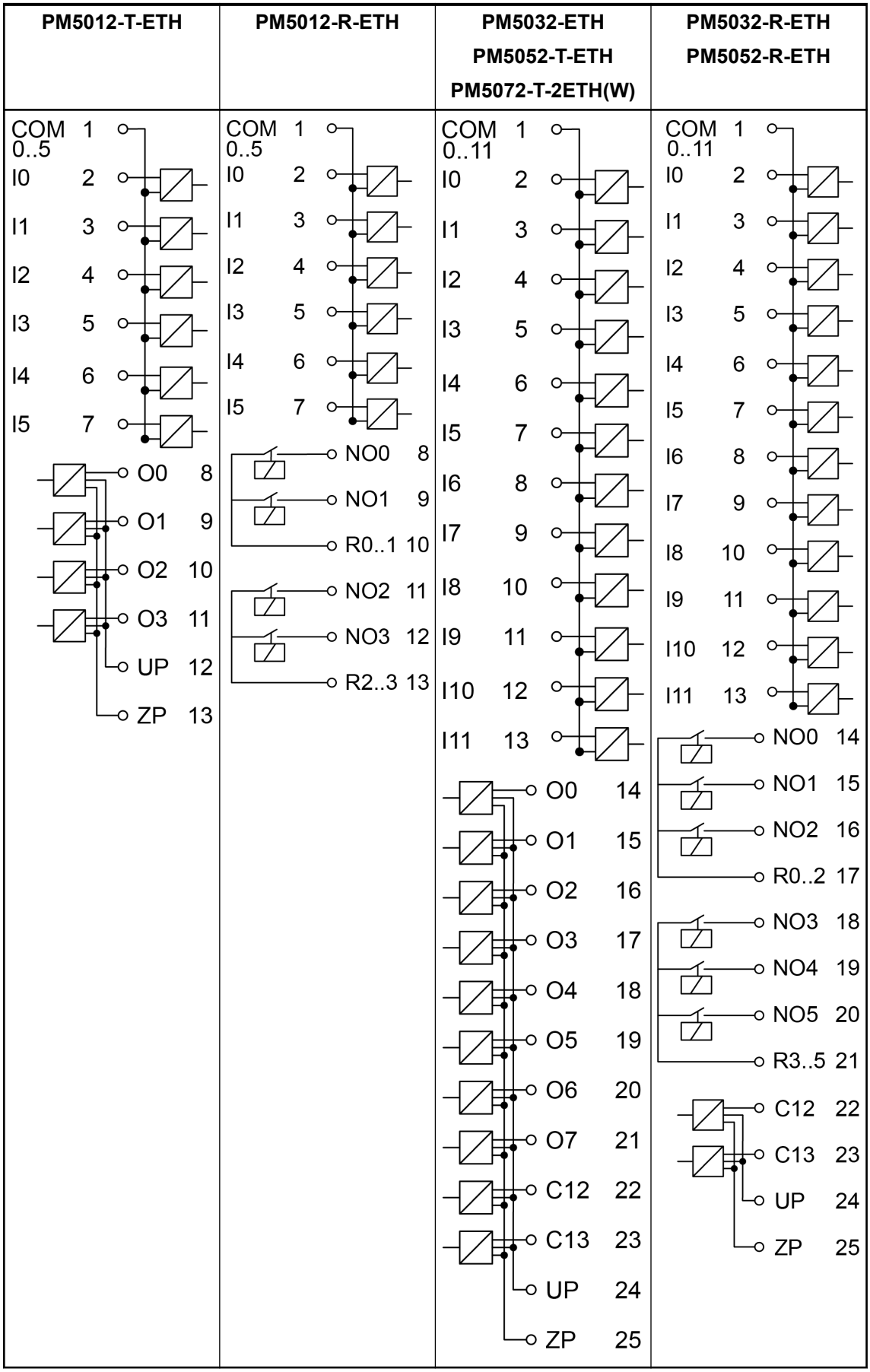


Table 419: Assignment of the terminals for PM5032-R-ETH and PM5052-R-ETH:

Terminal	Signal	Description
1	COM 0..11	Input common for digital input signals I0 to I11
2	I0	Digital input signal I0 (max. 5 kHz)
3	I1	Digital input signal I1 (max. 5 kHz)
4	I2	Digital input signal I2 (max. 5 kHz)
5	I3	Digital input signal I3 (max. 5 kHz)
6	I4	Digital input signal I4 Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
7	I5	Digital input signal I5 Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
8	I6	Digital input signal I6 Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
9	I7	Digital input signal I7 Forward counter (max. 100 kHz), Encoder (max. 200 kHz)
10	I8	Digital input signal I8
11	I9	Digital input signal I9
12	I10	Digital input signal I10
13	I11	Digital input signal I11
14	NO0	Normally-open relay contact of the output NO0
15	NO1	Normally-open relay contact of the output NO1
16	NO2	Normally-open relay contact of the output NO2
17	R0..2	Output common for signals NO0 to NO2
18	NO3	Normally-open relay contact of the output NO3
19	NO4	Normally-open relay contact of the output NO4
20	NO5	Normally-open relay contact of the output NO5
21	R3..5	Output common for signals NO3 to NO5
22	C12	Digital input/output signal configurable C12 PWM (max. 100 kHz), PTO (max. 200 kHz)
23	C13	Digital input/output signal configurable C13 PWM (max. 100 kHz), PTO (max. 200 kHz)
24	UP	Process supply voltage UP +24 V DC
25	ZP	Process supply voltage ZP 0 V DC

Block diagrams The following block diagram shows the internal structure of the onboard I/Os.



Connection of the digital inputs

The digital inputs can be used as source inputs or as sink inputs.

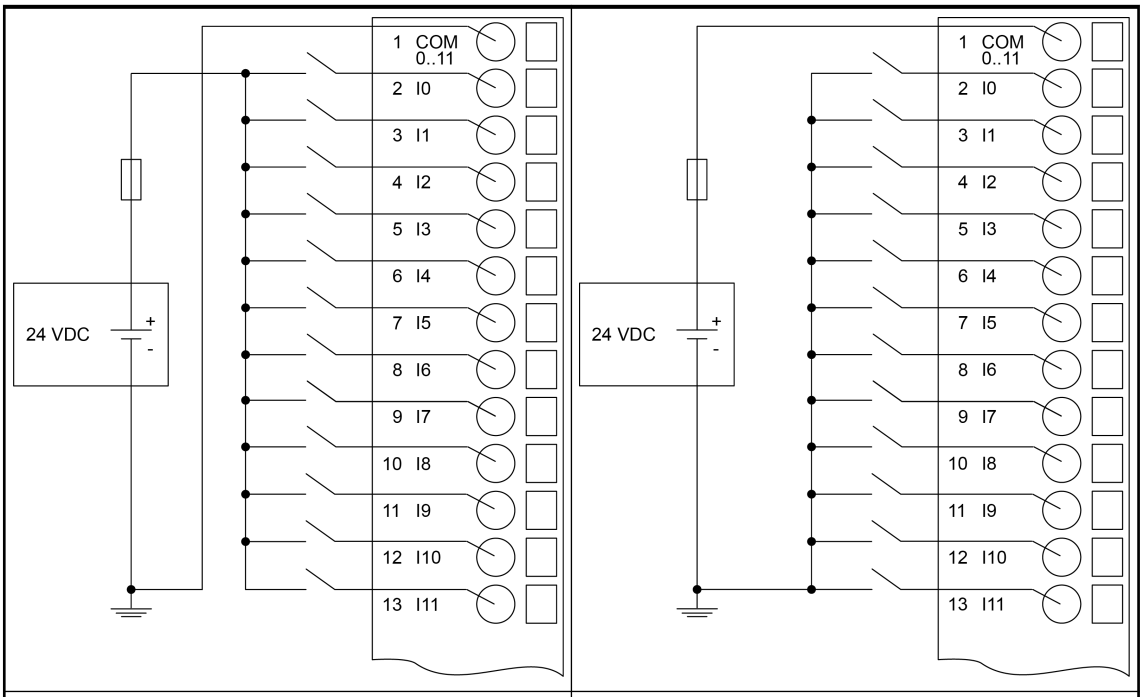
NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the digital inputs to the PM50x2 processor modules:



Connection of digital inputs (sink inputs)	Connection digital inputs (source inputs)

Connection of the digital transistor outputs (PM50xx-T-ETH only)

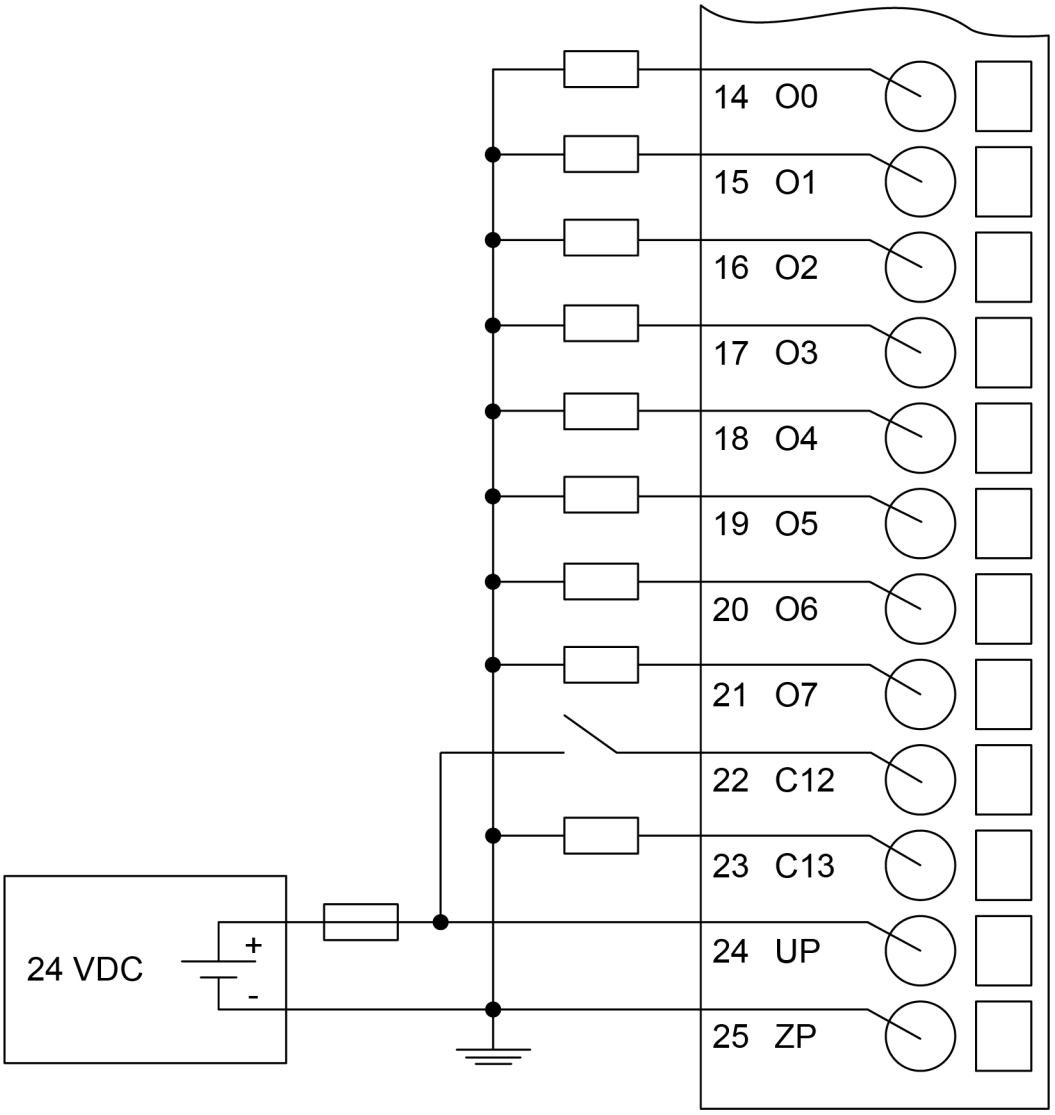


Fig. 100: Connection of digital transistor outputs and configurable digital inputs/outputs

C12 used as configurable digital input
C13 used as configurable digital transistor output

CAUTION!
Risk of damaging the processor module!
The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external fuse for the outputs.

Connection of the digital relay outputs (PM50xx-R-ETH only)

The following figures show the connection of the digital relay outputs to the processor modules:

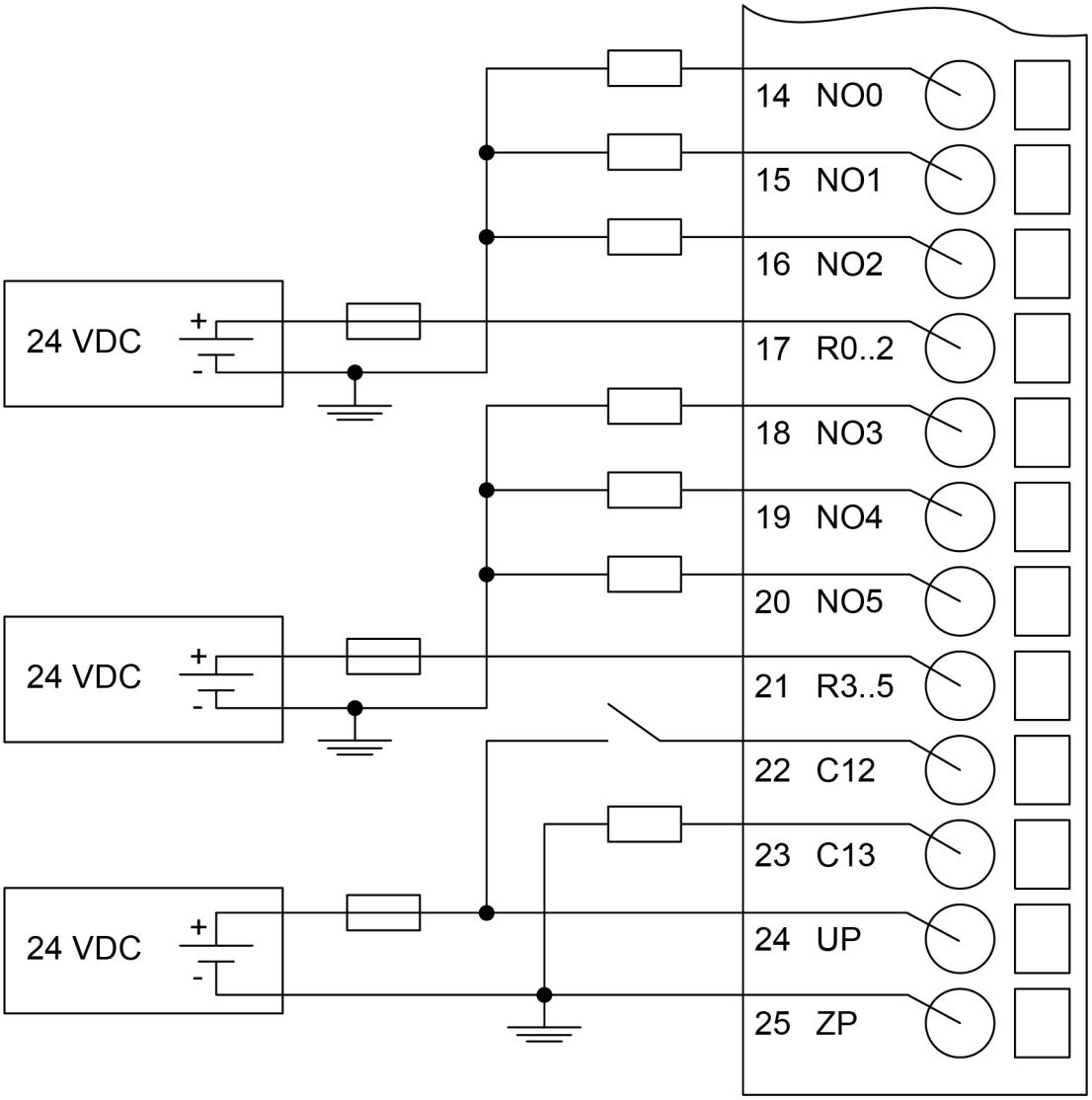


Fig. 101: Connection of digital relay outputs and configurable digital inputs/outputs

C12 used as configurable digital input
C13 used as configurable digital transistor output

WARNING!
Risk of death by electric shock!
Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

CAUTION!
Risk of damaging the processor module!
- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be fed from the same phase.
- Use an external fuse to protect the outputs.

I/O configuration

The configuration data of the onboard I/Os is stored in the processor modules PM50x2. See PLC configuration: *Chapter 1.6.6.2.5 “Configure the onboard I/O channel” on page 3700*

Parameterization

For information about parameterization, refer to the description for onboard I/Os for processor modules PM50x2. See PLC configuration: *“PM5012-x-ETH Basic CPU” on page 3702* and *“PM5032-x-ETH, PM5052-x-ETH Standard CPU” on page 3703*

Diagnosis

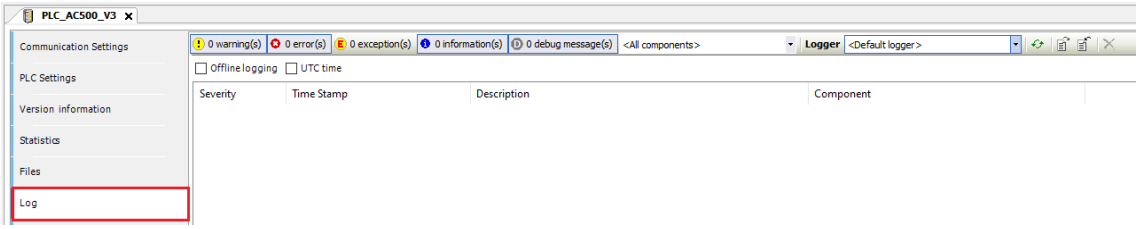
No diagnosis is generated for the onboard I/O.

There is only an error message if the configuration does not work. A log entry is generated.

The Automation Builder already prevents faulty values from being entered in the configuration.

If the configuration does not work, there is a system error, if e.g. faulty software or wrong versions are installed.

Otherwise there are error messages from the blocks for the individual functions.



Displays

Table 420: States of the I/Os

LED	Status	Color	LED = ON	LED = OFF
I	Digital input	yellow	Input is ON	Input is OFF
O	Digital transistor output	yellow	Output is ON	Output is OFF
NO	Digital relay output	yellow	Relay contact is closed	Relay contact is open
C	Digital configurable input/output	yellow	Configured input/ output is ON	Configured input/ output is OFF

Technical data

Technical data of the digital inputs

Parameter	Value
Number of channels per module	12
Distribution of the channels into groups	1 group of 12 channels
Galvanic isolation	Yes, per group
Connections of the channels I0 to I11	Terminals 2 to 13

Parameter		Value	
Reference potential for the channels I0 to I11		Terminal 1	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1) and the module's logic is in operation	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal voltage		-24 V DC	+24 V DC
	Signal 0	-5 V...+3 V	-3 V...+5 V
	Undefined signal	-15 V...- 5 V	+5 V...+15 V
	Signal 1	-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		Within -5 V...+3 V	Within -3 V...+5 V
Ripple with signal 1		Within -30 V...-15 V	Within +15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 4.6 mA	
	Input voltage +5 V	Typ. 0.8 mA	
	Input voltage +15 V	> 2.5 mA	
	Input voltage +30 V	< 8 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		On request	
Max. cable length *)			
	Shielded	On request	
	Unshielded	On request	

*) For fast inputs and fast outputs including PTO and PWM, a shielded cable must be used and the max. cable length is 50 m.

Technical data of the fast counter inputs

For AC500 devices the function "fast counter" is available in S500 I/O modules as of firmware version V1.3.

For AC500-eCo V3 devices the function "fast counter" is available in onboard I/Os of PM50xx.

The AC500-eCo V3 processor modules with onboard I/Os provide some special functionality on the digital inputs or digital outputs. Fast counter, encoder inputs, interrupt inputs or PWM/PTO outputs are available depending on the device used.

The fast counter functionality can be activated within the onboard I/O configuration.

The fast counter can work in pulse/direction mode or A/B track counter mode.

The pulse/direction counter detects the rising edge of the counter input. It will increase or decrease the count value (depending on the direction input) at every rising edge.

The A/B track counter is used to count the signal from an encoder.

The counter can count with quad phases. In the following the behavior of the A/B track counter is described.

Further information:

Operating modes of the fast counter: ↳ Chapter 1.6.6.2.13.9.1.2 “Operating modes” on page 3781

Configurarion of the fast counter: ↳ Chapter 1.6.6.2.7.2 “Fast counters in the onboard I/Os” on page 3710

Parameter		**PM5012-T-ETH**	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH**	**PM5032-R-ETH** **PM5052-R-ETH**
Fast counter					
	Useable inputs	2	2	4	4
	Fast input max. 5 kHz	DI4 ... DI5	DI4 ... DI5	-	-
	Fast input, max. 100 kHz	-	-	DI4 … DI7	DI4 … DI7

Technical data of the interrupt inputs

Parameter		**PM5012-T-ETH**	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH**	**PM5032-R-ETH** **PM5052-R-ETH**
Interrupt					
	Useable inputs	4	4	4	4
	Fast input max. 5 kHz	DI0 ... DI3	DI0 ... DI3	DI0 ... DI3	DI0 ... DI3

Technical data of the Touch/Reset inputs

Parameter		**PM5012-T-ETH**	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH**	**PM5032-R-ETH** **PM5052-R-ETH**
Touch/Reset					
	Useable inputs	-	-	4 together with dedicated encoder	4 together with dedicated encoder

Parameter		PM5012-T-ETH	PM5012-R-ETH	PM5032-T-ETH PM5052-T-ETH PM5072-T-2ETH	PM5032-R-ETH PM5052-R-ETH
	Fast input max. 5 kHz	-	-	DI0 ... DI3	DI0 ... DI3
	Fast input, max. 100 kHz	-	-	DI6 ... DI7 When using the A/B encoder on DI04...DI05 and the Touch/ Reset inputs on fast inputs	DI6 ... DI7 When using the A/B encoder on DI04...DI05 and the Touch/ Reset inputs on fast inputs

Technical data of the digital transistor outputs

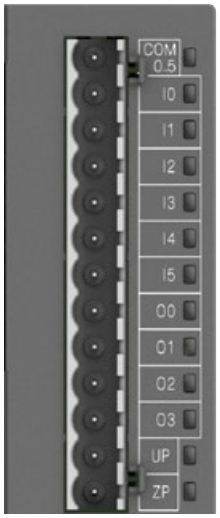


Table 421: PM5012-T-ETH

Parameter		Value
Number of channels per module		4
Distribution of the channels into groups		1 group of 4 channels
Galvanic isolation		Yes, per group
Connection of the channels O0 to O3		Terminals 8 to 11
Common power supply voltage		Terminals 12 (+24 V DC, signal name UP)
Reference potential for the channels O0 to O7		Terminal 13 (0 V DC, negative pole of the process voltage, signal name ZP)
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1)
Way of operation		Non-latching type
Min. output voltage at signal 1		UP - 0.1 V
Output delay (max. at rated load)		
	0 to 1	On request
	1 to 0	On request
Rated protection fuse (per group)		On request
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC (resistance, general use and pilot duty)
	Rated current per group (max.)	2 A
	Rated current (all channels together, max.)	2 A
Max. leakage current with signal 0		On request
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	On request
Short-circuit-proof / Overload-proof		No
Overload message		No

Parameter		Value
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Max. cable length *)		
	Shielded	On request
	Unshielded	On request

*) For fast inputs and fast outputs including PTO and PWM, a shielded cable must be used and the max. cable length is 50 m.

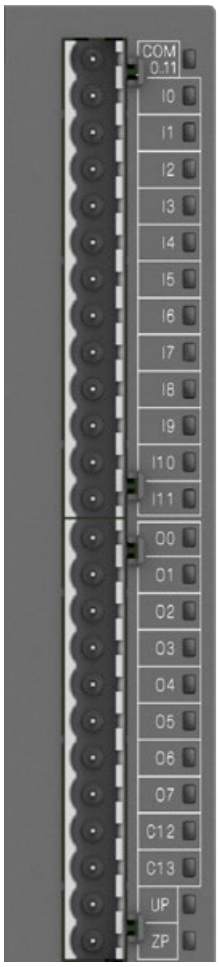


Table 422: PM5032-T-ETH, PM5072-T-2ETH and PM5072-T-2ETHW

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Galvanic isolation		Yes, per group
Connection of the channels O0 to O7		Terminals 14 to 21
Common power supply voltage		Terminals 24 (+24 V DC, signal name UP)
Reference potential for the channels O0 to O7		Terminal 25 (0 V DC, negative pole of the process voltage, signal name ZP)
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1)
Way of operation		Non-latching type
Min. output voltage at signal 1		UP - 0.1 V
Output delay (max. at rated load)		
	0 to 1	On request
	1 to 0	On request
Rated protection fuse (per group)		On request
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC (resistance, general use and pilot duty)
	Rated current per group (max.)	4 A
	Rated current (all channels together, max.)	4 A
Max. leakage current with signal 0		0.5 mA
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	On request
Short-circuit-proof / Overload-proof		No
Overload message		No
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Max. cable length *)		

Parameter		Value
	Shielded	On request
	Unshielded	On request

*) For fast inputs and fast outputs including PTO and PWM, a shielded cable must be used and the max. cable length is 50 m.

Technical data of the digital relay outputs

Table 423: PM5012-R-ETH

Parameter		Value
Number of channels per module		4 normally-open relay outputs
Distribution of the channels into groups		2 groups for 2 channels
Galvanic isolation		Yes, per group
Connection of the channels NO0 to NO1		Terminals 8 to 9
Connection of the channels NO2 to NO3		Terminals 11 to 12
Reference potential R0..1 for the channels NO0 to NO1		Terminal 10
Reference potential R2..3 for the channels NO2 to NO3		Terminal 13
Relay output voltage		
	Rated value	24 V DC or 100 V AC...240 V AC 50 Hz/60 Hz
	Range	5 V DC...30 V DC or 5 V AC...250 V AC
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1)
Way of operation		Non-latching type
Output delay		
	0 to 1	Typ. 10 ms
	1 to 0	Typ. 10 ms
Rated protection fuse		On request
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC resistance and general use, 100 V AC...240 V AC, resistance, general use and pilot duty)
	Rated current per group (max.)	6 A
	Rated current (all channels together, max.)	12 A
Demagnetization when inductive loads are switched off		External demagnetization measures must be implemented when switching inductive loads.
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching frequencies		

Parameter		Value
	With resistive loads	Max. 1 Hz
	With inductive loads	On request
	With lamp loads	On request
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
Rated protection fuse (for each channel)		On request
Overload message		No
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100,000 at rated load
Max. cable length *)		
	Shielded	On request
	Unshielded	On request

*) For fast inputs and fast outputs including PTO and PWM, a shielded cable must be used and the max. cable length is 50 m.

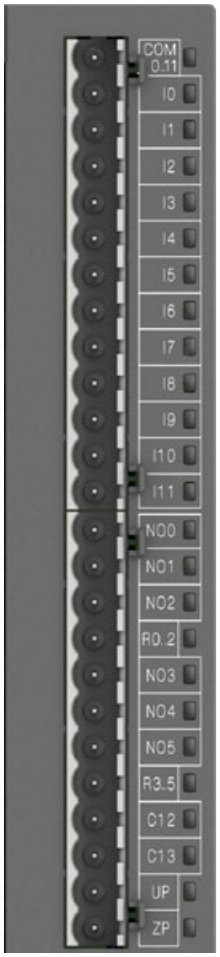


Table 424: PM5032-R-ETH and PM5052-R-ETH

Parameter		Value
Number of channels per module		6 normally-open relay outputs
Distribution of the channels into groups		2 groups for 3 channels
Galvanic isolation		Yes, per group
Connection of the channels NO0 to NO2		Terminals 14 to 16
Connection of the channels NO3 to NO5		Terminals 18 to 20
Reference potential R0..2 for the channels NO0 to NO2		Terminal 17
Reference potential R3..5 for the channels NO3 to NO5		Terminal 21
Relay output voltage		
	Rated value	24 V DC or 100 V AC...240 V AC 50 Hz/60 Hz
	Range	5 V DC...30 V DC or 5 V AC...250 V AC
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered through the I/O bus
Way of operation		Non-latching type
Output delay		
	0 to 1	Typ. 10 ms
	1 to 0	Typ. 10 ms
Rated protection fuse		On request

Parameter		Value
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC resistance and general use, 100 V AC...240 V AC, resistance, general use and pilot duty)
	Rated current per group (max.)	6 A
	Rated current (all channels together, max.)	12 A
Demagnetization when inductive loads are switched off		External demagnetization measures must be implemented when switching inductive loads.
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	On request
	With lamp loads	On request
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
Rated protection fuse (for each channel)		On request
Overload message		No
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100,000 at rated load
Max. cable length *)		
	Shielded	On request
	Unshielded	On request

*) For fast inputs and fast outputs including PTO and PWM, a shielded cable must be used and the max. cable length is 50 m.

Technical data of the limit switch outputs

Parameter		PM5012-T-ETH	PM5012-R-ETH	PM5032-T-ETH PM5052-T-ETH PM5072-T-2ETH	PM5032-R-ETH PM5052-R-ETH
Limit switch					
	Useable outputs	4	-	8	2
	Fast output max. 5 kHz	DO0 ... DO3	-	DO0 … DO3	-
	Fast output, max. 100 kHz	-	-	DO4 … DO7	DC12 … DC13

Technical data of the PTO outputs

Parameter		PM5012-T-ETH	PM5012-R-ETH	PM5032-T-ETH PM5052-T-ETH PM5072-T-2ETH	PM5032-R-ETH PM5052-R-ETH
PTO					
	Useable outputs	-	-	4	1 pair of output
	Fast output, max. 100 kHz	-	-	DO4 … DO7 For 2 PTO 200 kHz *) Pulse/ Direction or CC/Ccw modes as pair of outputs	DC12 … DC13
				DO4 … DO7 as 4 PTO 100 kHz Pulse outputs / Direction using fast output 5kHz DO0...DO3	

*) If the load is less than 100 mA it is strongly recommended to connect an additional load resistor (240 Ω/5 W or 270 Ω/5 W) to the output to improve the pulse signal.

Technical data of the PWM outputs

Parameter		PM5012-T-ETH	PM5012-R-ETH	PM5032-T-ETH PM5052-T-ETH PM5072-T-2ETH	PM5032-R-ETH PM5052-R-ETH
PWM					
	Useable outputs	4	-	8	2
	Fast output max. 5 kHz	DO0 ... DO3	-	DO0 … DO3	-
	Fast output, max. 100 kHz	-	-	DO4 … DO7	DC12 … DC13

Ordering data

Table 425: Processor modules for AC500-eCo V3

Part no.	Description	Product life cycle phase *)
1SAP 122 600 R0072	Basic CPU PM5012-T-ETH, AC500-eCo V3 processor module, programmable logic controller 1 MB, 6DI/4DO-Transistor, Ethernet, 24 V DC, 1 option board slot	Active
1SAP 122 700 R0072	Basic CPU PM5012-R-ETH, AC500-eCo V3 processor module, programmable logic controller 1 MB, 6DI/4DO-Relay, Ethernet, 24 V DC, 1 option board slot	Active
1SAP 123 400 R0072	Standard CPU PM5032-T-ETH, AC500-eCo V3 processor module, programmable logic controller 2 MB, 12DI/8DO-Transistor/2DC, Ethernet, 24 V DC, 2 option board slots	Active
1SAP 123 500 R0072	Standard CPU PM5032-R-ETH, AC500-eCo V3 processor module, programmable logic controller 2 MB, 12DI/6DO-Relay/2DC, Ethernet, 24 V DC, 2 option board slots	Active
1SAP 124 000 R0072	Standard CPU PM5052-T-ETH, AC500-eCo V3 processor module, programmable logic controller 4 MB, 12DI/8DO-Transistor/2DC, Ethernet, 24 V DC, 3 option board slots	Active
1SAP 124 100 R0072	Standard CPU PM5052-R-ETH, AC500-eCo V3 processor module, programmable logic controller 4 MB, 12DI/6DO-Relay/2DC, Ethernet, 24 V DC, 3 option board slots	Active
1SAP 124 500 R0073	Pro CPU PM5072-T-2ETH, AC500-eCo V3 processor module, programmable logic controller 8 MB, 12DI/8DO-Transistor/2DC, 2 Ethernet, 24 V DC, 3 option board slots	Active
1SAP 124 400 R0073	Pro CPU PM5072-T-2ETHW, AC500-eCo V3 processor module, programmable logic controller 8 MB, 12DI/8DO-Transistor/2DC, 2 Ethernet, 24 V DC, 3 option board slots, wide temperature	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 426: Accessories for AC500-eCo V3

Part no.	Description
1SAP 187 000 R0001	TA5101-4DI: AC500, option board for digital I/O extension, 4DI 24 V DC, spring/cable front terminal 3.50 mm pitch
1SAP 187 000 R0002	TA5105-4DOT: AC500, option board for digital I/O extension, 4DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch
1SAP 187 000 R0003	TA5110-2DI2DOT: AC500, option board for digital I/O extension, 2DI 24 V DC, 2DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch
1SAP 187 200 R0001	TA5130-KNXPB: AC500, option board KNX adress push button
1SAP 187 200 R0002	TA5131-RTC:AC500, real-time clock without battery, option board for AC500-eCo V3 Basic CPU
1SAP 187 300 R0001	TA5141-RS232I: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 300 R0002	TA5142-RS485I: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 300 R0003	TA5142-RS485: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 400 R0001	TA5211-TSCL-B: screw terminal block set for AC500-eCo V3 CPU Basic screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors
1SAP 187 400 R0002	TA5211-TSPF-B: spring terminal block set for AC500-eCo V3 CPU Basic spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors
1SAP 187 400 R0004	TA5212-TSCL: screw terminal block set for AC500-eCo V3 Standard and Pro CPU screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors
1SAP 187 400 R0005	TA5212-TSPF: spring terminal block set for AC500-eCo V3 Standard and Pro CPU spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors
1SAP 187 600 R0001	TA5400-SIM: input simulator (for CPU testing), 6 switches
1SAP 180 100 R0002	MC5102 - Micro memory card with memory card adapter
1SAP 182 800 R0001	TA543: screw mounting accessory, 20 pieces per packing unit
1SAP 187 500 R0003	TA5301-CFA: cable fixing part accessory, 20 pieces per packing unit
Spare parts	
1SAP 187 400 R0012	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 6 pieces per packing unit

Part no.	Description
1SAP 187 400 R0013	TA5220-SPF6: spring terminal block, removable, 6-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0014	TA5220-SPF7: spring terminal block, removable, 7-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0015	TA5220-SPF8: spring terminal block, removable, 8-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 500 R0001	TA5300-CVR: option board slot cover, removable plastic part, 6 pieces per packing unit

Technical data

The system data of AC500-eCo V3 apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

General data

Parameter		Value			
		PM5012	PM5032	PM5052	PM5072
Power supply		24 V DC			
Connection of power supply		Via removable 3-pin terminal			
Current consumption from power supply (max.)					
	Transistor version	200 mA	340 mA	400 mA	420 mA
	Relay version	200 mA	340 mA	400 mA	-
Inrush current at nominal voltage		On request			
Required fuse		On request			
Max. power dissipation within the processor module					
	Transistor version	On request	On request	On request	On request
	Relay version	On request	On request	On request	-
Processor module interfaces		RS485/RS232 (optional), Ethernet			
		-	I/O bus		
Weight					
	Transistor version	300 g	400 g	400 g	400 g
	Relay version	400 g	400 g	400 g	400 g
Mounting position		Horizontal or vertical			

Detailed data

Parameter		Value			
		PM5012	PM5032	PM5052	PM5072
Total maximum downloadable application size [1)]		1 MB	5 MB	7 MB	9 MB
	Thereof user program code / data memory dynamically allocated	256 kB	512 kB	768 kB	1 MB

Parameter		Value			
		PM5012	**PM5032**	**PM5052**	**PM5072**
	Thereof user web server memory for web visualization max.	no web	1.5 MB	3.2 MB	7 MB
	User data memory saved in FLASH	8 kB	32 kB	100 kB	
	VAR_RETAIN persistent	4 kB	16 kB	36 kB	
	%MB data	4 kB	16 kB	64 kB	
Data buffering		FRAM memory without battery			
Real-time clock (RTC) (no battery, supercap)		Optional with TA5131-RTC	Built in		
Min. retention time for RTC / accuracy in s/day		On request	On request	On request	On request
Programming languages		● Instruction List (IL) ● Function Block Diagram (FBD) ● Ladder Diagram (LD) ● Sequential Function Chart (SFC) ● Structured Text (ST) ● Continuous Function Chart (CFC)			
Cycle time per instructions (minimum)		PM5012	PM5032	PM5052	PM5072
	Binary	20 ns			
	Word	50 ns			
	Floating point	600 ns			
Program execution		PM5012	PM5032	PM5052	PM5072
	Cyclic min. configurable	10 ms	5 ms	2 ms	1 ms
	Time-controlled	Yes			
	Multitasking	Yes			
	Interruption	Yes			
LEDs		Power, Error, Run, MC, MOD1, States of I/Os			
RUN/STOP button		Yes			
Protection of the user program by password		On request			
Usable accessories		On request			
Remarks: [1]: The values are for information only and cannot be fulfilled altogether. The available resources are limited at the end by the maximal downloadable application size for each CPU.					

Data of I/Os	**PM5012-x-ETH**	**PM5032-x-ETH**	**PM5052-x-ETH**	**PM5072-T-2ETH**
Onboard digital inputs				
Channels	6 (incl. 2 counter inputs 5 kHz and 4 interrupts)	12 (incl. 4 fast counter/encoder inputs (100 kHz/200 kHz), 4 counter inputs (5 kHz), 4 standard inputs)		
Signal voltage	24 V DC type 1			
Onboard digital outputs				

Data of I/Os	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
Type of digital outputs	PM5012-T-ETH: Transistor	PM5032-T-ETH: Transistor	PM5052-T-ETH: Transistor	PM5072-T-2ETH: Transistor
	PM5012-R-ETH: Relay	PM5032-R-ETH: Relay	PM5052-R-ETH: Relay	-
Channels for transistor version	4 (5 kHz standard and PWM)	8 (incl. 4 fast outputs for standard or 4 PWM/2 PTO (100 kHz/200 kHz), 4 standard outputs (5 kHz))		
Channels digital input/output configurable (valid for both PLC version relais or transistor)	-	2 Relay version: The DC channels can be used as 1 PTO/2 PWM (100 kHz) or standard digital inputs/outputs Transistor version: The DC channels can only be used as standard digital inputs/outputs		2 Transistor version: The DC channels can only be used as standard digital inputs/outputs
Rated voltage transistor	24 V DC			
Nominal current per transistor channel	0.5 A resistive			
Channels for relay version	4	6		-
Rated voltage relay	100 V AC...240 V AC or 24 V DC			-
Nominal current per relay channel	2 A resistive			-
Analog inputs	Optional			
Analog outputs	Optional			
Number of option board slots	1	2	3	3
Usage of option board	Each slot can be used for all type of existing option boards, same option board for serial interface or digital/analog I/O extension can be used on several slot per CPU. Note: RTC option board is only for PM5012 possible.			
KNX address switch	No			TA5130-KNXPB only on 1 slot
Real-time clock (RTC)	TA5131-RTC	No		
Serial interface	TA5141-RS232I, TA5142-RS485/TA5142-RS485I			
Digital in/out channels	TA5101-4DI, TA5105-4DOT, TA5110-2DI2DOT			
Analog in/out channels	TA5120-2AI-UI, TA5122-2AI-TC, TA5123-2AI-RTD, TA5126-2AO-UI			
Max. number of I/O modules on I/O bus	0	10		

Data of I/Os	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
Digital inputs	Onboard I/O only	128 B	1 kB	
Digital outputs		128 B	1 kB	
Number of decentralized inputs and outputs	Depending on the fieldbus used			
Internal interfaces				
Serial COMx	Optional, use a dedicated serial interface option board (up to 1)	Optional, use a dedicated serial interface option board (up to 2)	Optional, use a dedicated serial interface option board (up to 3)	
	Modbus RTU Master/Slave, ASCII			
Ethernet interface RJ45	1			2 Independent with switch functionality
Ethernet functions	Programming, TCP/IP, UDP/IP, DHCP, PING, network variables, and other listed below			
Modbus TCP/IP client/server	Yes 8 / 3	Yes 13 / 8	Yes 20 / 10	Yes 30 / 15
SNTP client/ server	No	Yes		
HTTPs and Web-Visu number of connections	No	Yes 1	Yes 2	Yes 4
FTPs number of connections	No	Yes 1	Yes 2	
OPC UA server number of free tags	No	Yes 125	Yes 250	Yes 1000
MQTT and JSON library	No	Yes		
OPC DA server	Yes			
IEC 60870-5-104 telecontrol protocol	No			Yes Substation only, 5 connections max., only 1 Ethernet supported
Licensed protocols (runtime protocol per CPU)				
BACnet IP B-BC ↳ *Chapter 1.5.5 “BACnet-BC” on page 2209*	No			Yes (max. 1000 object variables)

Data of I/Os	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
KNXIP ↳ *Chapter 1.6.6.3.8 “KNX configurator” on page 3924*	No			Yes (max. 1000 object variables)
IEC 61850 MMS server/goose pub/sub	No			Yes (max. 1000 data attributes)
EtherNet/IP adapter/scanner	No	Yes (in preparation)		

Ordering Data

Table 427: Processor modules for AC500-eCo V3

Part no.	Description	Product life cycle phase *)
1SAP 122 600 R0072	Basic CPU PM5012-T-ETH, AC500-eCo V3 processor module, programmable logic controller 1 MB, 6DI/4DO-Transistor, Ethernet, 24 V DC, 1 option board slot	Active
1SAP 122 700 R0072	Basic CPU PM5012-R-ETH, AC500-eCo V3 processor module, programmable logic controller 1 MB, 6DI/4DO-Relay, Ethernet, 24 V DC, 1 option board slot	Active
1SAP 123 400 R0072	Standard CPU PM5032-T-ETH, AC500-eCo V3 processor module, programmable logic controller 2 MB, 12DI/8DO-Transistor/2DC, Ethernet, 24 V DC, 2 option board slots	Active
1SAP 123 500 R0072	Standard CPU PM5032-R-ETH, AC500-eCo V3 processor module, programmable logic controller 2 MB, 12DI/6DO-Relay/2DC, Ethernet, 24 V DC, 2 option board slots	Active
1SAP 124 000 R0072	Standard CPU PM5052-T-ETH, AC500-eCo V3 processor module, programmable logic controller 4 MB, 12DI/8DO-Transistor/2DC, Ethernet, 24 V DC, 3 option board slots	Active
1SAP 124 100 R0072	Standard CPU PM5052-R-ETH, AC500-eCo V3 processor module, programmable logic controller 4 MB, 12DI/6DO-Relay/2DC, Ethernet, 24 V DC, 3 option board slots	Active

Part no.	Description	Product life cycle phase *)
1SAP 124 500 R0073	Pro CPU PM5072-T-2ETH, AC500-eCo V3 processor module, programmable logic controller 8 MB, 12DI/8DO-Transistor/2DC, 2 Ethernet, 24 V DC, 3 option board slots	Active
1SAP 124 400 R0073	Pro CPU PM5072-T-2ETHW, AC500-eCo V3 processor module, programmable logic controller 8 MB, 12DI/8DO-Transistor/2DC, 2 Ethernet, 24 V DC, 3 option board slots, wide temperature	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 428: Accessories for AC500-eCo V3

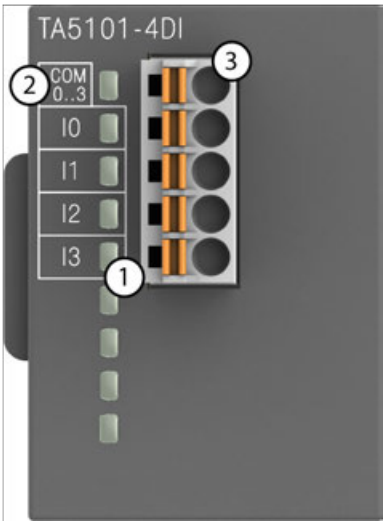
Part no.	Description
1SAP 187 000 R0001	TA5101-4DI: AC500, option board for digital I/O extension, 4DI 24 V DC, spring/cable front terminal 3.50 mm pitch
1SAP 187 000 R0002	TA5105-4DOT: AC500, option board for digital I/O extension, 4DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch
1SAP 187 000 R0003	TA5110-2DI2DOT: AC500, option board for digital I/O extension, 2DI 24 V DC, 2DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch
1SAP 187 200 R0001	TA5130-KNXPB: AC500, option board KNX adress push button
1SAP 187 200 R0002	TA5131-RTC:AC500, real-time clock without battery, option board for AC500-eCo V3 Basic CPU
1SAP 187 300 R0001	TA5141-RS232I: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 300 R0002	TA5142-RS485I: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 300 R0003	TA5142-RS485: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 400 R0001	TA5211-TSCL-B: screw terminal block set for AC500-eCo V3 CPU Basic screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors
1SAP 187 400 R0002	TA5211-TSPF-B: spring terminal block set for AC500-eCo V3 CPU Basic spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors

Part no.	Description
1SAP 187 400 R0004	TA5212-TSCL: screw terminal block set for AC500-eCo V3 Standard and Pro CPU screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors
1SAP 187 400 R0005	TA5212-TSPF: spring terminal block set for AC500-eCo V3 Standard and Pro CPU spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors
1SAP 187 600 R0001	TA5400-SIM: input simulator (for CPU testing), 6 switches
1SAP 180 100 R0002	MC5102 - Micro memory card with memory card adapter
1SAP 182 800 R0001	TA543: screw mounting accessory, 20 pieces per packing unit
1SAP 187 500 R0003	TA5301-CFA: cable fixing part accessory, 20 pieces per packing unit
Spare parts	
1SAP 187 400 R0012	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0013	TA5220-SPF6: spring terminal block, removable, 6-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0014	TA5220-SPF7: spring terminal block, removable, 7-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0015	TA5220-SPF8: spring terminal block, removable, 8-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 500 R0001	TA5300-CVR: option board slot cover, removable plastic part, 6 pieces per packing unit

Option boards

TA5101-4DI - Option board for digital I/O extension

- 4 digital inputs 24 V DC (I0 to I3) in 1 group
- Module-wise galvanically isolated



1 4 yellow LEDs to display the signal states of the inputs I0 to I3
2 Allocation of signal name
3 5-pin terminal block for input signals

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with "NC"). Reserved terminals may carry internal voltages.

Intended purpose

The device is used as an optional I/O extension module for AC500-eCo V3 CPUs (PM50x2).

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via internal CPU connection
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the system assembly chapter.

The connection is carried out by using a removable 5-pin terminal block. For more information, please refer to the chapter terminal blocks for AC500-eCo V3 system. The terminal blocks are included in the module's scope of delivery and additional terminal blocks as spare parts can be ordered separately.

The following block diagram shows the internal construction of the digital inputs:

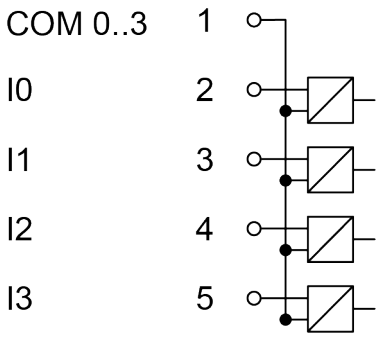


Table 429: Assignment of the terminals:

Terminal	Signal	Description
1	COM 0..3	Input common for signals I0 to I3
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3

The internal power supply voltage for the module's circuitry is carried out via the connection to CPU. Thus, the current consumption from 24 V DC power supply at the terminals L+ and M of the CPU module increases by 10 mA per TA5101-4DI.

An external power supply connection is not needed.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with "NC"). Reserved terminals may carry internal voltages.

The digital inputs can be used as source inputs or as sink inputs.

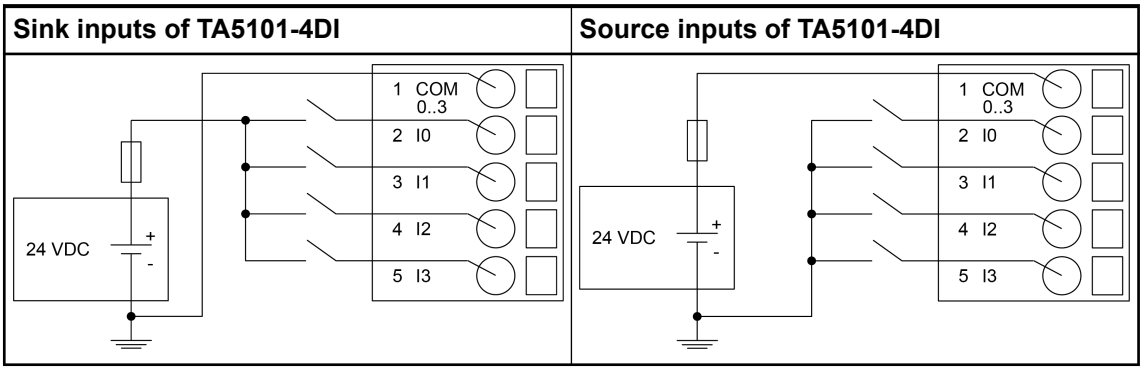
NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the option board for digital I/O extension TA5101-4DI:



The module provides several diagnosis functions, see Diagnosis *“Diagnosis” on page 2482*.

The meaning of the LEDs is described in the section State LEDs *“State LEDs” on page 2482*.

I/O configuration

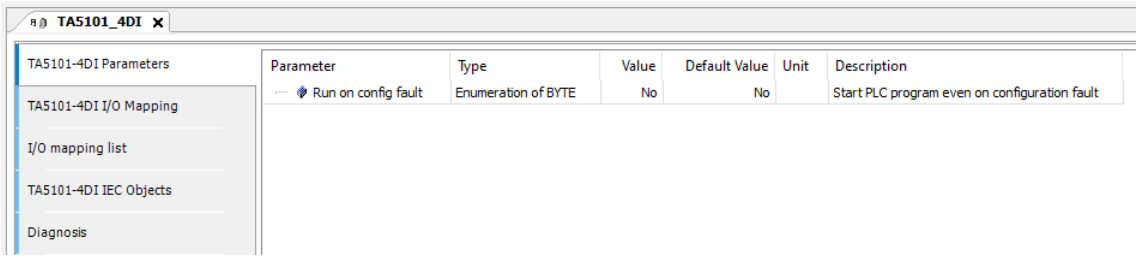
The module itself does not store configuration data. It receives its parameterization data from the CPU module during power-up of the system.

Hence, replacing optional modules is possible without any re-parameterization via software.

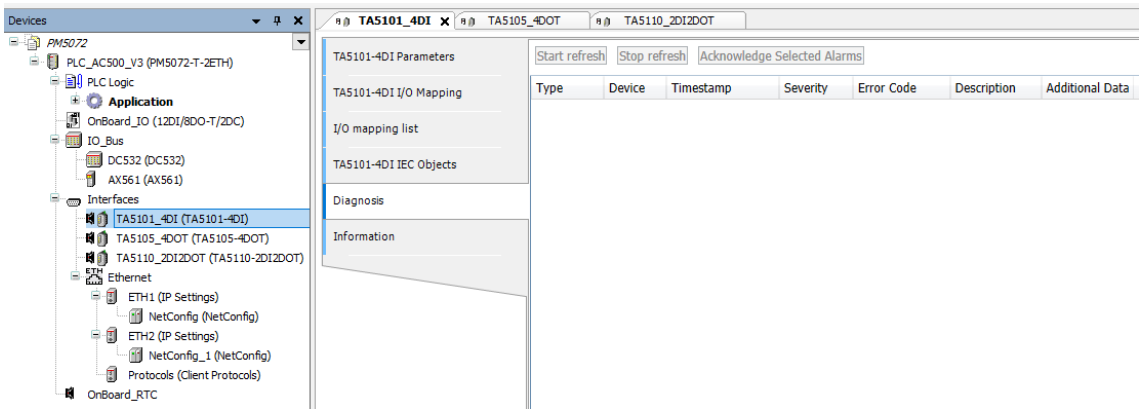
Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *“TA51xx Parameters”* tab to edit the parameterization of the desired option board.



Diagnosis



1. In the device tree, double-click the desired option board.
2. Select the *“Diagnosis”* tab to view the diagnosis messages of the desired option board.

Table 430: Diagnosis messages

Device	Severity	Error code	Description	
			Error Message	Remedy
TA5101-4DI	11	1	Wrong or no board plugged	Replace with correct functional board
TA5101-4DI	11	2	Board defective	Replace with correct functional board
TA5101-4DI	11	3	Failed to set direction	Replace with correct functional board
TA5101-4DI	11	4	Parameter wrong	Verify setting of parameter “Run on config fault”

State LEDs

LED	State	Color	LED = OFF	LED = ON
Inputs I0...I3	Digital input	Yellow	Input is OFF	Input is ON

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input group and the rest of the module
Isolated groups	1 (4 channels per group)
Current consumption from 24 V DC power supply at the L+ and M terminals of the CPU	Ca. 10 mA
Max. power dissipation within the module	0.8 W
Weight	15 g

Parameter	Value
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Table 431: Technical data of the digital inputs

Parameter		Value	
Number of channels per module		4 inputs 24 V DC	
Distribution of the channels into groups		1 (4 channels per group)	
Connections of the channels I0 to I3		Terminals 2 to 5	
Reference potential for the channels I0 to I3		Terminal 1 (plus or negative pole of the process supply voltage, signal name COM 0..3)	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the CPU connection.	
Monitoring point of input indicator		LED	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
	Signal 0	-5 V...+3 V	-3 V...+5 V
	Undefined signal	-15 V...-5 V	+5 V...+15 V
	Signal 1	-30 V...-15 V	+15 V...+30 V
Input current per channel			
	Input voltage 24 V	Typ. 5 mA	
	Input voltage 5 V	Typ. 1 mA	
	Input voltage 14 V		
	Input voltage 15 V	< 3 mA	
	Input voltage 27 V		
	Input voltage 30 V	< 7 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 8 ms	
Input data length		1 byte	
Max. cable length			
	Shielded	On request	
	Unshielded	On request	

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 000 R0001	TA5101-4DI: AC500, option board for digital I/O extension, 4DI 24 V DC, spring/cable front terminal 3.50 mm pitch	Active
Spare parts		
1SAP 187 400 R0012 **)	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 6 pieces per packing unit	Active

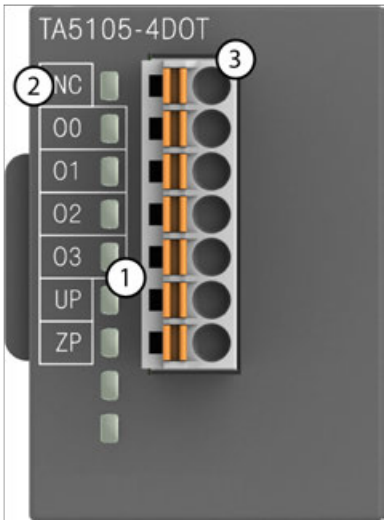
*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

**) *The needed spring terminal block is always delivered with the option board.*

The terminal block listed in the table is for spare part only if needed.

TA5105-4DOT - Option board for digital I/O extension

- 4 digital outputs 24 V DC (O0 to O3) in 1 group
- Module-wise galvanically isolated



1 4 yellow LEDs to display the signal states of the inputs O0 to O3
2 Allocation of signal name
3 7-pin terminal block for output signals

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with "NC"). Reserved terminals may carry internal voltages.

Intended purpose

The device is used as an optional I/O extension module for AC500-eCo V3 CPUs (PM50x2).

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via internal CPU connection
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the system assembly chapter.

The connection is carried out by using a removable 7-pin terminal block. For more information, please refer to the chapter terminal blocks for AC500-eCo V3 system. The terminal blocks are included in the module's scope of delivery and additional terminal blocks as spare parts can be ordered separately.

The following block diagram shows the internal construction of the digital outputs:

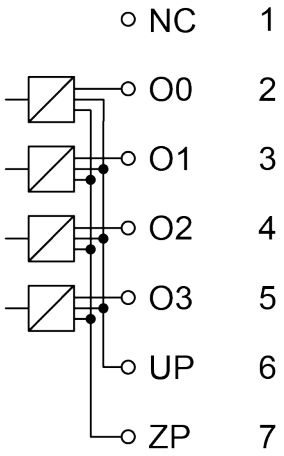


Table 432: Assignment of the terminals:

Terminal	Signal	Description
1	NC	Not connected
2	O0	Output signal O0
3	O1	Output signal O1

Terminal	Signal	Description
4	O2	Output signal O2
5	O3	Output signal O3
6	UP	Process supply voltage UP +24 V DC
7	ZP	Process supply voltage ZP 0 V DC

The internal power supply voltage for the module's circuitry is carried out via the connection to CPU. Thus, the current consumption from 24 V DC power supply at the terminals L+ and M of the CPU module increases by 10 mA per TA5105-4DOT.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

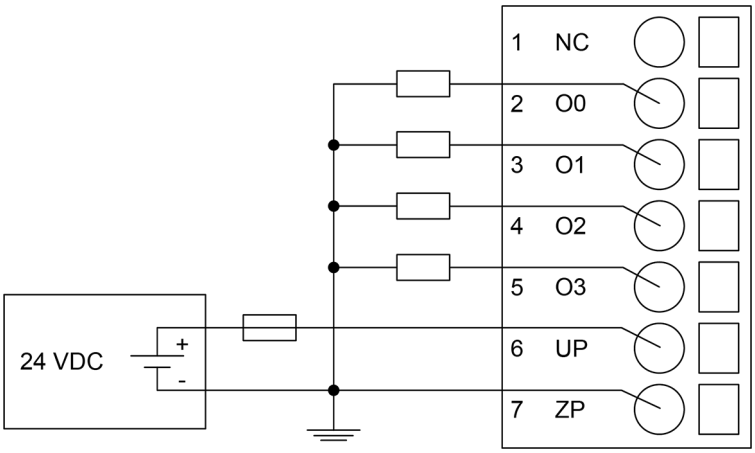
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with "NC"). Reserved terminals may carry internal voltages.

The following figure shows the connection of the option board for digital I/O extension TA5105-4DOT:



NOTICE!

Risk of malfunctions in the plant!

Only if L+/M of the CPU is available and the outputs are already configured in the AB program, the outputs will switch on as soon as the UP/ZP is available.

This must be considered in the application planning.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external fuse for the outputs.

The module provides several diagnosis functions, see Diagnosis *“Diagnosis” on page 2488*.

The meaning of the LEDs is described in the section State LEDs *“State LEDs” on page 2488*.

I/O configuration

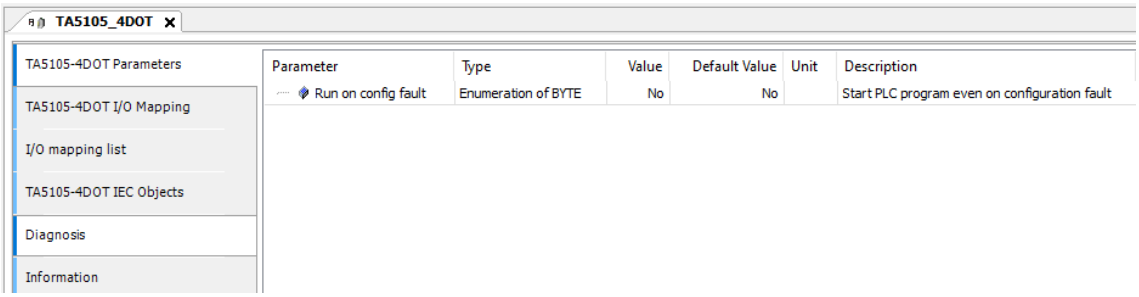
The module itself does not store configuration data. It receives its parameterization data from the CPU module during power-up of the system.

Hence, replacing optional modules is possible without any re-parameterization via software.

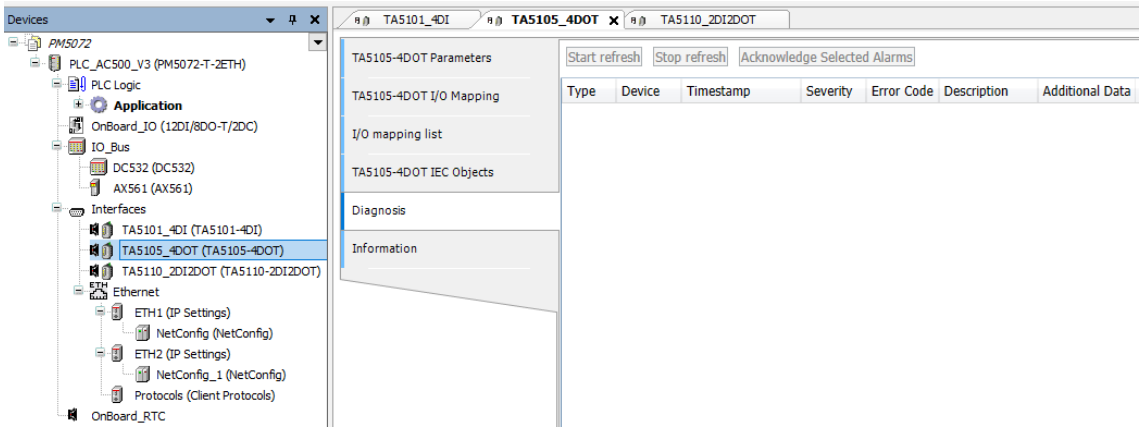
Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *“TA51xx Parameters”* tab to edit the parameterization of the desired option board.



Diagnosis



1. In the device tree, double-click the desired option board.
2. Select the *“Diagnosis”* tab to view the diagnosis messages of the desired option board.

Table 433: Diagnosis messages

Device	Severity	Error code	Description	
			Error Message	Remedy
TA5105-4DOT	11	1	Wrong or no board plugged	Replace with correct functional board
TA5105-4DOT	11	2	Board defective	Replace with correct functional board
TA5105-4DOT	11	3	Failed to set direction	Replace with correct functional board
TA5105-4DOT	11	4	Parameter wrong	Verify setting of parameter “Run on config fault”

State LEDs

LED	State	Color	LED = OFF	LED = ON
Outputs O0...O3	Digital output	Yellow	Output is OFF	Output is ON (The output voltage (normally 24 V DC) is only displayed if UP/ZP and L+/M (supply voltages for the module) are switched ON)

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 6 for UP (+24 V DC) and terminal 7 for ZP (0 V DC)

Parameter		Value
	Rated value	24 V DC
	Current consumption via UP terminal	5 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	On request
Current consumption from 24 V DC power supply at the L+/M terminals of the CPU		Ca. 10 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		1 (4 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		0.5 W
Weight		16 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Table 434: Technical data of the digital outputs

Parameter		Value
Number of channels per module		4 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups		1 (4 channels per group)
Connection of the channels O0 to O3		Terminals 2 to 5
Common power supply voltage		Terminal 6 (positive pole of the process voltage, signal name UP)
Reference potential for the channels O0 to O3		Terminal 7 (negative pole of the process voltage, signal name ZP)
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1). Only internal logic is powered from CPU. Outputs are powered from UP/ZP terminals.
Way of operation		Non-latching type
Min. output voltage at signal 1		UP - 0.1 V
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC (resistance, general use and pilot duty)
	Rated current per group (max.)	2 A (4 channels * 0.5 A)
Max. leakage current with signal 0		0.5 mA

Parameter		Value
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		On request
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	On request
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	On request
	Unshielded	On request

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 000 R0002	TA5105-4DOT: AC500, option board for digital I/O extension, 4DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch	Active
Spare parts		
1SAP 187 400 R0014 **)	TA5220-SPF7: spring terminal block, removable, 7-pin, spring front, cable front, 6 pieces per packing unit	Active

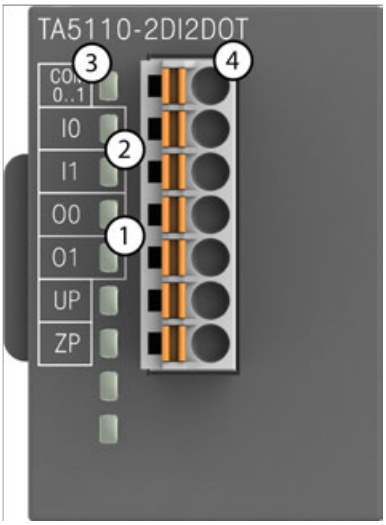
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

***) The needed spring terminal block is always delivered with the option board.*

The terminal block listed in the table is for spare part only if needed.

TA5110-2DI2DOT - Option board for digital I/O extension

- 2 digital inputs 24 V DC (I0 to I1) in 1 group
- 2 digital transistor outputs 24 V DC (O0 to O1) in 1 group
- Group-wise galvanically isolated



1 2 yellow LEDs to display the signal states of the outputs O0 to O1
2 2 yellow LEDs to display the signal states of the inputs I0 to I1
3 Allocation of signal name
4 7-pin terminal block for input/output signals

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with "NC"). Reserved terminals may carry internal voltages.

Intended purpose

The device is used as an optional I/O extension module for AC500-eCo V3 CPUs (PM50x2).

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via internal CPU connection
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the system assembly chapter.

The connection is carried out by using a removable 7-pin terminal block. For more information, please refer to the chapter terminal blocks for AC500-eCo V3 system. The terminal blocks are included in the module's scope of delivery and additional terminal blocks as spare parts can be ordered separately.

The following block diagram shows the internal construction of the digital inputs and outputs:

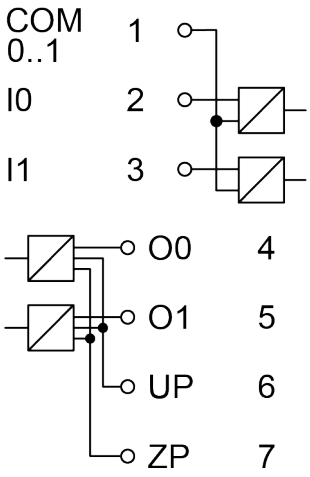


Table 435: Assignment of the terminals:

Terminal	Signal	Description
1	COM 0..1	Input common for signals I0 to I1
2	I0	Input signal I0
3	I1	Input signal I1
4	O0	Output signal O0
5	O1	Output signal O1
6	UP	Process supply voltage UP +24 V DC
7	ZP	Process supply voltage ZP 0 V DC

The internal power supply voltage for the module's circuitry is carried out via the connection to CPU. Thus, the current consumption from 24 V DC power supply at the terminals L+ and M of the CPU module increases by 10 mA per TA5110-2DI2DOT.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with "NC"). Reserved terminals may carry internal voltages.

The digital inputs can be used as source inputs or as sink inputs.

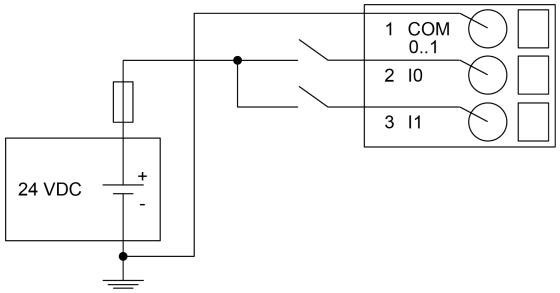
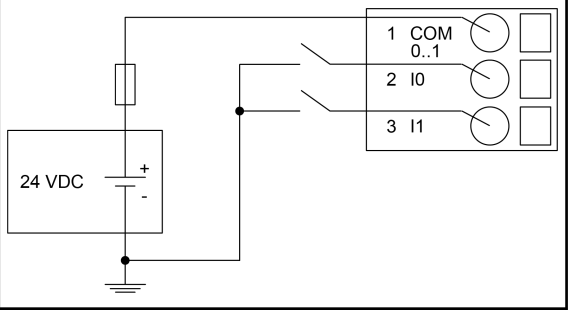
NOTICE!

Risk of malfunctions in the plant!

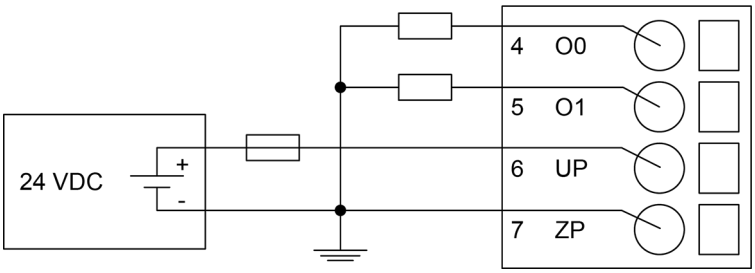
A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection for inputs of the option board for digital I/O extension TA5110-2DI2DOT:

Sink inputs of TA5110-2DI2DOT	Source inputs of TA5110-2DI2DOT
	

The following figure shows the connection for outputs of the option board for digital I/O extension TA5110-2DI2DOT:



NOTICE!

Risk of malfunctions in the plant!

Only if L+/M of the CPU is available and the outputs are already configured in the AB program, the outputs will switch on as soon as the UP/ZP is available.

This must be considered in the application planning.

NOTICE!
Risk of damaging the I/O module!
The outputs are not protected against short circuits and overload.
– Never short-circuit or overload the outputs.
– Never connect the outputs to other voltages.
– Use an external fuse for the outputs.

The module provides several diagnosis functions, see Diagnosis *“Diagnosis” on page 2494*.

The meaning of the LEDs is described in the section State LEDs *“State LEDs” on page 2495*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the CPU module during power-up of the system.

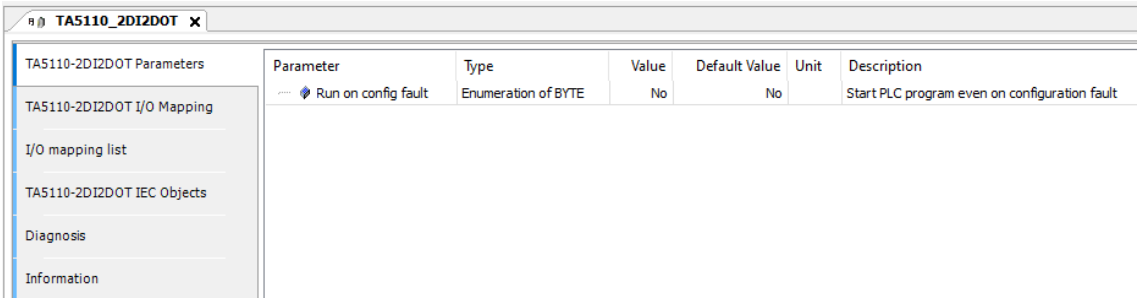
Hence, replacing optional modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

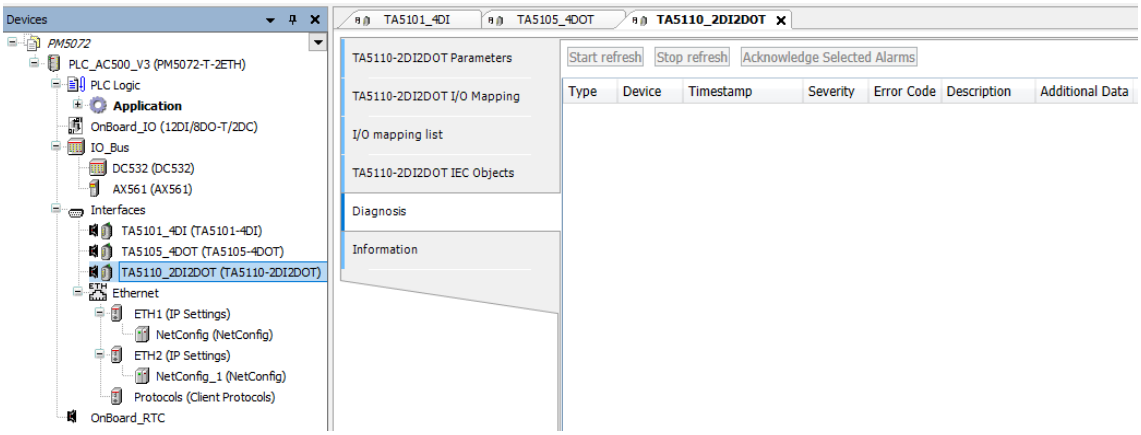
Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *“TA51xx Parameters”* tab to edit the parameterization of the desired option board.



Diagnosis



1. In the device tree, double-click the desired option board.
2. Select the *“Diagnosis”* tab to view the diagnosis messages of the desired option board.

Table 436: Diagnosis messages

Device	Severity	Error code	Description	
			Error Message	Remedy
TA5110-2DI2DOT	11	1	Wrong or no board plugged	Replace with correct functional board
TA5110-2DI2DOT	11	2	Board defective	Replace with correct functional board
TA5110-2DI2DOT	11	3	Failed to set direction	Replace with correct functional board
TA5110-2DI2DOT	11	4	Parameter wrong	Verify setting of parameter “Run on config fault”

State LEDs

LED	State	Color	LED = OFF	LED = ON
Inputs I0...I1	Digital input	Yellow	Input is OFF	Input is ON
Outputs O0...O1	Digital output	Yellow	Output is OFF	Output is ON

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 6 for UP (+24 V DC) and terminal 7 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	5 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	On request
Current consumption from 24 V DC power supply at the L+/M terminals of the CPU		Ca. 10 mA
Galvanic isolation		Yes, between the input group and the output group and the rest of the module
Isolated groups		2 groups (1 group for 2 input channels, 1 group for 2 output channels)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		0.7 W
Weight		15 g

Parameter	Value
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Table 437: Technical data of the digital inputs

Parameter		Value	
Number of channels per module		2	
Distribution of the channels into groups		1 group for 2 channels	
Connections of the channels I0 to I1		Terminals 2 to 3	
Reference potential for the channels I0 to I1		Terminal 1	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)	
Monitoring point of input indicator		LED It is not part of input circuit (its controlled by processor side, not process side)	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
Signal 0		-5 V...+3 V	-3 V...+5 V
Undefined signal		-15 V...+ 5 V	+5 V...+15 V
Signal 1		-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		-5 V...+3 V	-3 V...+5 V
Ripple with signal 1		-30 V...-15 V	+15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 5 mA	
	Input voltage +5 V	Typ. 1 mA	
	Input voltage +15 V	< 3 mA	
	Input voltage +30 V	< 7 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 8 ms	
Input data length		1 byte	
Max. cable length			
	Shielded	On request	
	Unshielded	On request	

Table 438: Technical data of the digital outputs

Parameter	Value
Number of channels per module	2 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 group of 2 channels

Parameter		Value
Connection of the channels O0 to O1		Terminals 4 to 5
Reference potential for the channels O0 to O17		Terminal 7 (negative pole of the process voltage, name ZP)
Common power supply voltage		Terminal 6 (positive pole of the process voltage, name UP)
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Monitoring point of output indicator		Controlled together with transistor
Way of operation		Non-latching type
Min. output voltage at signal 1		UP - 0.1 V
Output delay		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC (resistance, general use and pilot duty)
	Rated current per group (max.)	1 A
	Rated current (all channels together, max.)	1 A
	Max. leakage current with signal 0	0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		On request
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	On request
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	On request
	Unshielded	On request

Ordering data

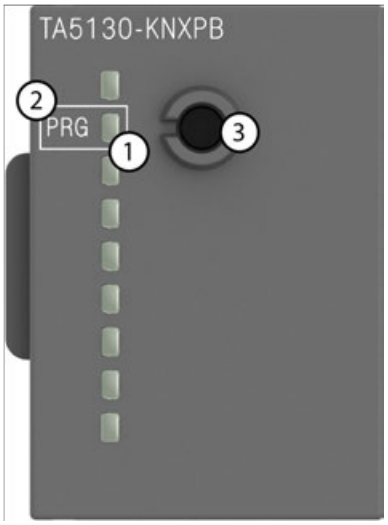
Part no.	Description	Product life cycle phase *)
1SAP 187 000 R0003	TA5110-2DI2DOT: AC500, option board for digital I/O extension, 2DI 24 V DC, 2DO-T 24 V DC / 0.5 A, spring/ cable front terminal 3.50 mm pitch	Active
Spare parts		
1SAP 187 400 R0014 **)	TA5220-SPF7: spring terminal block, removable, 7-pin, spring front, cable front, 6 pieces per packing unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

***) The needed spring terminal block is always delivered with the option board.*

The terminal block listed in the table is for spare part only if needed.

TA5130-KNXPB - Option board KNX adress push button



1 State LED
2 Allocation of signal name
3 Connector

For more information about TA5130-KNXPB, please refer to the Automation Builder online help.

Intended purpose

This option board is only intended to be used with PM5072-T-2ETH(W).

This option board can only be used once on one slot at a time!

The option board is not supported by other AC500-eCo V3 PLCs.

Information can be found in the chapter system technology: see ↳ Chapter 1.6.5.1.9 “KNX IP integration” on page 3527

Functionality

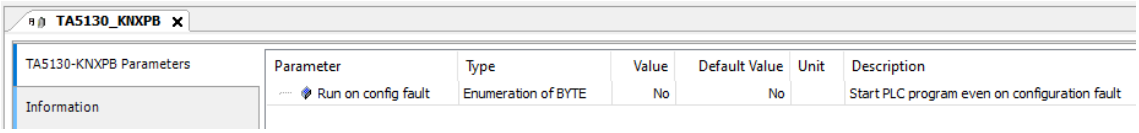
Information can be found in the chapter system technology: see ↳ Chapter 1.6.5.1.9 “KNX IP integration” on page 3527

Information about the integration of the PLC in KNX can be found here: ↳ “AC500-eCo V3 via TA5130-KNXPB” on page 3540

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *“TA51xx Parameters”* tab to edit the parameterization of the desired option board.



State LEDs

Signal	Color	State	Description
PRG	Red	ON	Programming state

Technical data

The system data of AC500-eCo V3 apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Usable CPUs	PM5072-T-2ETH(W)
Internal power supply	Via internal CPU connection
Additional current consumption from 24 V DC power supply at CPU	Max. 25 mA
Weight	14 g

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 200 R0001	TA5130-KNXPB: AC500, option board KNX adress push button	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA5131-RTC - Option board for real-time clock

1 TA5131-RTC option board

Intended purpose

This option board is only for the basic CPUs PM5012-T-ETH and PM5012-R-ETH.

All other AC500-eCo V3 CPUs have the real-time clock already integrated.

Information can be found in the chapter system technology: see ↳ Chapter 1.6.5.1.4 “Real-time clock and battery” on page 3478

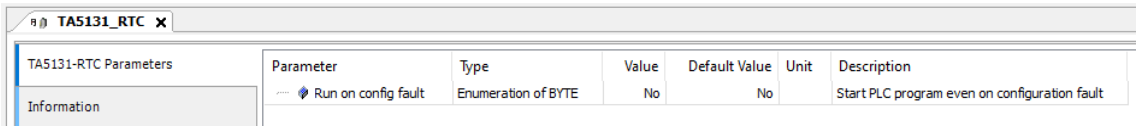
Functionality

Information can be found in the chapter system technology: see ↳ Chapter 1.6.5.1.4 “Real-time clock and battery” on page 3478

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *"TA51xx Parameters"* tab to edit the parameterization of the desired option board.



Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 "System data AC500-eCo V3" on page 3352*

Only additional details are therefore documented below.

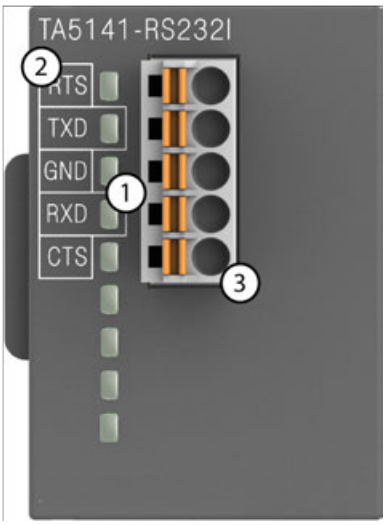
Parameter	Value
Buffering time	7 days at room temperature
Usable CPUs	PM5012
Internal power supply	Via internal CPU connection
Additional current consumption from 24 V DC power supply at CPU	Max. 25 mA
Weight	16 g

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 200 R0002	TA5131-RTC:AC500, real-time clock without battery, option board for AC500-eCo V3 Basic CPU	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA5141-RS232I - Option board for COMx serial communication



1 2 LEDs for communication state display (TxD and RxD)
2 Allocation of signal name
3 5-pin terminal block for communication interface

Intended purpose

Option board for COMx serial communication TA5141-RS232I is equipped with 1 RS-232 serial interface with handshake.

Connections

Serial interfaces

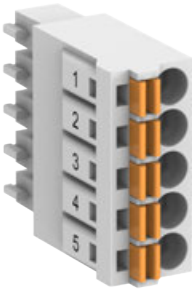
NOTICE!

Damage to the serial communication interface by using 5-pin terminal block of the TA5101-4DI!

If the 5-pin terminal block of the TA5101-4DI option board is plugged into a option board for COMx serial communication TA5141-RS232I, TA5142-RS485I or TA5142-RS485, the communication interface will be damaged by the 24 V.

Please do not confuse the 5-pin terminal block of the TA5101-4DI with the 5-pin terminal block for serial communication interface of TA5141-RS232I, TA5142-RS485I or TA5142-RS485.

Table 439: TA5141-RS232I

Serial interface	Pin	Signal	Description
	1	RTS	Request To Send DCE is ready to accept data from the DTE
	2	TxD	Transmit data (output)
	3	GND	Common Ground
	4	RxD	Receive data (input)
	5	CTS	Clear To Send (input) DCE is ready to accept data from the DTE

Cable length

The maximum possible cable length of a serial connection subnet within a segment depends on the transmission rate.

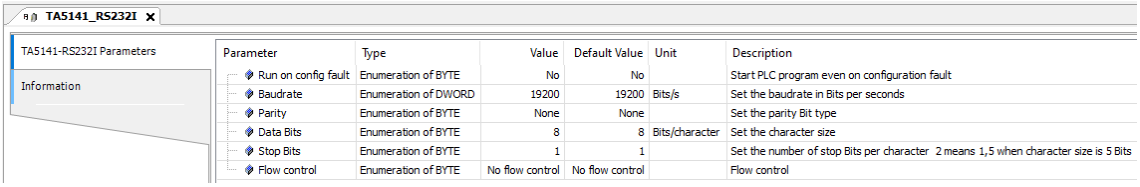
RS-232 for point-to-point connection:

Parameter	Value
Transmission rate	9.6 kBit/s to 115.2 kBit/s
Maximum cable length	On request

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *"TA51xx Parameters"* tab to edit the parameterization of the desired option board.



State LEDs

Signal	Color	State	Description
TxD	Yellow	ON (blinking)	Transmitting
RxD	Yellow	ON (blinking)	Receiving

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 "System data AC500-eCo V3" on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Protocol	Programmable with Automation Builder e.g. Modbus RTU / CAA SerialCom via serial interfaces
Interface	Serial interface
Serial interface standard	EIA RS-232
Potential separation	Yes, from the CPU, 500 V DC
Serial interface parameters	Configurable via software
Modes of operation	Data exchange
Transmission rate	9.6 kbit/s to 115.2 kbit/s
Protocol	Programmable
Interface connector	5-pin terminal block, male
Usable CPUs	PM50x2

Parameter	Value
Internal power supply	Via internal CPU connection
Additional current consumption from 24 V DC power supply at CPU	Max. 25 mA
Weight	Ca. 15 g

Ordering data

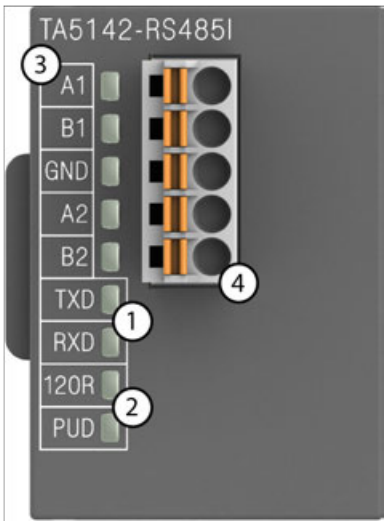
Part no.	Description	Product life cycle phase *)
1SAP 187 300 R0001	TA5141-RS232I: AC500, RS-232 option board for COMx serial communication, spring/cable front terminal, 3.50 mm pitch	Active
Spare parts		
1SAP 187 400 R0012 **)	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 3.5 mm pitch, 6 pieces per packing unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

***) The needed spring terminal block is always delivered with the option board.*

The terminal block listed in the table is for spare part only if needed.

TA5142-RS485I - Option board for COMx serial communication



1 2 LEDs for communication state display (TxD and RxD)
2 2 LEDs for termination state display
3 Allocation of signal name
4 5-pin terminal block for communication interface

Intended purpose

Option board for COMx serial communication TA5142-RS485(I) is equipped with 1 RS-485 (2-wire half-duplex) serial interface which can be used for communication via Modbus RTU or CAA SerialCom.

Bus terminations are built-in and configurable.

Connections

Serial interfaces

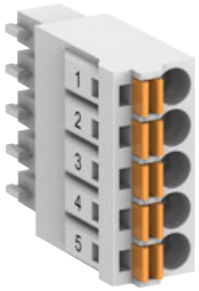
NOTICE!

Damage to the serial communication interface by using 5-pin terminal block of the TA5101-4DI!

If the 5-pin terminal block of the TA5101-4DI option board is plugged into a option board for COMx serial communication TA5141-RS232I, TA5142-RS485I or TA5142-RS485, the communication interface will be damaged by the 24 V.

Please do not confuse the 5-pin terminal block of the TA5101-4DI with the 5-pin terminal block for serial communication interface of TA5141-RS232I, TA5142-RS485I or TA5142-RS485.

Table 440: TA5142-RS485(I)

Serial interface	Pin	Signal
	1	A1 internally connected to A2
	2	B1 internally connected to B2
	3	GND
	4	A2 internally connected to A1
	5	B2 internally connected to B1

Protocols

No.	Protocol	Description
1	Modbus	Modbus RTU, master or slave
2	CAA SerialCom	Support for blocks contained in the CAA_SerialCom.lib library

Bus cable

Bus line	
Construction	2 cores, twisted, with common shield
Conductor cross section	> 0.22 mm² (24 AWG)
Twisting rate	> 10 per meter (symmetrically twisted)
Core insulation	Polyethylene (PE)
Resistance per core	< 100 Ω/km
Characteristic impedance	ca. 120 Ω (100 Ω...150 Ω)
Capacitance between the cores	< 55 nF/km (if higher, the max. bus length must be reduced)
Terminating resistors	120 Ω ¼ W at both line ends

Bus line	
Remarks	Commonly used telephone cables with PE insulation and a core diameter of > 0.8 mm are usually sufficient.
	Cables with PVC core insulation and core diameter of 0.8 mm can be used up to a length of approx. 250 m. In this case, the bus terminating resistor is approx. 100 Ω.

Cable length

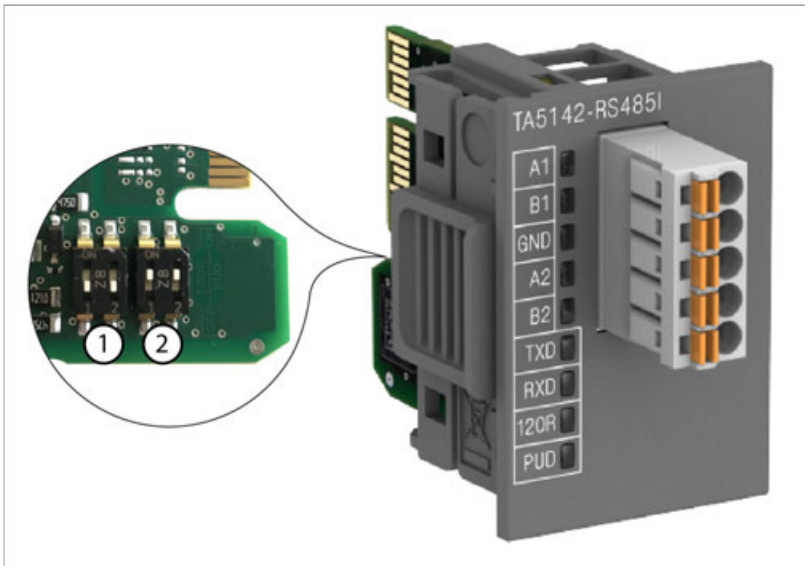
The maximum possible cable length of a serial connection subnet within a segment depends on the transmission rate.

RS-485 for point-to-point or bus connection:

Parameter	Value
Transmission rate	9.6 kbit/s to 115.2 kbit/s
Maximum cable length	On request

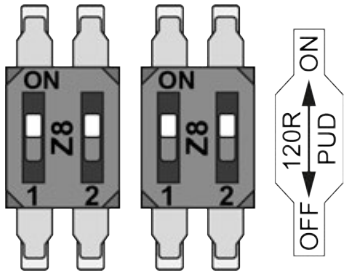
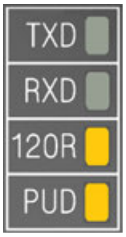
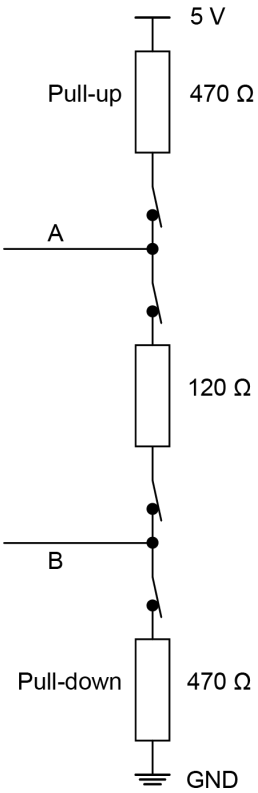
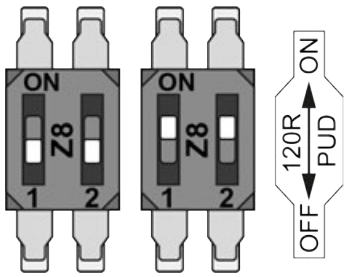
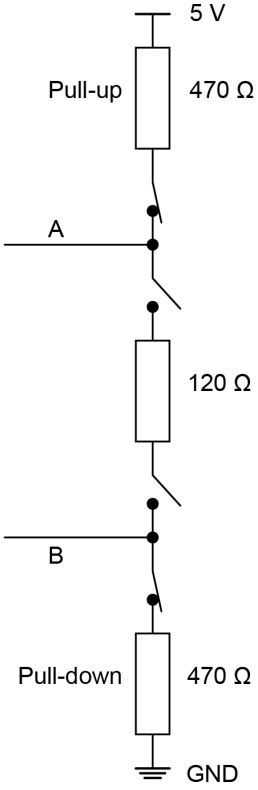
Bus termination

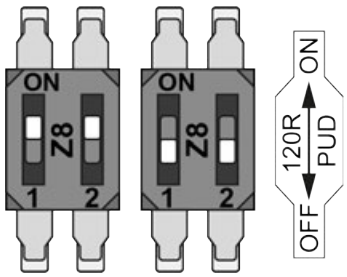
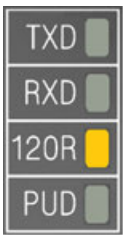
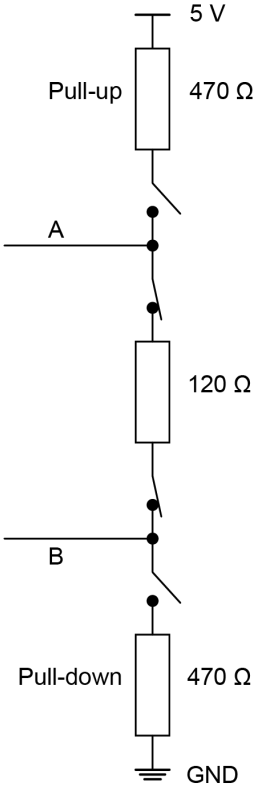
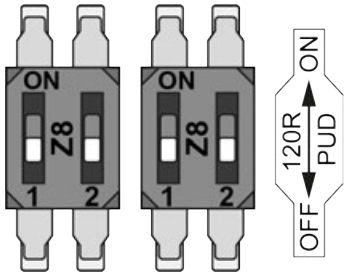
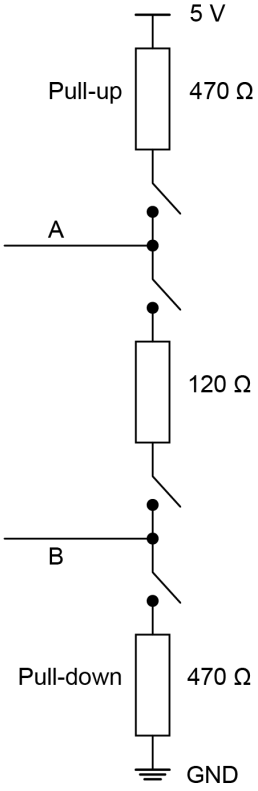
The line ends of the bus segment must be equipped with bus terminating resistors. These resistors are integrated in the module TA5142-RS485I. The pull-up and pull-down settings must also be made on the circuit board of the module.



1 Termination resistance settings
2 Pull-up and pull-down settings

Table 441: Configuration

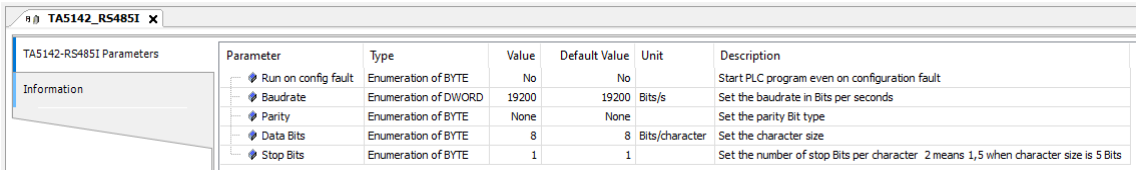
Settings on the module	State of LEDs	Internal wiring diagram	Description
			Master at the bus line end, pull-up and pull-down activated, bus termination 120 Ω
			Master within the bus line, pull-up and pull-down activated

Settings on the module	State of LEDs	Internal wiring diagram	Description
			Slave at the bus line end, bus termination 120 Ω
			Slave within the bus line

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *"TA51xx Parameters"* tab to edit the parameterization of the desired option board.



State LEDs

	Signal	Color	State	Description
	TxD	Yellow	ON (blinking)	Transmitting
	RxD	Yellow	ON (blinking)	Receiving
	120R	Yellow	ON	Bus termination
	PUD	Yellow	ON	Pull-up / Pull-down

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 "System data AC500-eCo V3" on page 3352*

Only additional details are therefore documented below.

Table 442: TA5142-RS485I

Parameter	Value
Protocol	Programmable with Automation Builder e.g. Modbus RTU / CAA_SerialCom via serial interfaces
Interface	Serial interface
Serial interface standard	EIA RS-485
Potential separation	Yes, from the CPU, 500 V DC
Serial interface parameters	Configurable via software
Modes of operation	Data exchange
Transmission rate	9.6 kbit/s to 115.2 kbit/s
Protocol	Programmable
Interface connector	5-pin terminal block, male
Usable CPUs	PM50x2
Internal power supply	Via internal CPU connection
Additional current consumption from 24 V DC power supply at CPU	Max. 25 mA
Weight	Ca. 16 g

Ordering data

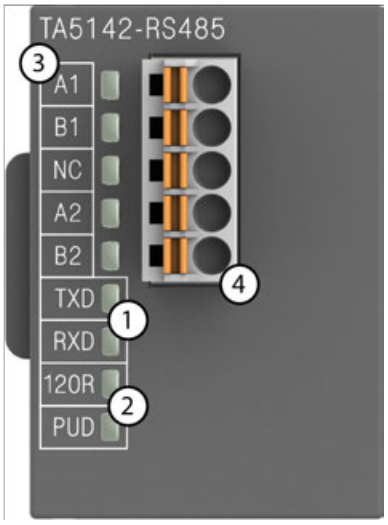
Part no.	Description	Product life cycle phase *)
1SAP 187 300 R0002	TA5142-RS485I: AC500, RS-485 serial adapter isolated option board, spring/cable front terminal, 3.50 mm pitch	Active
Spare parts		
1SAP 187 400 R0012 **)	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 3.5 mm pitch, 6 pieces per packing unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

***) The needed spring terminal block is always delivered with the option board.*

The terminal block listed in the table is for spare part only if needed.

TA5142-RS485 - Option board for COMx serial communication



1 2 LEDs for communication state display (TxD and RxD)
2 2 LEDs for termination state display
3 Allocation of signal name
4 5-pin terminal block for communication interface

Intended purpose

Option board for COMx serial communication TA5142-RS485(I) is equipped with 1 RS-485 (2-wire half-duplex) serial interface which can be used for communication via Modbus RTU or CAA SerialCom.

Bus terminations are built-in and configurable.

Connections

Serial interfaces

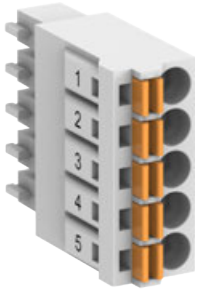
NOTICE!

Damage to the serial communication interface by using 5-pin terminal block of the TA5101-4DI!

If the 5-pin terminal block of the TA5101-4DI option board is plugged into a option board for COMx serial communication TA5141-RS232I, TA5142-RS485I or TA5142-RS485, the communication interface will be damaged by the 24 V.

Please do not confuse the 5-pin terminal block of the TA5101-4DI with the 5-pin terminal block for serial communication interface of TA5141-RS232I, TA5142-RS485I or TA5142-RS485.

Table 443: TA5142-RS485(I)

Serial interface	Pin	Signal
	1	A1 internally connected to A2
	2	B1 internally connected to B2
	3	GND
	4	A2 internally connected to A1
	5	B2 internally connected to B1

Protocols

No.	Protocol	Description
1	Modbus	Modbus RTU, master or slave
2	CAA SerialCom	Support for blocks contained in the CAA_SerialCom.lib library

Bus cable

Bus line	
Construction	2 cores, twisted, with common shield
Conductor cross section	> 0.22 mm² (24 AWG)
Twisting rate	> 10 per meter (symmetrically twisted)
Core insulation	Polyethylene (PE)
Resistance per core	< 100 Ω/km
Characteristic impedance	ca. 120 Ω (100 Ω...150 Ω)
Capacitance between the cores	< 55 nF/km (if higher, the max. bus length must be reduced)
Terminating resistors	120 Ω ¼ W at both line ends
Remarks	Commonly used telephone cables with PE insulation and a core diameter of > 0.8 mm are usually sufficient.
	Cables with PVC core insulation and core diameter of 0.8 mm can be used up to a length of approx. 250 m. In this case, the bus terminating resistor is approx. 100 Ω.

Cable length

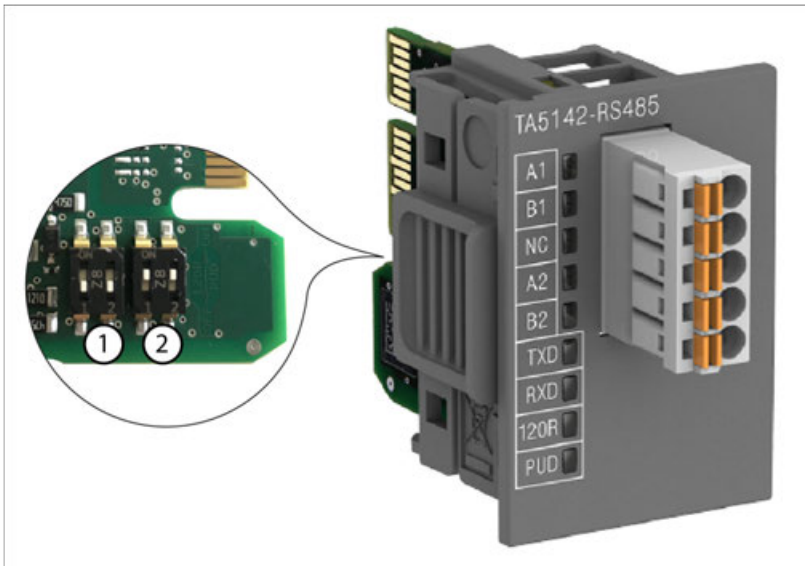
The maximum possible cable length of a serial connection subnet within a segment depends on the transmission rate.

RS-485 for point-to-point or bus connection:

Parameter	Value
Transmission rate	9.6 kbit/s to 115.2 kbit/s
Maximum cable length	On request

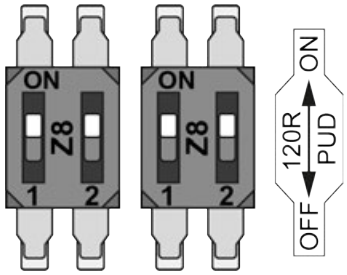
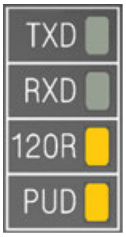
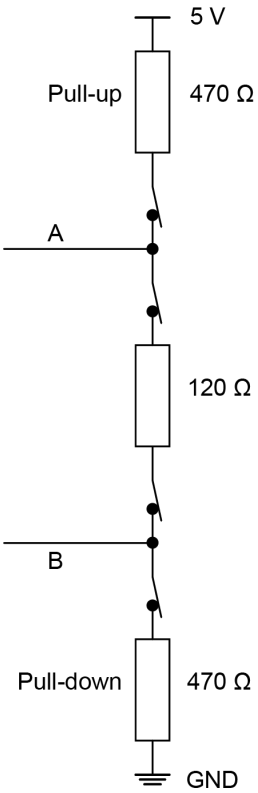
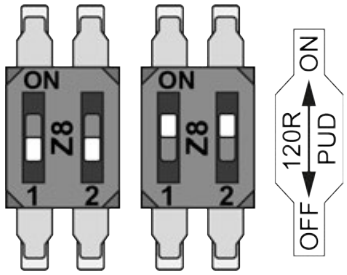
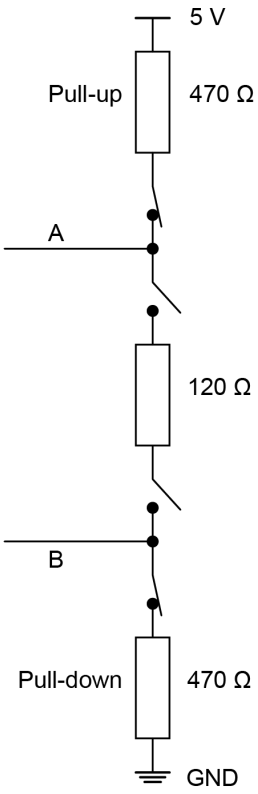
Bus termination

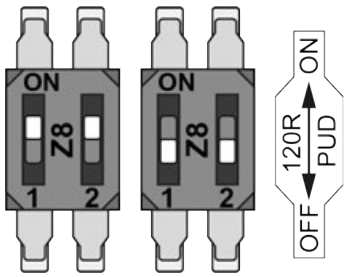
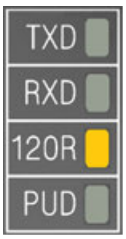
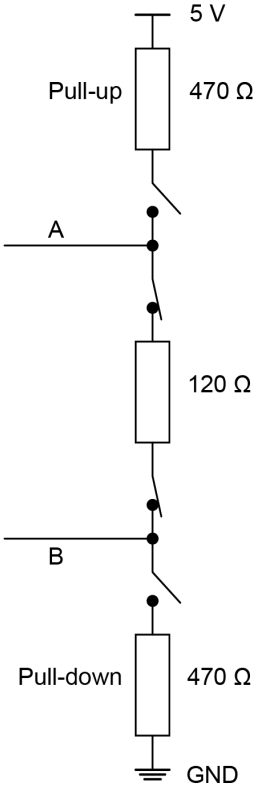
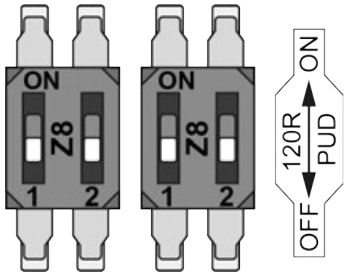
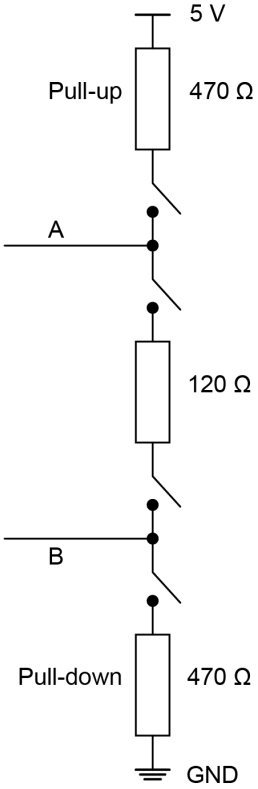
The line ends of the bus segment must be equipped with bus terminating resistors. These resistors are integrated in the module TA5142-RS485. The pull-up and pull-down settings must also be made on the circuit board of the module.



1 Termination resistance settings
2 Pull-up and pull-down settings

Table 444: Configuration

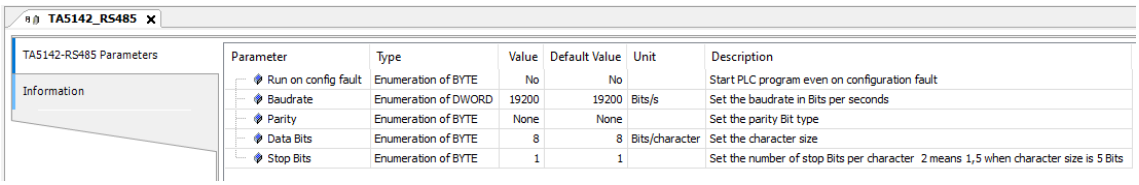
Settings on the module	State of LEDs	Internal wiring diagram	Description
			Master at the bus line end, pull-up and pull-down activated, bus termination 120 Ω
			Master within the bus line, pull-up and pull-down activated

Settings on the module	State of LEDs	Internal wiring diagram	Description
			Slave at the bus line end, bus termination 120 Ω
			Slave within the bus line

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

1. In the device tree, double-click the desired option board.
2. Select the *"TA51xx Parameters"* tab to edit the parameterization of the desired option board.



State LEDs

	Signal	Color	State	Description
	TxD	Yellow	ON (blinking)	Transmitting
	RxD	Yellow	ON (blinking)	Receiving
	120R	Yellow	ON	Bus termination
	PUD	Yellow	ON	Pull-up / Pull-down

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 "System data AC500-eCo V3" on page 3352*

Only additional details are therefore documented below.

Table 445: TA5142-RS485

Parameter	Value
Protocol	Programmable with Automation Builder e.g. Modbus RTU / CAA_SerialCom via serial interfaces
Interface	Serial interface
Serial interface standard	EIA RS-485
Potential separation	No
Serial interface parameters	Configurable via software
Modes of operation	Programming or data exchange
Transmission rate	9.6 kbit/s to 115.2 kbit/s
Protocol	Programmable
Interface connector	5-pin terminal block, male
Usable CPUs	PM50x2
Internal power supply	Via internal CPU connection
Additional current consumption from 24 V DC power supply at CPU	Max. 25 mA
Weight	Ca. 15 g

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 300 R0003	TA5142-RS485: AC500, RS-485 option board for COMx serial communication, spring/cable front terminal, 3.50 mm pitch	Active
Spare parts		
1SAP 187 400 R0012 **)	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 3.5 mm pitch, 6 pieces per packing unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

***) The needed spring terminal block is always delivered with the option board.*

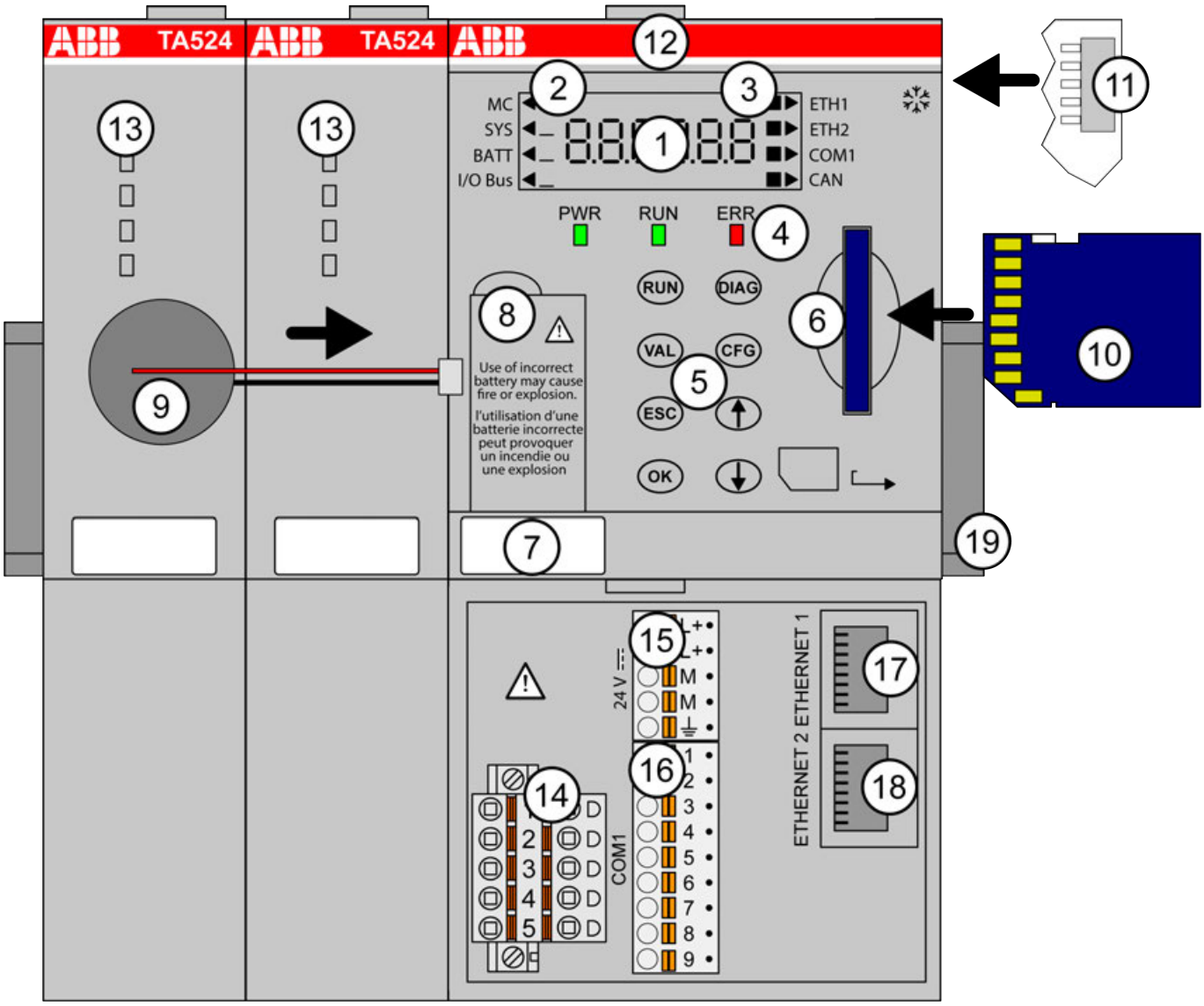
The terminal block listed in the table is for spare part only if needed.

1.6.3.3.2 AC500 (standard)

PM56xx-2ETH for AC500 V3 products

Processor modules with onboard interfaces:

- PM5630-2ETH: processor module, memory 8 MB, with Ethernet support (onboard Ethernet) – 2 network interfaces RJ45, CAN and COM1 on the terminal base.
- PM5650-2ETH: processor module, memory 80 MB, with Ethernet support (onboard Ethernet) – 2 network interfaces RJ45, CAN and COM1 on the terminal base.
- PM5670-2ETH: processor module, memory 160 MB, with Ethernet support (onboard Ethernet) – 2 network interfaces RJ45, CAN and COM1 on the terminal base.
- PM5675-2ETH: processor module, 160 MB, 8 GB flash disk, with Ethernet support (onboard Ethernet) – 2 network interfaces RJ45, CAN and COM1 on the terminal base.
- XC version for use in extreme ambient conditions available



1 6 7-segment state displays with backlight
2 "Triangle" displays for "item"
3 "Square" displays for "state"
4 3 state LEDs
5 8 function keys
6 Slot for memory card
7 Label
8 Compartment for lithium battery TA521
9 Lithium battery TA521
10 Memory card
11 I/O bus for connection of I/O modules
12 Slot for processor module (processor module mounted on terminal base)
13 Slots for communication modules (multiple, depending on terminal base; unused slots must be covered with TA524)
14 Interface for CAN (5-pin terminal block, removable)
15 Power supply (5-pin terminal block, removable)
16 Serial interface COM1 (9-pin terminal block, removable)
17 RJ45 female connector for ETHERNET1 connection
18 RJ45 female connector for ETHERNET2 connection
19 DIN rail
❄ Sign for XC version

Short description

The processor modules are the central units of the control system AC500. The types differ in their performance (memory size, speed etc.). Each processor module must be mounted on a suitable terminal base.

The terminal base type (TB56xx) depends on the number of communication modules which are used together with the processor module.

Table 446: Comparison: TB56xx

Processor module		PM5630	PM5650	PM5670	PM5675
Max. number of variables allowed for each communication module supported					
	Input variables	4 kB	4 kB	5 kB	5 kB
	Output variables	4 kB	4 kB	5 kB	5 kB
Type of communication module supported					
	CM574-RS/RCOM - serial interface	No	No	No	No
	CM582-DP - PROFIBUS DP V0/V1 slave	No	No	No	No
	CM592-DP - PROFIBUS DP V0/V1 master	[1])	[1])	[1])	[1])
	CM579-ETHCAT - EtherCAT master	x	x	x	x
	CM579-PNIO - PROFINET IO RT controller	x	x	x	x
	CM589-PNIO - PROFINET IO RT device	[1])	[1])	[1])	[1])
	CM589-PNIO-4 - PROFINET IO RT with 4 devices	[1])	[1])	[1])	[1])
	CM597-ETH - Ethernet interface	No	No	No	No
	CM588-CN - CAN, CANopen slave	No	No	No	No
	CM598-CN - CAN, CANopen master	only CAN 2A/2B	only CAN 2A/2B	only CAN 2A/2B	only CAN 2A/2B
Type of AC500-S module supported					
	SM560-S - safety module	x	x	x	x
	SM560-S-FD-1 - safety module with F-Device functionality for 1 PROFIsafe network	[1])	[1])	[1])	[1])
	SM560-S -FD-4 - safety module with F-Device functionality for 1 PROFIsafe network	[1])	[1])	[1])	[1])
Remarks: [1]) in preparation					

All terminal bases (TB56xx) provide the same communication interfaces (ETH1, ETH2, CAN and COM1). *Chapter 1.6.3.2.1.3 “Technical data” on page 2437*

All other V3 processor modules can operate multiple communication modules via their communication module interface.

The communication modules are mounted on the left side of the processor module on the same terminal base.

On the right side of the processor module, up to 10 digital or analog I/O expansion modules can be connected to the I/O bus. Each I/O module requires a suitable terminal unit depending on the module type.

Terminal bases, terminal units, I/O modules, communication modules and accessories have their own technical descriptions.

Each processor module can be used as:

- Stand-alone processor module
- Stand-alone processor module with local I/Os
- Remote IO server
- Remote IO client

The processor modules are powered with 24 V DC.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

Connections

All terminals for connection are available on the terminal base. For information on connection and available interfaces see the descriptions for

- *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*.

Processor modules PM56xx-2ETH can only be used with TB56xx-2ETH terminal bases.

Table 447: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])
Remarks: The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base. [1]) PM567x must have an index ≥ C0.				

Storage elements

Lithium battery

The processor modules are supplied without lithium battery. It must be ordered separately. The TA521 lithium battery is used for data (SRAM) and RTC buffering while the processor module is not powered.

See system technology - AC500 battery. *Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479*

The CPU monitors the discharge degree of the battery. A warning is issued before the battery condition becomes critical (about 2 weeks before). Once the warning message appears, the battery should be replaced as soon as possible.

The technical data, handling instructions and the insertion/replacement of the battery is described in detail in the chapter TA521 lithium battery *Chapter 1.6.3.8.2.4 “TA521 - Battery” on page 3324*.

Memory card

AC500 processor modules are supplied without memory card. It must be ordered separately.

The memory card can be used

- to read and write user files
- to download a user program
- for firmware updates

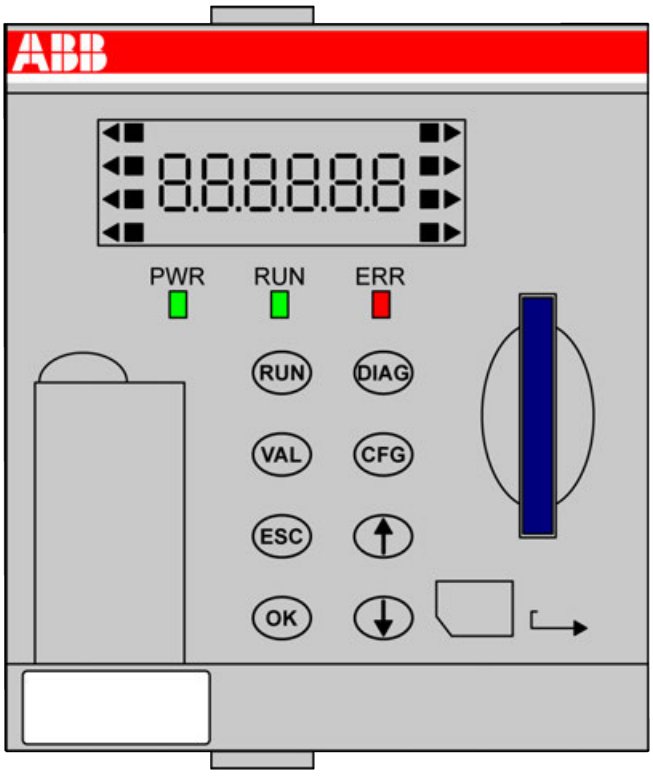
Detailed information can be found in the system technology chapter. *Chapter 1.6.5.1.2 “System processing” on page 3463*

AC500 processor modules can be operated with and without memory cards. The processor module uses a standard file system (FAT). This allows standard card readers to read and write the memory cards.

> *Only genuine MC502 memory cards are supported.*

For more information on the technical data, handling instructions and the insertion/replacement of the memory card, please refer to the chapter memory card MC502. *Chapter 1.6.3.8.2.1 “MC502 - Memory card” on page 3311*

LEDs, display and function keys on the front panel



Detailed information on using the LEDs, display and the function keys such as startup procedure and error coding is described in the system technology section Chapter 1.6.5.1.6 “LEDs, display and function keys on the front panel” on page 3486.

Technical data

The system data of AC500 and S500 are applicable to the standard version. *Chapter 1.6.4.6.1 “System data AC500” on page 3398*

The system data of AC500-XC are applicable to the XC version. *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Processor module and terminal base

Parameter	Value
Connection of the supply voltage 24 V DC at the terminal base of the processor module	Removable 5-pin terminal block with spring connection
Current consumption on 24 V DC	
Min. typ. (module alone)	PM5630-2ETH: 110 mA PM5650-2ETH: 120 mA PM5670-2ETH: 130 mA PM5675-2ETH: 140 mA
Max. typ. (all communication modules and I/Os)	PM5630-2ETH: 850 mA PM5650-2ETH: 900 mA PM5670-2ETH: 950 mA PM5675-2ETH: 950 mA
Number of slots for processor modules	1 (on all terminal bases)
Processor module interfaces at the terminal bases TB56xx	I/O bus, ETH1, ETH2, CAN, COM1
Connection system	See *Chapter 1.6.4.6.4 “Connection and wiring” on page 3416*
Weight (processor module without terminal base)	135 g
Mounting position	Horizontal or vertical

Table 448: Comparison: PM56xx

Processor module		PM5630	PM5650	PM5670	PM5675
Total maximum downloadable application size [1]		9 MB	84 MB	176 MB	176 MB
	Thereof user program code and data (dynamically allocated)	2 MB	8 MB	32 MB	32 MB
	Thereof user webserver data	7 MB	76 MB	144 MB	144 MB

Processor module		**PM5630**	**PM5650**	**PM5670**	**PM5675**
	Remaining for all other usage (project save, infra-structure...)	30 MB	285 MB	643 MB	643 MB
Buffered (SRAM)		256 kB	256 kB	1.5 MB	1.5 MB
	Thereof VAR retain persistent	128 kB	128 kB	1024 kB	1024 kB
	Thereof %M memory (e.g. Modbus register)	128 kB	128 kB	512 kB	512 kB
Expandable memory		None	None	None	None
Integrated mass storage memory (FLASH)		None	None	None	8 GB
Slot for pluggable memory card		MC502	MC502	MC502	MC502
Processor type		TI ARM Cortex-A9 32-bit-RISC			
Processor speed		300 MHz	600 MHz	1 GHz	1 GHz
Cycle time for 1 instruction (minimum):					
	Binary	Min. 0.02 µs	Min. 0.01 µs	Min. 0.002 µs	Min. 0.002 µs
	Word	Min. 0.02 µs	Min. 0.01 µs	Min. 0.002 µs	Min. 0.002 µs
	Floating point	Min. 0.12 µs	Min. 0.01 µs	Min. 0.002 µs	Min. 0.002 µs
Mathematic co-processor		x	x	x	x
Motion capability					
	No. synchronized axis per 1 ms on EtherCAT CM typically	-	8*	16*	16*
	No. synchronized axis per 2 ms on EtherCAT CM typically	4*	16*	>32	>32
	No. synchronized axis per 4 ms on EtherCAT CM or CANopen onboard typically	8*	>32	>32	>32
	Min. bus cycle time for EtherCAT using external CM579	2 ms	1 ms	0,5 ms	0,5 ms
* in addition: 1 virtual axis					
Max. number of central inputs and outputs (10 exp. modules):					
	Digital inputs	320			
	Digital outputs	320			
	Analog inputs	160			
	Analog outputs	160			
Number of decentralized inputs and outputs		Depends on the used fieldbus			
Data backup		Battery			
Data buffering time at 25 °C		Typ. 3 years			
Battery low indication		via application program			

Processor module		**PM5630**	**PM5650**	**PM5670**	**PM5675**
Real-time clock:					
	With battery backup	x			
	Accuracy	Typ. ±2 s / day at 25 °C			
Program execution:					
	Cyclic	x			
	Time-controlled	x			
	Multitasking	x			
	Minimum cycle time configurable for cyclical task	1 ms	1 ms	0,5 ms	0,5 ms
User program protection by password		x (user management)			
Internal interfaces for communication:					
Serial interface COM1:					
	Physical link	Configurable for RS-232 or RS-485 (9.6 kb/s, 19.2 kb/s, 38.4 kb/s, 57.6 kb/s and 115.2 kb/s)			
	Connection	Pluggable terminal block, spring connection			
	Usage	Serial ASCII communication,Modbus RTU			
CAN interface:					
	Physical link	CAN 2A/2B (from 50 kb/s to 1 Mb/s)			
	Connection	Pluggable terminal block, spring connection			
	Usage	CANopen master communication, CAN 2A/2B, J1939 protocol, CAN sync			
	Max. number of variables allowed				
	Input variables	2 kB	4 kB	5 kB	5 kB
	Output variables	2 kB	4 kB	5 kB	5 kB
Network interface ETH1, ETH2:					
	Usage	Ethernet			
	Physical link	10/100 base-TX, configurable as internal switch or independent Interfaces			
	Connection	2x RJ45 socket, provided on TB56xx-2ETH			
LEDs, LCD display, function keys		RUN / STOP, status, diagnosis, settings			
Number of timers		Unlimited			
Number of counters		Unlimited			
Programming languages:					
	Structured Text ST	x			
	Instruction list IL	x			
	Function Block Diagram FBD	x			
	Ladder Diagram LD	x			

Processor module		PM5630	PM5650	PM5670	PM5675
	Sequential function chart SFC	x			
	Continuous function chart (CFC)	x			

Remarks:

[1]: The values are for information only and cannot be fulfilled altogether. The available resources are limited at the end by the maximal downloadable application size for each CPU.

Table 449: Combination of TB56xx-2ETH(-XC) and PM56xx(-XC)

Processor module	PM5630	PM5650	PM5670	PM5675
TB5600-2ETH	0 slot	0 slot	0 slot	0 slot
TB5610-2ETH	1 slot	1 slot	1 slot	1 slot
TB5620-2ETH	2 slots	2 slots	2 slots	2 slots
TB5640-2ETH	-	4 slots	4 slots	4 slots
TB5660-2ETH	-	-	6 slots [1])	6 slots [1])

Remarks:

The slots can be used for connecting communication modules or AC500-S modules. Note that only one AC500-S module can be connected at one terminal base.

[1]) PM567x must have an index ≥ C0.

Table 450: Comparison: TB56xx

Processor module		PM5630	PM5650	PM5670	PM5675
Max. number of variables allowed for each communication module supported					
	Input variables	4 kB	4 kB	5 kB	5 kB
	Output variables	4 kB	4 kB	5 kB	5 kB
Type of communication module supported					
	CM574-RS/RCOM - serial interface	No	No	No	No
	CM582-DP - PROFIBUS DP V0/V1 slave	No	No	No	No
	CM592-DP - PROFIBUS DP V0/V1 master	[1])	[1])	[1])	[1])
	CM579-ETHCAT - EtherCAT master	x	x	x	x
	CM579-PNIO - PROFINET IO RT controller	x	x	x	x
	CM589-PNIO - PROFINET IO RT device	[1])	[1])	[1])	[1])
	CM589-PNIO-4 - PROFINET IO RT with 4 devices	[1])	[1])	[1])	[1])
	CM597-ETH - Ethernet interface	No	No	No	No
	CM588-CN - CAN, CANopen slave	No	No	No	No
	CM598-CN - CAN, CANopen master	only CAN 2A/2B	only CAN 2A/2B	only CAN 2A/2B	only CAN 2A/2B
Type of AC500-S module supported					
	SM560-S - safety module	x	x	x	x
	SM560-S-FD-1 - safety module with F-Device functionality for 1 PROFIsafe network	[1])	[1])	[1])	[1])

Processor module		PM5630	PM5650	PM5670	PM5675
	SM560-S -FD-4 - safety module with F-Device functionality for 1 PROFIsafe network	[1])	[1])	[1])	[1])
Remarks: [1]) in preparation					

Communication and onboard protocols

Table 451: OPC UA server / OPC DA server

Processor module		PM5630	PM5650	PM5670	PM5675
OPC UA server		x	x	x	x
	Number of free tags + additional license for extension [1])	1.000	5.000	30.000	30.000
	Number of connections	10	20	50	50
	Min. sampling rate (limit)	500 ms	100 ms	50 ms	50 ms
OPC DA server AE		x	x	x	x
	Number of connections	8	8	8	8
Remarks: [1]) in preparation					

Table 452: Modbus, Telecontrol

Processor module		PM5630	PM5650	PM5670	PM5675
Modbus TCP client / server		x	x	x	x
	Number of Modbus clients ModMast in parallel on a CPU master (server)	30	50	120	120
	Number of Modbus server in parallel (e.g. for SCADA access)	15	25	50	50
IEC 60870-5-104 telecontrol protocol		x	x	x	x
	Number of free tags + additional license for extension [1])	1.000	5.000	10.000	10.000
	Control station (number of connections)	5	10	20	20
	Sub-station (number of connections)	5	10	20	20
Remarks: [1]) in preparation					

Ordering data

Processor modules for AC500 (Standard) V3 products

Part no.	Description	Product life cycle phase *)
1SAP 131 000 R0278	PM5630-2ETH, processor module, memory 8 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols	Active
1SAP 331 000 R0278	PM5630-2ETH-XC, processor module, memory 8 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version	Active
1SAP 141 000 R0278	PM5650-2ETH, processor module, memory 80 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols	Active
1SAP 341 000 R0278	PM5650-2ETH-XC, processor module, memory 80 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version	Active
1SAP 151 000 R0278	PM5670-2ETH, processor module, memory 160 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols	Active
1SAP 351 000 R0278	PM5670-2ETH-XC, processor module, memory 160 MB, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version	Active

Part no.	Description	Product life cycle phase *)
1SAP 151 500 R0278	PM5675-2ETH, processor module, memory 160 MB, 8 GB flash disk, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols	Active
1SAP 351 500 R0278	PM5675-2ETH-XC, processor module, memory 160 MB, 8 GB flash disk, 24 V DC, memory card slot, interface 1 RS-232/485, display, 2 RJ45 independent onboard Ethernet TCP/IP interfaces with Modbus TCP, web server, IEC60870-5-104 or selectable Ethernet based protocols, XC version	Active

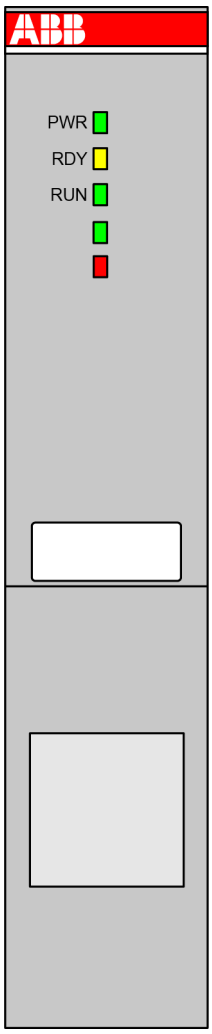
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 453: Accessories

Part no.	Description
1SAP 180 300 R0001	TA521, lithium battery
1SAP 180 100 R0001	MC502, memory card

1.6.3.4 Communication modules (AC500 standard)

1.6.3.4.1 Overview



AC500 communication modules are required for

- a connection to standard field bus systems and
- for integration into existing networks.

AC500 communication modules

- enable communication on different field buses.
- are mounted on the left side of the processor module on the same terminal base.
- are directly powered via the internal communication module bus of the terminal base. A separate voltage source is not required.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

For information on mounting and demounting, please refer to the chapter mounting and demounting the communication modules ↳ Chapter 1.6.4.6.3.5 “Mounting/Demounting the communication modules” on page 3414.

The communication between the processor module and the communication modules takes place via the communication module bus, which is integrated in the terminal base. Depending on the used terminal base up to 6 communication modules can be connected.

- ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*

There are no restrictions concerning which communication modules can be arranged for a processor module.

Within the AC500 control system, the communication modules can be used as

- bus master or
- slave.

It depends on the

- selected protocol,
- the functionality of the communication module and
- the several field buses and networks.

The following name extensions of the device names describe the supported field bus/protocol:

- CMxyz-ETH: Ethernet
- CMxyz-DP: PROFIBUS
- CMxyz-PNIO: PROFINET
- CMxyz-ETHCAT: EtherCAT
- CMxyz-CN: CANopen
- CMxyz-RCOM: RCOM/RCOM+ protocol (and 2 serial interfaces)
- CMxyz-RS: 2 serial interfaces (COM1/COM2)

If a XC version of the device is available, for use in extreme ambient conditions (e.g. wider temperature and humidity range), this is indicated with a snowflake sign.

Compatibility of communication modules and communication interface modules

Table 454: Modbus TCP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard Ethernet interface	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O

Table 455: PROFINET IO RT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	--	--	hot swap I/O

Table 456: CANopen

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard CAN interface	CI581-CN CI582-CN	--	--	--	remote I/O

Table 457: EtherCAT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-ETHCAT master	CI511-ETHCAT CI512-ETHCAT	x	x	--	remote I/O

Technical data (Overview)

Communication module	Field bus	Transmission rate	Field bus connector	Processor	Communication module interface	Current consumption from 24 V DC power supply at the terminal base of the CPU	Internal RAM memory	External RAM memory	External flash memory
CM579-ETHCAT	EtherCAT	10 or 100 MBit/s	2 x RJ45	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 85 mA	128 kB	8 MB	4 or 8 MB
CM598-CN	CANopen	10 ... 1 MBit/s	COMBICON 2x 5-pin, bended	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 65 mA	128 kB	8 MB	8 MB
CM579-PNIO	PROFINET	100 MBit/s	2 x RJ45	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 85 mA	128 kB	8 MB	4 or 8 MB

1.6.3.4.2 Compatibility of communication modules and communication interface modules

Table 458: Modbus TCP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard Ethernet interface	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O

Table 459: PROFINET IO RT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	--	--	hot swap I/O

Table 460: CANopen

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard CAN interface	CI581-CN CI582-CN	--	--	--	remote I/O

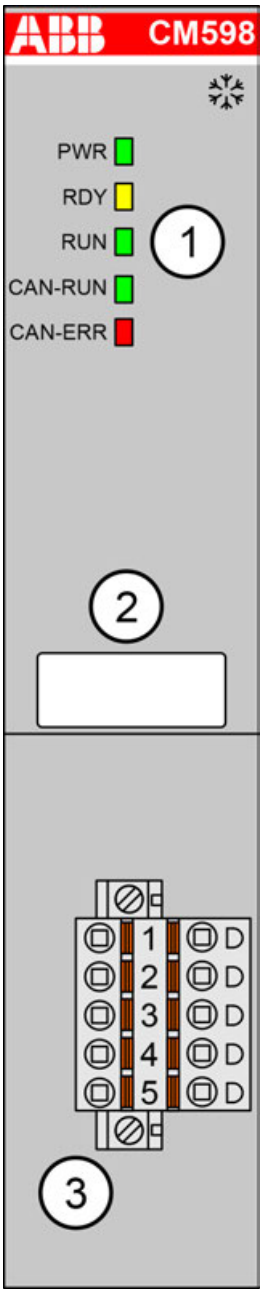
Table 461: EtherCAT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-ETHCAT master	CI511-ETHCAT CI512-ETHCAT	x	x	--	remote I/O

1.6.3.4.3 CANopen

CM598-CN - CANopen master

- CANopen master 1 Mbit/s
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 Label
3 Communication interface, 5-pin, Combicon, male, removable plug with spring terminals
❄ Sign for XC version

Purpose

Communication module CM598-CN enables communication over the CANopen field bus.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

The AC500 V3 CPUs only support CAN 2A/2B protocol on the communication module CM598-CAN ↬ Chapter 1.6.6.2.11.1.1.2 “Configuration of the protocols CAN 2.0 A / CAN 2.0 B” on page 3740.

Support of CANopen protocol is in preparation.

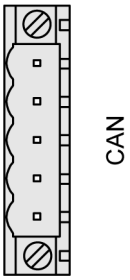
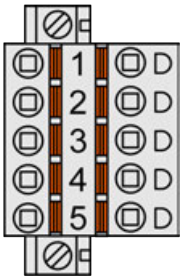
Connections

Field bus interface

Interface socket	5-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	CANopen (CAN), 1 Mbaud max.
Transfer rate (transmission rate)	10 kbit/s, 20 kbit/s, 50 kbit/s, 100 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s,

The CANopen connector has the following pin assignment:

Pin assignment

Interface		PIN	Signal	Description
 Terminal block removed	 Terminal block inserted	1	CAN_GND	CAN reference potential
		2	CAN_L	Bus line, receive/transmit line, LOW
		3	CAN_SHLD	Shield of the bus line
		4	CAN_H	Bus line, receive/transmit line, HIGH
		5	NC	Not connected

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
50 kbit/s	1000 m

Types of bus cables

For CANopen, only bus cables with characteristics as recommended in ISO 11898 are to be used. The requirements for the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	Bus cable (shielded, twisted pair)			Max. transmission rate [kbit/s]
	Conductor cross section [mm²]	Line resistance [Ω/km]	Wave impedance [Ω]	
0...40	0.25...0.34 / AWG23, AWG22	70	120	1000 at 40 m
40...300	0.34...0.60 / AWG22, AWG20	< 60	120	< 500 at 100 m
300...600	0.50...0.60 / AWG20	< 40	120	< 100 at 500 m
600...1000	0.75...0.80 / AWG18	< 26	120	< 50 at 1000 m

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

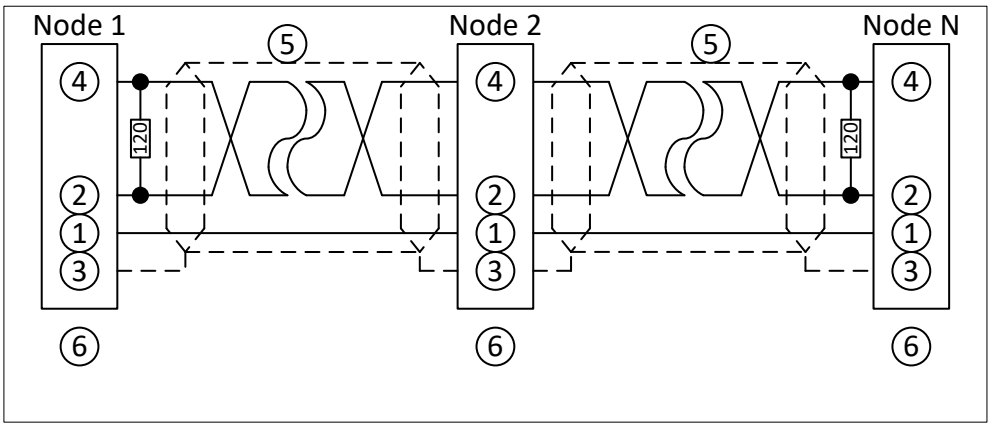


Fig. 102: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

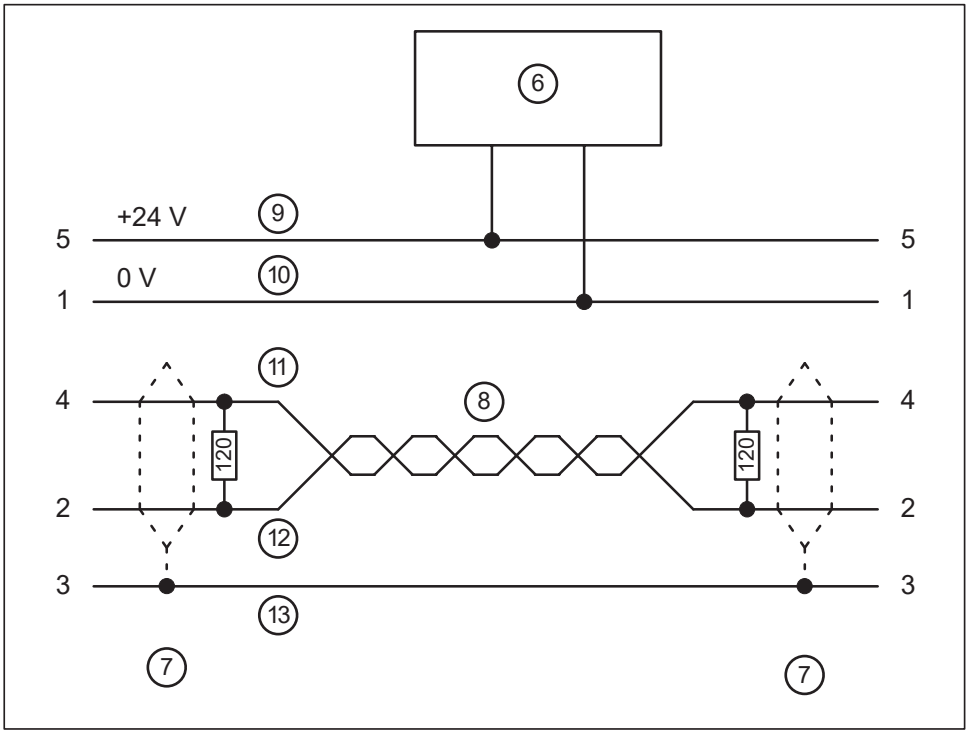


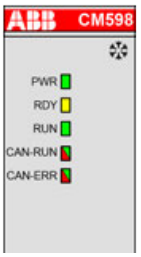
Fig. 103: DeviceNet interface, bus terminating resistors connected to the line ends

6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to Chapter 1.6.4.6.1 “System data AC500” on page 3398.

State LEDs

Table 462: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	ON (light)	Power supply available
			OFF (dark)	Power supply not available or defective hardware
	RDY	Yellow	ON	Boot procedure
			Blinking	Boot failure
			OFF	---
	RUN	Green	ON	Communication module is operational
			Blinking	---
			OFF	Communication module is not operational
	CAN-RUN	Green	ON	Operational: Device is in the OPERATIONAL state
			Single Flash	Stopped: Device is in STOPPED state
			Blinking	Pre-operational: Device is in the PREOPERATIONAL state
			OFF	No communication or no power supply
	CAN-ERR	Red	ON	CANopen bus is off
			Single flash	Warning limit reached: At least one of the error counters of the CAN controller has reached or exceeded the warning level (too many error frames)
			Double flash	Error control event: A guard event (NMT Slave or NMTmaster) or a heartbeat event (Heartbeat consumer) has occurred
			OFF	No Error: Device is in working condition
	CAN-RUN	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	CAN-ERR	Yellow		
LED state during firmware update	CAN-RUN	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	CAN-ERR	Red		
	CAN-RUN	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	CAN-ERR	Red		

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	CANopen master (in preparation), CAN2A, CAN2B
Transmission rate	10 kbit/s to 1 Mbit/s
Ambient temperature	see: System data AC500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* System Data AC500 XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*
Usable terminal bases	All TB5xx
Field bus connector	Pluggable connector COMBICON, 5-pin
Technology	Hilscher NETX 100
Indicators	5 LEDs
Internal power supply	Via the communication module interface of the terminal base
Current consumption from 24 V DC power supply at the Terminal Base of the CPU	Typ. 65 mA
Number of Slaves	Max. 126
Number of receive/transmit PDOs	Max. 512 (respectively for receive and transmit)
Total quantity of input and output data	Max. 3584 byte (respectively for input and output)
Weight	Ca. 150 g

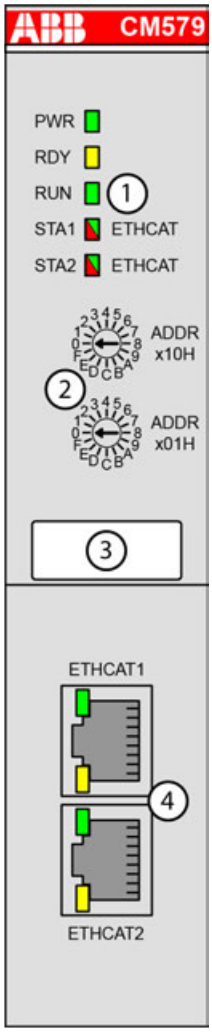
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 173 800 R0001	CM598-CN, communication module CANopen master	Active
1SAP 373 800 R0001	CM598-CN-XC, communication module CANopen master, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.4.4 EtherCAT

CM579-ETHCAT - EtherCAT master



1 5 LEDs for state display
2 2 rotary switches for address setting (not used)
3 Label
4 2 communication interfaces RJ45 (ETHCAT1 and ETHCAT2)

Intended purpose

Communication module CM579-ETHCAT is for EtherCAT communication.

The comunication module is configured via the dual-port memory by means of a system configurator. The configuration is saved on a non-volatile Flash EPROM memory.

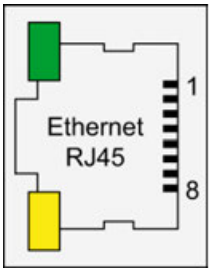
Chapter 1.7.3.4.1 “CM579-ETHCAT” on page 4074

Connections

Field bus interfaces

The EtherCAT communication module provides 2 RJ45 interfaces with the following pin assignment. The pin assignment is used for the EtherCAT slaves (communication interface modules CI5xy-ETHCAT) as well.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.

The EtherCAT network differentiates between input-connectors (IN) and output-connectors (OUT):

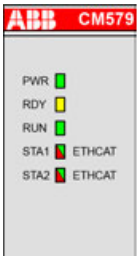
At the EtherCAT slaves (communication interface modules), the ETH1-connector is IN and the ETH2-connector is OUT.

At the EtherCAT master (communication module), the ETHCAT1 connector has to be used. The ETHCAT2 connector is reserved for future extensions.

State LEDs

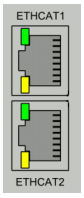
The EtherCAT state is shown by the EtherCAT communication module's LEDs. Some LEDs are two-colored.

Table 463: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	On	Power supply available
			Blinking	---
			Off	Power supply not available or defective hardware
	RDY	Yellow	On	Boot procedure
			Blinking	Boot failure
			Off	---
	RUN	Green	On	Communication module is operational
			Blinking	---
			Off	Communication module is not operational
	STA1	Green	On	No bus error, communication running
			Blinking	Establishing communication
			Off	System error
	STA2	Red	On	Configuration error
			Blinking	---
			Off	No error
	STA1	Yellow	Blinking	No production data available,
	STA2	Yellow	(synchronously)	no bus communication possible.
LED state during firmware update	STA1	Green	Blinking	Firmware file transfers during communication module firmware update.
	STA2	Red	(synchronously)	
	STA1	Green	Blinking	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	STA2	Red	(alternately)	

The RJ45 Ethernet connector contains two LEDs showing the current Ethernet port connection state.

Table 464: Meaning of the diagnosis LEDs

LED		Color	State	Description
	ETHCAT1 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	ETHCAT1 LED "RX/TX"	Yellow	On	Device sends/receives frames
			Off	No Ethernet connection
	ETHCAT2 LED "Link"	Green		Connector ETHCAT2 is not used
	ETHCAT2 LED "RX/TX"	Yellow		

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Internal Supply	Via the communication module interface of the terminal base
Protocol	EtherCAT
Field bus connector	2 x RJ45 (ETHCAT1 and ETHCAT2)
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Bus length (segment length max.)	100 m at 100 Mbit/s
Indicators	5 LEDs
Usable CPUs	PM56xx *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*
Usable terminal bases	All TB56xx (not TB5600) *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*
Ambient temperature	System data AC500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* System Data AC500 XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 85 mA
Internal supply	Via the communication module interface of the terminal base
Number of slaves	Limited to 200
Quantity of input and output data for a single slave	Max. 5760 bytes (respectively for input and output)
Total quantity of input and output data	Max. 5760 bytes (only valid for asynchronous operation, for synchronous operation the reachable values depends on the additional load of SoE, CoE and EoE, typical reachable values are 1024 bytes).
Supported protocols	RTC - Real-time cyclic protocol, class 1 RTA - Real-time acyclic protocol
Acyclic services	● CoE upload ● CoE download (1500 bytes max.) ● Emergency
Min. bus cycle	1 ms

Parameter	Value
Max. size of the bus configuration file	2 MB
Weight	Ca. 170 g

Ordering data

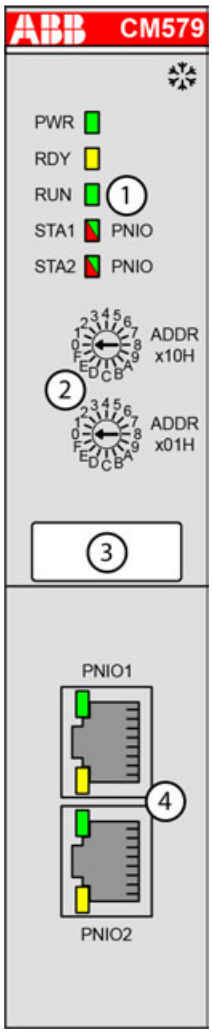
Part no.	Description	Product life cycle phase *)
1SAP 170 902 R0101	CM579-ETHCAT, EtherCAT communication module	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.4.5 PROFINET

CM579-PNIO - PROFINET IO RT controller

- PROFINET IO controller
- Integrated 2-port switch
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 2 rotary switches for address setting (not used)
3 Label
4 2 communication interfaces RJ45 (PNIO1 and PNIO2)
❄ Sign for XC version

Intended purpose

The communication module is for PROFINET RT communication.

The PROFINET communication module includes an internal Ethernet switch. The connection to the Ethernet can be established directly to the communication module. An additional switch is not necessary.

The communication module is configured via the dual-port memory by means of a system configurator. The configuration is saved on a non-volatile Flash EPROM memory.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

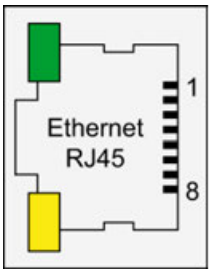
Parameter	Value
Protocol	PROFINET IO RT
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*
Usable terminal bases	All TB56xx (not TB5600) *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*
Field bus connector	2 RJ45 (PNIO1 and PNIO2), with integrated 2-port switch
Internal supply	Via the communication module interface of the terminal base

Connections

Field bus interfaces

The communication module provides 2 RJ45 interfaces.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

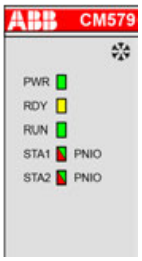
Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.

State LEDs

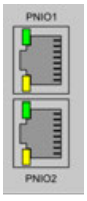
The PROFINET state is shown by the state LEDs.

Table 465: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	On	Power supply available
			Blinking	---
			Off	Power supply not available or defective hardware
	RDY	Yellow	On	Boot procedure
			Blinking	Boot failure
			Off	---
	RUN	Green	On	Communication module is operational
			Blinking	---
			Off	Communication module is not operational
	STA1	Red	On	Diagnosis alarm reported. At least one device is having a diagnosis alarm. In incorporation with STA2 PNIO: License fault.
			Blinking	System error
			Off	No system error
	STA2	Red	On	No connection; in incorporation with STA1 PNIO: license fault
			Blinking	Configuration fault: some configured I/O modules are not connected
			Off	No bus error, communication is running
	STA1	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	STA2	Yellow		
LED state during firmware update	STA1	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	STA2	Red		
	STA1	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	STA2	Red		

The RJ45 Ethernet connector contains two LEDs showing the current Ethernet port connection state.

Table 466: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PNIO1 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	PNIO1 LED "RX/TX"	Yellow	On	---
			Blinking	PROFINET device sends/receives frames
			Off	---
	PNIO2 LED "Link"	Green	On	Ethernet connection established

LED		Color	State	Description
			Off	No Ethernet connection
	PNIO2 LED "RX/TX"	Yellow	On	---
			Blinking	PROFINET device sends/receives frames
			Off	---

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	PROFINET IO RT
Bus connection	2 RJ45 (PNIO1 and PNIO2), with integrated 2-port switch
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Bus length (segment length max.)	100 m
Indicators	5 LEDs
Usable terminal bases	All TB5xx All TB56xx (not TB5600) *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*
Supported alarm types	Process alarm, diagnostic alarm, return of Sub-Module, plug alarm, pull alarm
Alarm processing	Requires handling in application program
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 85 mA
Internal supply	Via the communication module interface of the terminal base
Weight	Ca. 170 g

Parameter		Value
Supported protocols		RTC - real-time cyclic protocol, class 1 RTA - real-time acyclic protocol DCP - discovery and configuration protocol *) CL-RPC - connectionless remote procedure call Since revision FW 2.4.8.0 additionally LLDP - link layer discovery protocol SNMP - simply network management protocol (SNMP v1)
Acyclic services		PNIO read / write (max. 1392 bytes per telegram, max. 4096 bytes per service request)
Total quantity of input and output data		
	CM579-PNIO < FW 2.4.8.0	1024 bytes per I/O module 3072 bytes in total
	CM579-PNIO = FW 2.4.8.0	1024 bytes per I/O module 4096 bytes in total
	CM579-PNIO > FW 2.4.8.0	1440 bytes per I/O module PM5630, PM5650: 4096 bytes in total PM567x: 5120 bytes in total
Min. bus cycle		1 ms
Conformance class		CC A

*) CM579-PNIO does not allow setting "Station name" by using PROFINET service "DCP SET NameOfStation".

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 170 901 R0101	CM579-PNIO, PROFINET communication module	Active
1SAP 370 901 R0101	CM579-PNIO-XC, PROFINET communication module, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.5 Terminal units (AC500 standard)

Hot swap

System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor modules PM56xx-2ETH with firmware version as of V3.2.0.*

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

Conditions for hot swapping

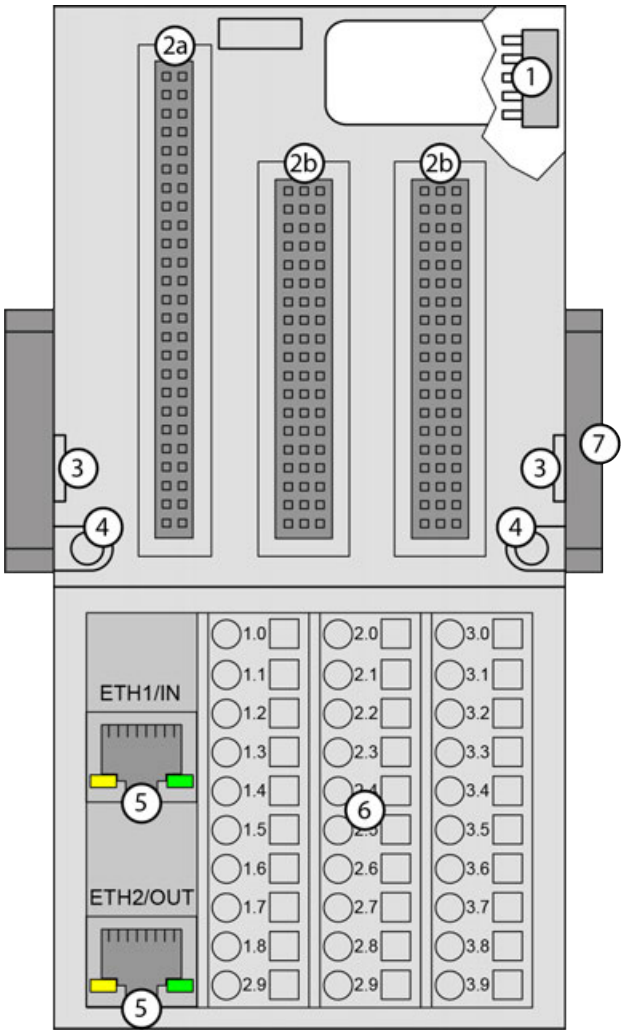
- *Digital outputs are not under load.*
- *Input/output voltages above safety extra low voltage/ protective extra low voltages (SELV/PELV) are switched off.*
- *Modules are completely plugged on the terminal unit with both snap fit engaged before switching on loads or input/output voltage.*

Hot swap

Further information about hot swap: ↬ Chapter 1.6.5.1.8 “Hot swap” on page 3523.

1.6.3.5.1 TU507-ETH and TU508-ETH for Ethernet communication interface modules

- TU507-ETH, Ethernet terminal unit, 24 V DC, screw terminals
- TU508-ETH, Ethernet terminal unit, 24 V DC, spring terminals
- TU508-ETH-XC, Ethernet terminal unit, 24 V DC, spring terminals, XC version



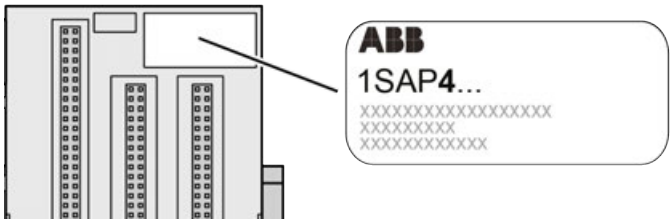
1 I/O bus (10 pins, female) to connect the first terminal unit
2a Plug (2x 25 pins) to connect the inserted Ethernet communication interface module
2b Plug (3x 19 pins) to connect the inserted Ethernet communication interface module
3 With a screwdriver, inserted in this place, the terminal unit and the adjacent terminal unit can be shoved from each other
4 2 holes for wall mounting
5 2 RJ45 interfaces with indication LEDs for connection with the Ethernet network
6 30 terminals for signals and process supply voltages (UP and UP3)
7 DIN rail

The Ethernet communication interface modules plug into the Ethernet terminal unit. When properly seated, they are secured with two mechanical locks. All the connections are made through the Ethernet terminal unit, which allows removal and replacement of the Ethernet communication interface modules without disturbing the wiring at the Ethernet terminal unit.

The Ethernet terminal units TU507-ETH and TU508-ETH are specifically designed for use with AC500/S500 Ethernet communication interface modules (e. g. CI501-PNIO).

XC version

XC = e**X**treme **C**onditions

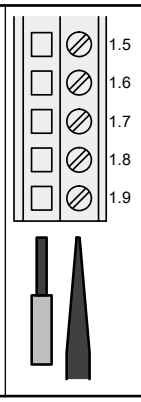
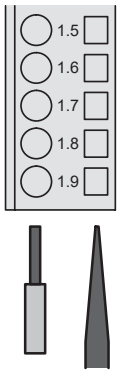


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			**Spring terminals**		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.4.6.4.3 “Terminals at the terminal unit” on page 3417.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.4.6.2.3 “Mechanical dimensions S500” on page 3406*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

The assignment of the other terminals is dependent on the inserted communication interface module.

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.3.8.3.4 “TA535 - Protective caps for XC devices” on page 3333*

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of I/O channels per module	Max. 24 (depending on the inserted communication interface module)
Distribution of the channels into groups	3 groups of max. 8 channels each (1.0...1.7, 2.0...2.7, 3.0...3.7), the allocation of the channels is given by the inserted Ethernet bus module
Network interface connector	2 RJ45, 8-pole
Rated voltage	24 V DC
Max. permitted total current	10 A via the supply terminals (UP, UP3 and ZP)
Ethernet	10/100 base-TX or 100 base-TX (depending on CI5xx module plugged in), 2 RJ45 socket
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring-type terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

Ordering data

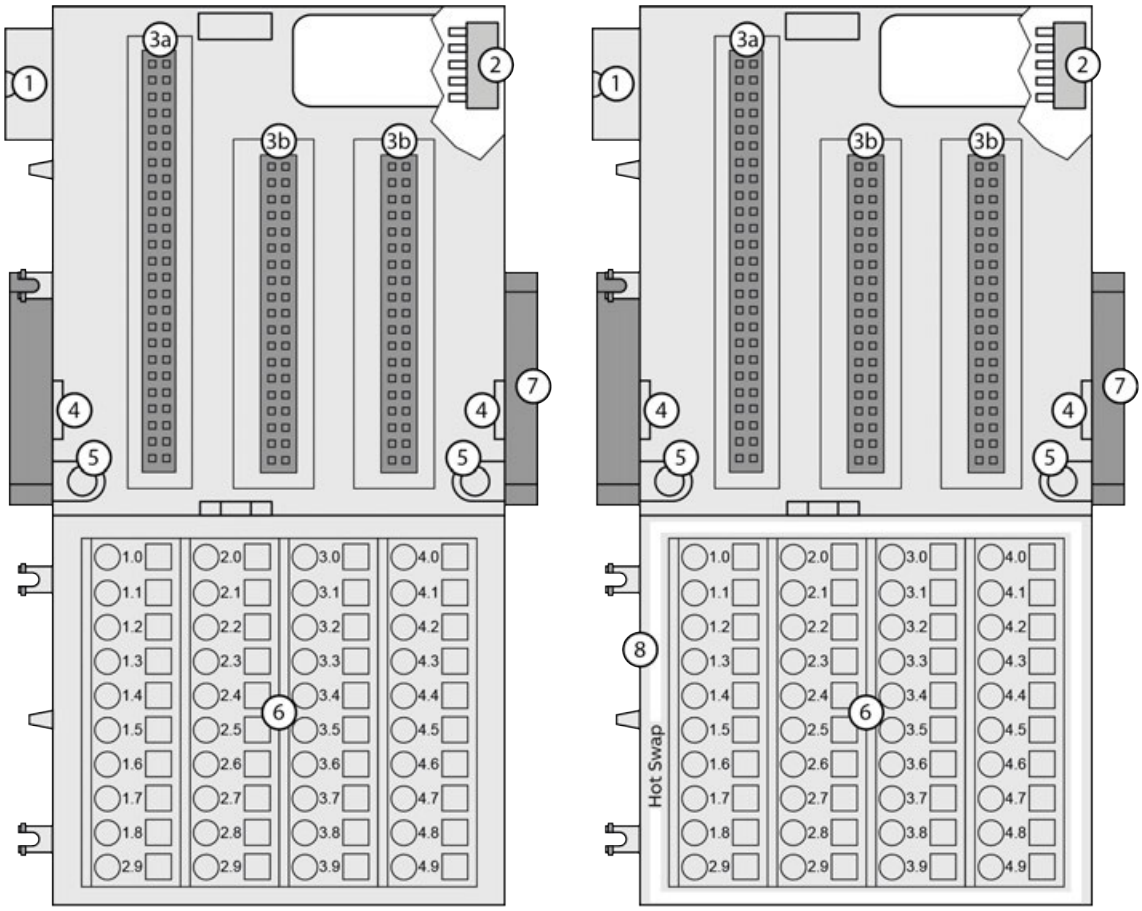
Part no.	Description	Product life cycle phase *)
1SAP 214 200 R0001	TU507-ETH, Ethernet terminal unit, 24 V DC, screw terminals	Active
1SAP 214 000 R0001	TU508-ETH, Ethernet terminal unit, 24 V DC, spring terminals	Active
1SAP 414 000 R0001	TU508-ETH-XC, Ethernet terminal unit, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.5.2 TU515, TU516, TU541 and TU542 for I/O modules

- TU515, I/O terminal unit, 24 V DC, screw terminals
- TU516, I/O terminal unit, 24 V DC, spring terminals
- TU516-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
- TU516-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
- TU516-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version
- TU541, I/O terminal unit, 24 V DC, screw terminals
- TU542, I/O terminal unit, 24 V DC, spring terminals
- TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
- TU542-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
- TU542-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version

The input/output modules plug into the I/O terminal unit. When properly seated, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the I/O modules without disturbing the wiring at the terminal unit.



1 I/O bus (10 pins, male) to connect the previous terminal unit, the CPU terminal base or the communication interface module to the terminal unit
2 I/O bus (10 pins, female) to connect other terminal units
3a Plug (2 x 25 pins) to connect the inserted I/O modules
3b Plug (2 x 19 pins) to connect the inserted I/O modules
4 With a screwdriver inserted in this place, the terminal unit and the adjacent terminal unit can be shoved from each other
5 Holes for screw mounting
6 40 terminals for signals and process supply voltage
7 DIN rail
8 White border signifies hot swap capability of the terminal unit

Hot swap

WARNING!

Risk of explosion or fire in hazardous environments during hot swapping!

Hot swap must not be performed in flammable environments to avoid life-threatening injury and property damage resulting from fire or explosion.

WARNING!

Electric shock due to negligent behavior during hot swapping!

To avoid electric shock

- make sure the following conditions apply:
  - Digital outputs are not under load.
  - Input/output voltages above safety extra low voltage/ protective extra low voltage (SELV/PELV) are switched off.
  - Modules are fully interlocked with the terminal unit with both snap-fits engaged before switching on loads or input/output voltage.
- Never touch exposed contacts (dangerous voltages).
- Stay away from electrical contacts to avoid arc discharge.
- Do not operate a mechanical installation improperly.

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

H = **H**ot swap

Hot swap

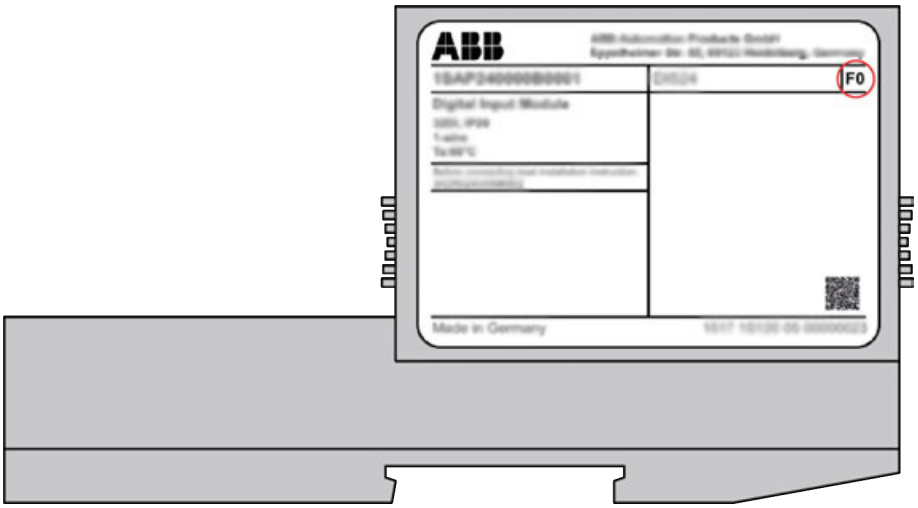
System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor modules PM56xx-2ETH with firmware version as of V3.2.0.*

Hot swap is not supported by AC500-eCo V3 CPU!



The index of the module is in the right corner of the label.

NOTICE!

Risk of damage to I/O modules!

Modules with index below F0 can be damaged when inserted or removed from the terminal unit in a powered system.

NOTICE!

Risk of damage to I/O modules!

Do not perform hot swapping if any I/O module with firmware version lower than 3.0.14 is part of the I/O configuration.

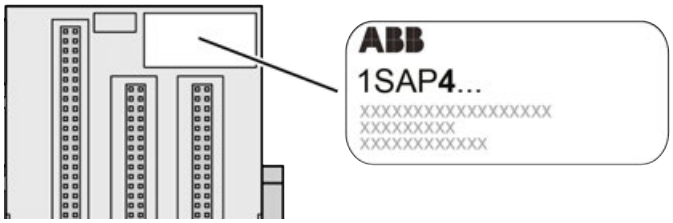
For min. required device index see table below.

Device	Min. required device index for I/O module as of FW Version 3.0.14
AC522(-XC)	F0
AI523 (-XC)	D2
AI531	D4
AI531-XC	D2
AI561	B2
AI562	B2
AI563	B3
AO523 (-XC)	D2
AO561	B2
AX521 (-XC)	D2
AX522 (-XC)	D2
AX561	B2
CD522 (-XC)	D1
DA501 (-XC)	D2
DA502 (-XC)	F0
DC522 (-XC)	D2
DC523 (-XC)	D2
DC532 (-XC)	D2
DC561	B2
DC562	A2
DI524 (-XC)	D2
DI561	B2
DI562	B2
DI571	B2

Device	Min. required device index for I/O module as of FW Version 3.0.14
DI572	A1
DO524 (-XC)	A3
DO526	A2
DO526-XC	A0
DO561	B2
DO562	A2
DO571	B3
DO572	B2
DO573	A1
DX522 (-XC)	D2
DX531	D2
DX561	B2
DX571	B3
FM562	A1

XC version

XC = e**X**treme **C**onditions

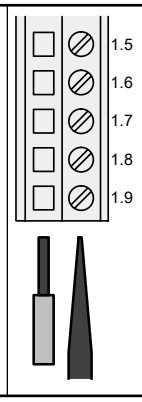
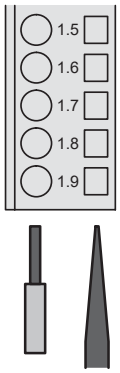


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (lable) identifies the XC version.*

Terminals

Screw terminals			Spring terminals		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↬ Chapter 1.6.4.6.4.3 “Terminals at the terminal unit” on page 3417.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↬ Chapter 1.6.4.6.2.3 “Mechanical dimensions S500” on page 3406*

The following terminals are used for connection of the process supply voltage.

	Terminals							
Type	1.8	2.8	3.8	4.8	1.9	2.9	3.9	4.9
TU515, TU516 and TU516-H	These terminals are internally connected with assignment: process supply voltage UP = +24 V DC				These terminals are internally connected with assignment: process supply voltage ZP = 0 V			
TU541, TU542 and TU542-H	These terminals are internally connected with assignment: process voltage UP = +24 V DC		Separate process supply voltage UP3 = +24 V DC	Separate process supply voltage UP4 = +24 V DC	These terminals are internally connected with assignment: process supply voltage ZP = 0 V		Separate process supply voltage ZP = 0 V	Separate process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see the description of the respective module used).

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	**Value**
Number of channels per module	Max. 32
Distribution of the channels into groups	4 groups of 8 channels each (1.0...1.7, 2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted I/O module
Rated voltage	24 V DC
Max. permitted total current	10 A, per separated process voltage terminal or for internal connection of process voltages
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting

Parameter	Value
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

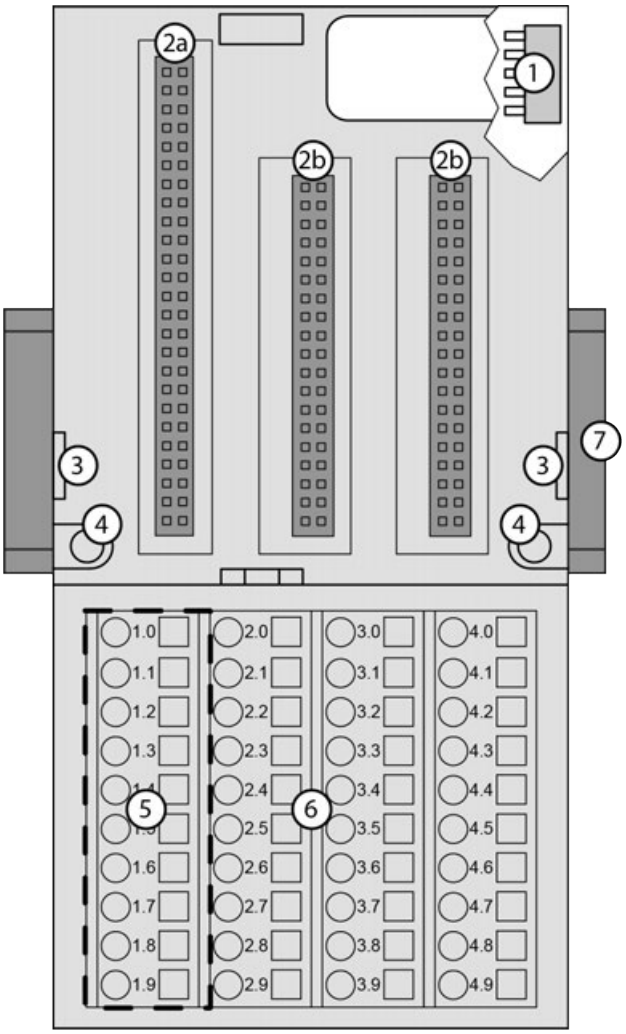
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 212 200 R0001	TU515, I/O terminal unit, 24 V DC, screw terminals	Active
1SAP 212 000 R0001	TU516, I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 412 000 R0001	TU516-XC, I/O terminal unit, 24 V DC, spring terminals, XC version	Active
1SAP 215 000 R0001	TU516-H, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version	Active
1SAP 415 000 R0001	TU516-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals	Active
1SAP 213 000 R0001	TU541, I/O terminal unit, 24 V DC, screw terminals	Active
1SAP 213 200 R0001	TU542, I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 413 200 R0001	TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version	Active
1SAP 215 200 R0001	TU542-H, I/O terminal unit, hot swap, 24 V DC, spring terminals	Active
1SAP 415 200 R0001	TU542-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.5.3 TU517 and TU518 for communication interface modules

- TU517, terminal unit, 24 V DC, screw terminals
- TU518, terminal unit, 24 V DC, spring terminals
- TU518-XC, terminal unit, 24 V DC, spring terminals, XC version



1 I/O bus (10 pins, female) to connect the first terminal unit
2a Plug (2 25 pins) to connect the inserted communication interface module
2b Plug (2 19 pins) to connect the inserted communication interface module
3 With a screwdriver, inserted in this place, the terminal unit and the adjacent I/O terminal unit can be shoved from each other
4 2 holes for wall mounting
5 10 terminals for connection with the bus system
6 30 terminals for signals and process supply voltages (UP and UP3)
7 DIN rail

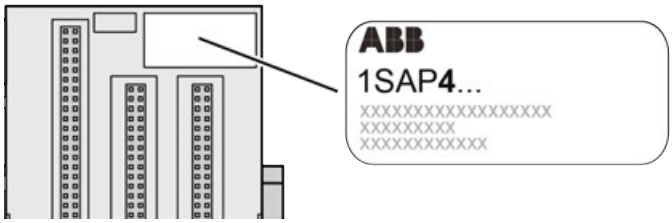
The communication interface modules plug into the terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the communication interface modules without disturbing the wiring at the terminal unit.

The terminal units TU517 and TU518 are specifically designed for use with AC500/S500 communication interface modules (e. g. CI581-CN, CI541-DP):

- CANopen communication interface modules
- DeviceNet modules
- PROFIBUS DP communication interface modules

XC version

XC = e**X**treme **C**onditions

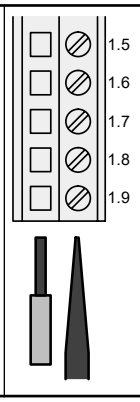
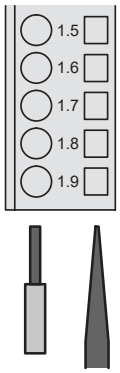


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			**Spring terminals**		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.4.6.4.3 “Terminals at the terminal unit” on page 3417.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.4.6.2.3 “Mechanical dimensions S500” on page 3406*

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted communication interface module:

- Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC
- Terminal 4.8: process supply voltage UP3 = +24 V DC
- Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see communication interface modules for CANopen and PROFIBUS).

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of I/O channels per module	Max. 24 (depending on the inserted communication interface module)
Distribution of the channels into groups	3 groups of max. 8 channels each (2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted communication interface module
Network interface connector	10 screw or spring terminals (1.0...1.9)
Rated voltage	24 V DC
Max. permitted total current	10 A via the supply terminals (UP, UP3 and ZP)
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

Ordering data

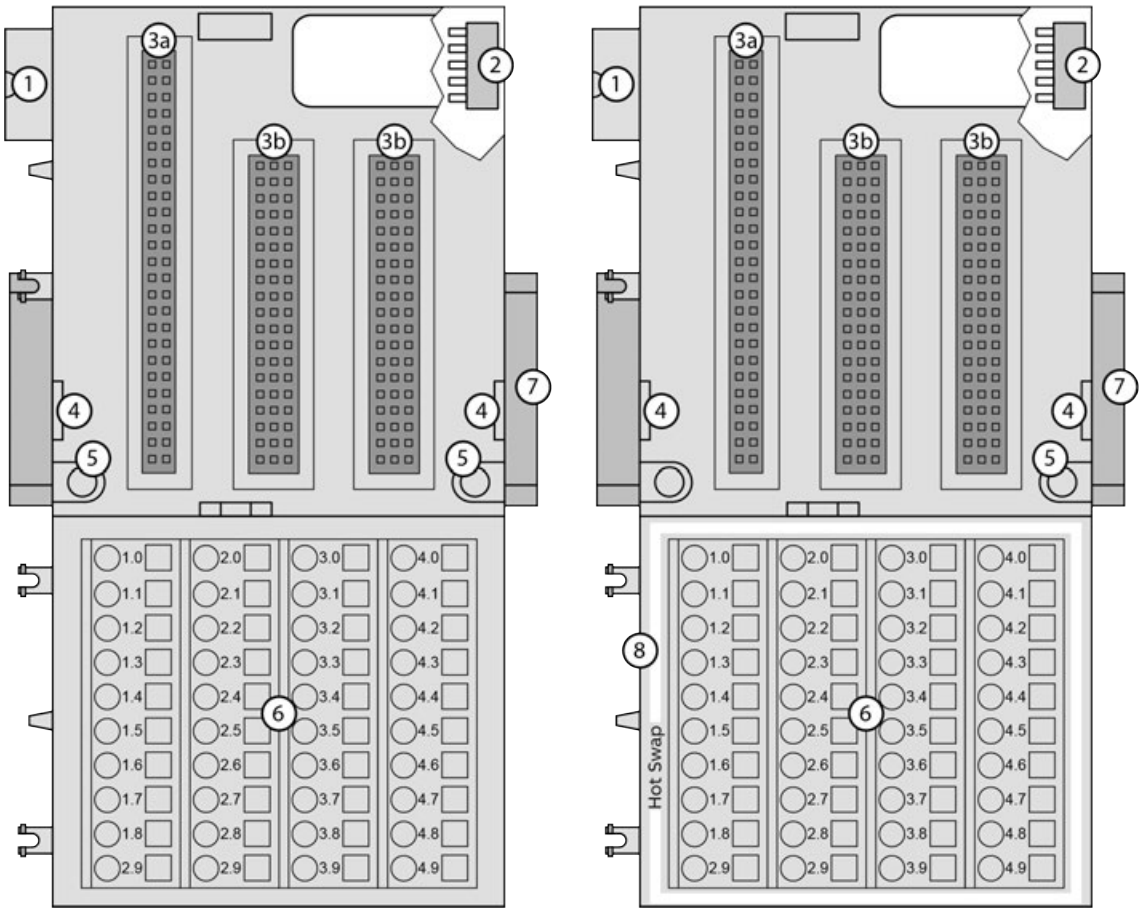
Part no.	Description	Product life cycle phase *)
1SAP 211 400 R0001	TU517, terminal unit, 24 V DC, screw terminals	Active
1SAP 211 200 R0001	TU518, terminal unit, 24 V DC, spring terminals	Active
1SAP 411 200 R0001	TU518-XC, terminal unit, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.5.4 TU531 and TU532 for I/O modules

- TU531, I/O terminal unit, 230 V AC, screw terminals
- TU532, I/O terminal unit, 230 V AC, spring terminals
- TU532-XC, I/O terminal unit, 230 V AC, spring terminals, XC version

- TU532-H, I/O terminal unit, hot swap, 230 V AC, spring terminals
- TU532-H-XC, I/O terminal unit, hot swap, 230 V AC, spring terminals, XC version



1 I/O bus (10 pins, male) to connect the previous terminal unit, the CPU terminal base or the communication interface module to the terminal unit
2 I/O bus (10 pins, female) to connect other terminal units
3a Plug (2 x 25 pins) to connect the inserted I/O modules
3b Plug (3 x 19 pins) to connect the inserted I/O modules
4 With a screwdriver inserted in this place, the terminal unit and the adjacent I/O terminal unit can be shoved from each other
5 Holes for screw mounting
6 40 terminals for signals and process supply voltage
7 DIN rail
8 White border signifies hot swap capability of the terminal unit

The input/output modules (I/O modules) plug into the I/O terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the I/O modules without disturbing the wiring at the terminal unit.

The terminal units TU531 and TU532 are specifically designed for use with AC500/S500 I/O modules that incorporate 115-230 V AC inputs and/or 230 V AC relay outputs.

Hot swap

WARNING!

Risk of explosion or fire in hazardous environments during hot swapping!

Hot swap must not be performed in flammable environments to avoid life-threatening injury and property damage resulting from fire or explosion.

WARNING!
Electric shock due to negligent behavior during hot swapping!
To avoid electric shock

- make sure the following conditions apply:
  - Digital outputs are not under load.
  - Input/output voltages above safety extra low voltage/ protective extra low voltage (SELV/PELV) are switched off.
  - Modules are fully interlocked with the terminal unit with both snap-fits engaged before switching on loads or input/output voltage.
- Never touch exposed contacts (dangerous voltages).
- Stay away from electrical contacts to avoid arc discharge.
- Do not operate a mechanical installation improperly.

NOTICE!
Risk of damage to I/O modules!
Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

H = **H**ot swap

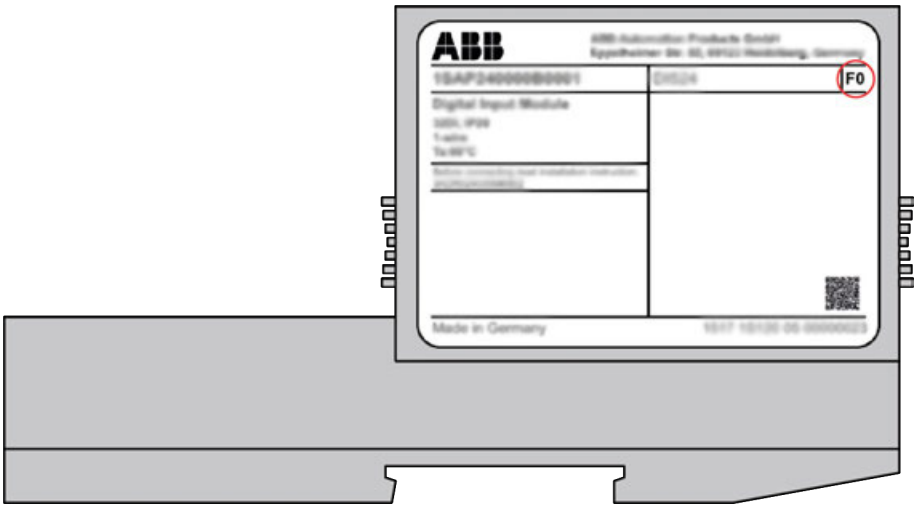
Hot swap
System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor modules PM56xx-2ETH with firmware version as of V3.2.0.*

Hot swap is not supported by AC500-eCo V3 CPU!



The index of the module is in the right corner of the label.

NOTICE!

Risk of damage to I/O modules!

Modules with index below F0 can be damaged when inserted or removed from the terminal unit in a powered system.

NOTICE!

Risk of damage to I/O modules!

Do not perform hot swapping if any I/O module with firmware version lower than 3.0.14 is part of the I/O configuration.

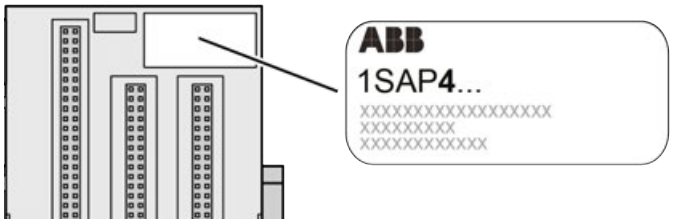
For min. required device index see table below.

Device	Min. required device index for I/O module as of FW Version 3.0.14
AC522(-XC)	F0
AI523 (-XC)	D2
AI531	D4
AI531-XC	D2
AI561	B2
AI562	B2
AI563	B3
AO523 (-XC)	D2
AO561	B2
AX521 (-XC)	D2
AX522 (-XC)	D2
AX561	B2
CD522 (-XC)	D1
DA501 (-XC)	D2
DA502 (-XC)	F0
DC522 (-XC)	D2
DC523 (-XC)	D2
DC532 (-XC)	D2
DC561	B2
DC562	A2
DI524 (-XC)	D2
DI561	B2
DI562	B2
DI571	B2

Device	Min. required device index for I/O module as of FW Version 3.0.14
DI572	A1
DO524 (-XC)	A3
DO526	A2
DO526-XC	A0
DO561	B2
DO562	A2
DO571	B3
DO572	B2
DO573	A1
DX522 (-XC)	D2
DX531	D2
DX561	B2
DX571	B3
FM562	A1

XC version

XC = e**X**treme **C**onditions

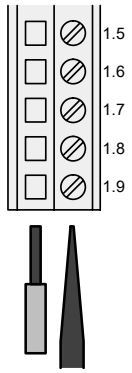
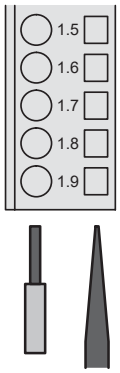


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			Spring terminals		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.4.6.4.3 “Terminals at the terminal unit” on page 3417.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.4.6.2.3 “Mechanical dimensions S500” on page 3406*

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the terminal unit and always have the same assignment, independent of the inserted module:

- Terminals 1.8 to 4.8: process supply voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see the description of the respective module used).

The supply voltage of 24 V DC for the module's circuitry comes from the I/O expansion bus (I/O bus).

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Number of channels per module		32
Distribution of the channels into groups		4 groups of 8 channels each (1.0...1.7, 2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted I/O module
Terminals 1.8...4.8 and 1.9...4.9		
	Max. voltage	30 V DC
	Max. permitted total current	10 A
Terminals 1.0...1.7, 2.0...2.7, 3.0...3.7, 4.0...4.7		
	Max. voltage	300 V AC [1])
	Max. permitted current	3 A [2])
Grounding		Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals		Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals		Front terminal, conductor connection vertically with respect to the printed circuit board
Weight		200 g
Mounting position		Horizontal or vertical

[1]) Only when the voltage is not limited by the specification of the I/O channel or the supply input which is internally connected to the terminal.

[2]) The terminals are connected to the electronic module via internal connectors (X22 (or 3b), X23 (or 3b), X32, X33 and X34). The current per terminal is limited by the permitted current of these connectors.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 217 200 R0001	TU531, terminal unit, 230 V AC, relays, screw terminals	Active
1SAP 217 000 R0001	TU532, terminal unit, 230 V AC, relays, spring terminals	Active
1SAP 417 000 R0001	TU532-XC, terminal unit, 230 V AC, relays, spring terminals, XC version	Active
1SAP 215 100 R0001	TU532-H, terminal unit, hot swap, 230 V AC, relays, spring terminals	Active
1SAP 415 100 R0001	TU532-H-XC, terminal unit, hot swap, 230 V AC, relays, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.6 I/O modules

Hot swap

System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor modules PM56xx-2ETH with firmware version as of V3.2.0.*

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

Conditions for hot swapping

- *Digital outputs are not under load.*
- *Input/output voltages above safety extra low voltage/ protective extra low voltages (SELV/PELV) are switched off.*
- *Modules are completely plugged on the terminal unit with both snap fit engaged before switching on loads or input/output voltage.*

Hot swap

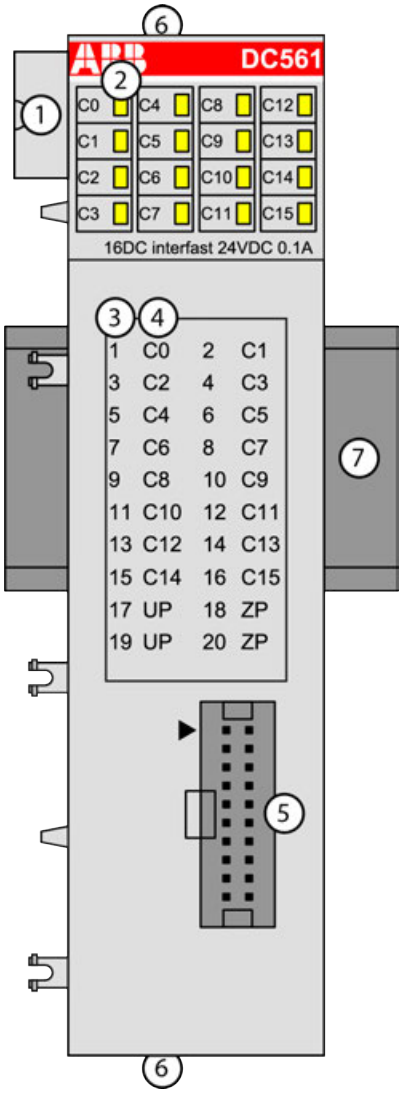
Further information about hot swap: ↬ Chapter 1.6.5.1.8 “Hot swap” on page 3523.

1.6.3.6.1 Digital I/O modules

S500-eCo

DC561 - Digital input/output module

- 16 configurable digital inputs/outputs 24 V DC,
- Connection via Interfast
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the states of the inputs/outputs C0 to C15
3 Terminal number
4 Allocation of signal name
5 Interfast connector (20-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The digital I/O module DC561 can be connected to the following devices via the I/O bus connector:

- S500 communication interface modules (e. g. CI501-PNIO, CI541-DP, CI581-CN)
- AC500 CPUs
- other AC500 I/O modules

> *The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.*

The module contains 16 digital channels in 1 group, each channel can be used as a digital 24 V DC input or 24 V DC output.

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

Functionality

Parameter	Value
Digital inputs	Max. 16 (24 V DC), can be used as sink inputs
Digital outputs	Max. 16 (transistor outputs 24 V DC, max. 0.1 A)
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

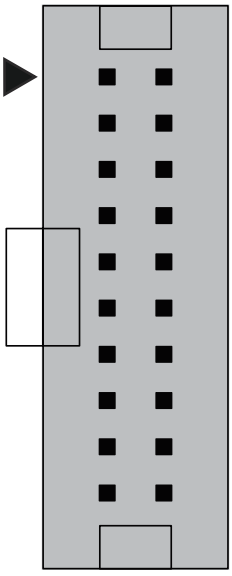
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 "AC500-eCo" on page 3352.

The connection is established out by using the 20-pin Interfast connector. For further information, refer to the Interfast documentation.

The assignment of the terminals:

Table 467: Assignment of the terminals for DC561



PIN	Signal	Description
1	C0	Input/output signal C0
2	C1	Input/output signal C1
3	C2	Input/output signal C2
4	C3	Input/output signal C3
5	C4	Input/output signal C4
6	C5	Input/output signal C5
7	C6	Input/output signal C6
8	C7	Input/output signal C7
9	C8	Input/output signal C8
10	C9	Input/output signal C9
11	C10	Input/output signal C10
12	C11	Input/output signal C11
13	C12	Input/output signal C12
14	C13	Input/output signal C13
15	C14	Input/output signal C14
16	C15	Input/output signal C15
17	UP	Process voltage UP +24 V DC
18	ZP	Process voltage ZP 0 V DC

	PIN	Signal	Description
	19	UP	Process voltage UP +24 V DC
	20	ZP	Process voltage ZP 0 V DC

The arrow located next to the Interfast connector marks terminal 1.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DC561.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Process supply voltage must be connected to UP/ZP of the module. The inputs and UP/ZP must use the same power supply.

If DC561 with index A0 is used, the process supply voltage must stem from the same source as the power supply voltage of the CPU. The index consists of 1 letter, followed by 1 digit, and can be found on the type plate of the module next to the type designator "DC561".

The module provides several diagnosis functions *Chapter 1.6.3.6.1.1.1.6 “Diagnosis” on page 2574*.

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.3.6.1.1.1.7 “State LEDs” on page 2574*.

I/O Configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6100 [1])	WORD	6100 0x17D4	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

Remarks:

[1])	With CS31 and addresses smaller than 70, the value is increased by 1
[2])	The module has no additional user-configurable parameters
[3])	Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0x25, 0x17, 0x00;

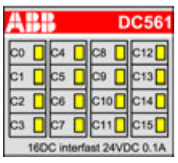
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error DI571							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Inputs/outputs C0...C15	Digital input or digital output	Yellow	Input/output is OFF	Input/output is ON (the LEDs are only operating if the module's circuitry is supplied via the I/O bus)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process voltage UP		
	Connections	Terminals 17 and 19 for UP (+24 V DC); terminals 18 and 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption via UP terminal	10 mA + 0.1 A per output (max.)
	Max. ripple	5 %
	Inrush current	0.000001 A^2s
	Protection against reversed voltage	Yes
	Protection fuse on UP	Recommended; the outputs must be protected by an 1 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the input/output group and the rest of the module
Isolated groups		1 group for 16 channels
Surge voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		On request
Input data length		2 bytes
Output data length		2 bytes
Weight		Ca. 115 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs/outputs if used as inputs

Parameter	Value
Number of channels per module	16 configurable inputs (24 V DC)
Distribution of the channels into groups	1 (16 channels per group)
Connections of the channels C0 to C15	Terminals 1 to 16
Reference potential for the channels C0 to C15	Terminals 18 and 20 (negative pole of the process voltage, name ZP)

Parameter		Value
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered via the I/O bus.
Input type according to EN 61131-2		Type 1 sink
Input signal range		+24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	+5 V...+15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		-3 V...+5 V
Ripple with signal 1		+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	Typ. 1 mA
	Input voltage +15 V	> 2.5 mA
	Input voltage +30 V	< 8 mA
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Input delay (0->1 or 1->0)		Typ. 8 ms
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		16 configurable transistor outputs
Distribution of the channels into groups		1 (16 channels per group)
Connections of the channels C0 to C15		Terminals 1 to 16
Reference potential for the channels C0 to C15		Terminals 18 and 20 (negative pole of the process voltage, signal name ZP)
Common power supply voltage		Terminals 17 and 19 (positive pole of the process voltage, signal name UP)
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered via the I/O bus.
Way of operation		Non-latching type
Output voltage at signal 1		UP -0.3 V at max. current
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output current		
	Rated current per channel (max.)	0.1 A at UP 24 V DC
	Rated current per group (max.)	1.6 A

Parameter		Value
	Rated current (all channels together, max.)	1.6 A
	Lamp load (max.)	Not applicable
	Max. leakage current with signal 0	< 0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		1 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to load specification
Switching frequency		
	With inductive loads	Max. 0.5 Hz
Short-circuit-proof / overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC signals	Yes
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

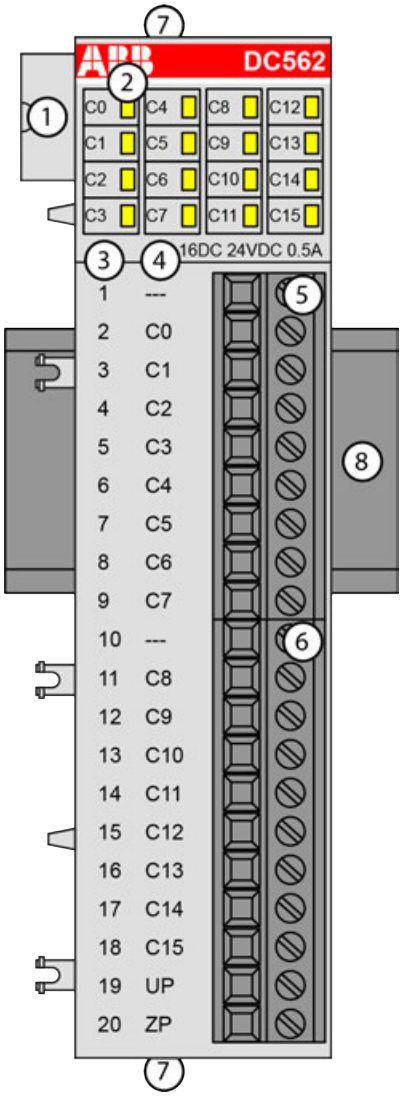
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2001	DC561, digital input/output module, 16 configurable inputs/outputs, transistor output, interfast connector	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC562 - Digital input/output module

- 16 configurable digital inputs/outputs in 1 group, 24 V DC
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the states of the inputs/outputs C0 to C15
3 Terminal number
4 Allocation of signal name
5 Terminal block for input and output signals (9-pin)
6 Terminal block for input and output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↪ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs and outputs:

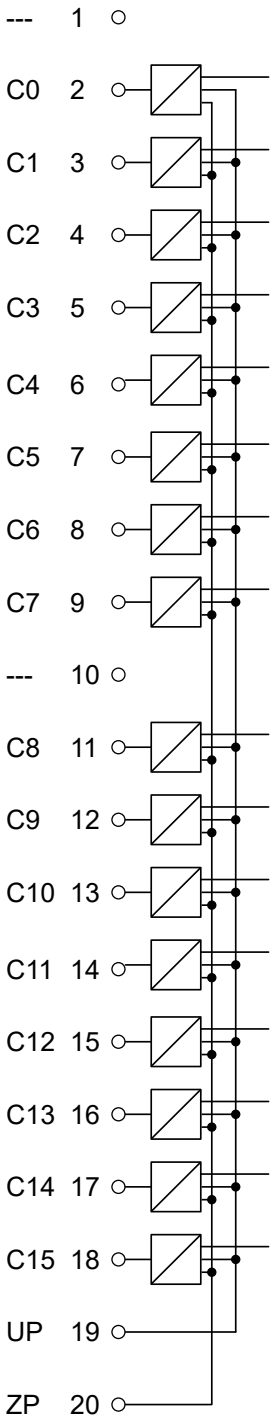


Table 468: Assignment of the terminals:

Terminal	Signal	Description
1	---	Reserved
2	C0	Input/output signal C0
3	C1	Input/output signal C1
4	C2	Input/output signal C2
5	C3	Input/output signal C3
6	C4	Input/output signal C4
7	C5	Input/output signal C5

Terminal	Signal	Description
8	C6	Input/output signal C6
9	C7	Input/output signal C7
10	---	Reserved
11	C8	Input/output signal C8
12	C9	Input/output signal C9
13	C10	Input/output signal C10
14	C11	Input/output signal C11
15	C12	Input/output signal C12
16	C13	Input/output signal C13
17	C14	Input/output signal C14
18	C15	Input/output signal C15
19	UP	Process voltage UP +24 V DC
20	ZP	Process voltage ZP 0 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DC562.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

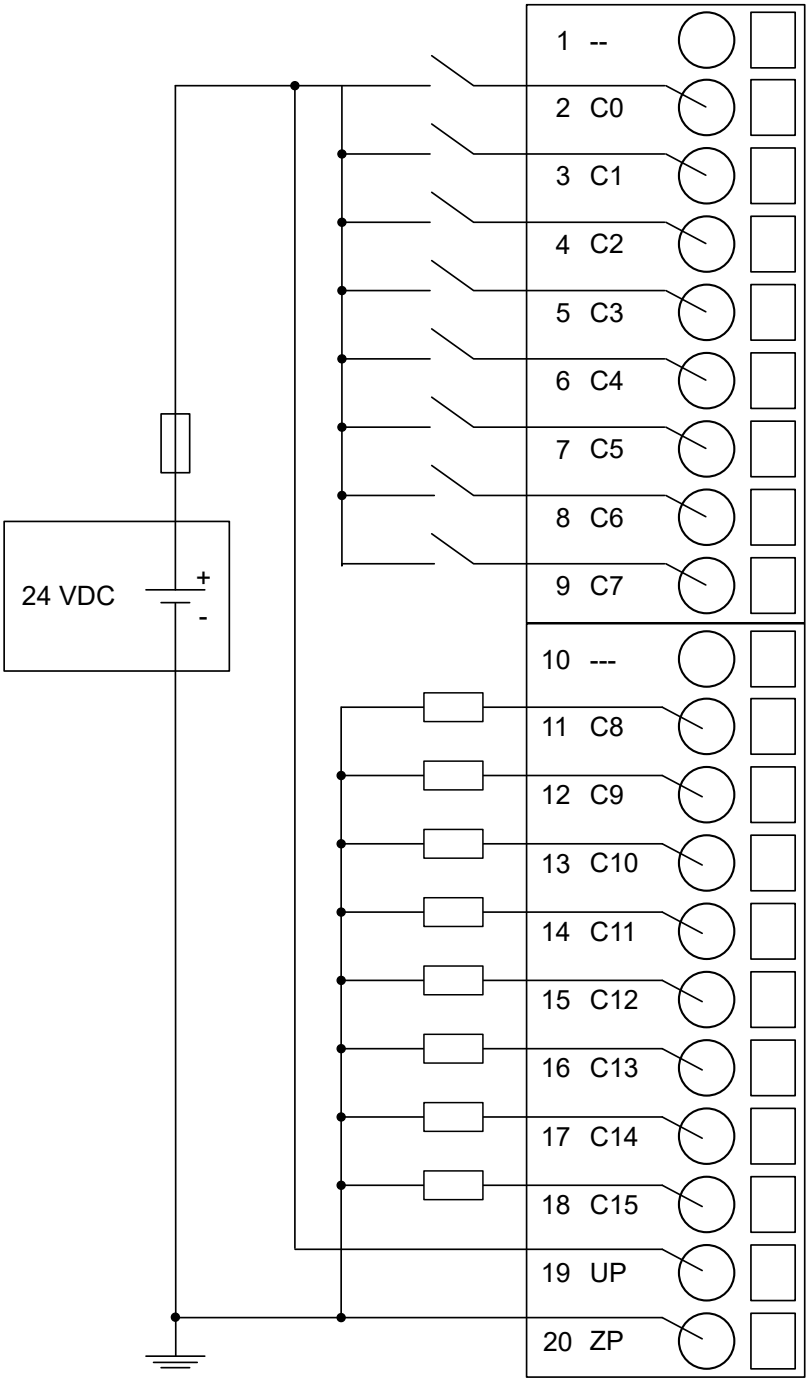
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Process supply voltage must be connected to UP/ZP of the module. The inputs and UP/ZP must use the same power supply.

The following figure shows the connection of the digital input/output module DC562:



In this connection example, the inputs/outputs C0...C7 are connected as inputs and the inputs/outputs C8...C15 are connected as outputs.

The module provides several diagnosis functions *Chapter 1.6.3.6.1.1.2.6 “Diagnosis” on page 2584*.

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.3.6.1.1.2.7 “State LEDs” on page 2584*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6155 [1])	WORD	6155 0x180B	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) the module has no additional user-configurable parameters

[3]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x06
Ext_User_Prm_Data_Const(0) =	0x18, 0x0C, 0x00, 0x02, 0x00, 0x00;

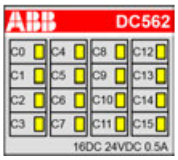
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error DC562							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Inputs/outputs C0...C15	Digital input or digital output	Yellow	Input/output is OFF	Input/output is ON (the LEDs are only operating if the module's circuitry is supplied via the I/O bus)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption via UP terminal	90 mA + 0.5 A per output (max.)
	Max. ripple	5 %
	Inrush current	0.000001 A^2s
	Protection against reversed voltage	Yes
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the input/output group and the rest of the module
	Isolated groups	1 group for 16 channels
Surge voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		4.8 W
Input data length		2 bytes
Output data length		2 bytes
Weight		Ca. 125 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs/outputs if used as inputs

Parameter	Value
Number of channels per module	16 configurable inputs (24 V DC)
Distribution of the channels into groups	1 (16 channels per group)
Connections of the channels C0 to C15	Terminals 1 to 16
Reference potential for the channels C0 to C15	Terminal 20 (negative pole of the process voltage, name ZP)
Indication of the input signals	1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.
Input type according to EN 61131-2	Type 1 sink

Parameter		Value
Input signal range		+24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	+5 V...+15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		-3 V...+5 V
Ripple with signal 1		+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	Typ. 1 mA
	Input voltage +15 V	> 2.5 mA
	Input voltage +30 V	< 8 mA
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Input delay (0->1 or 1->0)		Typ. 8 ms
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		16 configurable transistor outputs
Distribution of the channels into groups		1 (16 channels per group)
Connections of the channels C0 to C15		Terminals 1 to 16
Reference potential for the channels C0 to C15		Terminal 20 (negative pole of the process voltage, signal name ZP)
Common power supply voltage		Terminal 19 (positive pole of the process voltage, signal name UP)
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.
Way of operation		Non-latching type
Output voltage at signal 1		UP -0.3 V at max. current
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	8 A
	Rated current (all channels together, max.)	8 A
	Lamp load (max.)	5 W

Parameter		Value
	Max. leakage current with signal 0	< 0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching frequency		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC signals	Yes
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

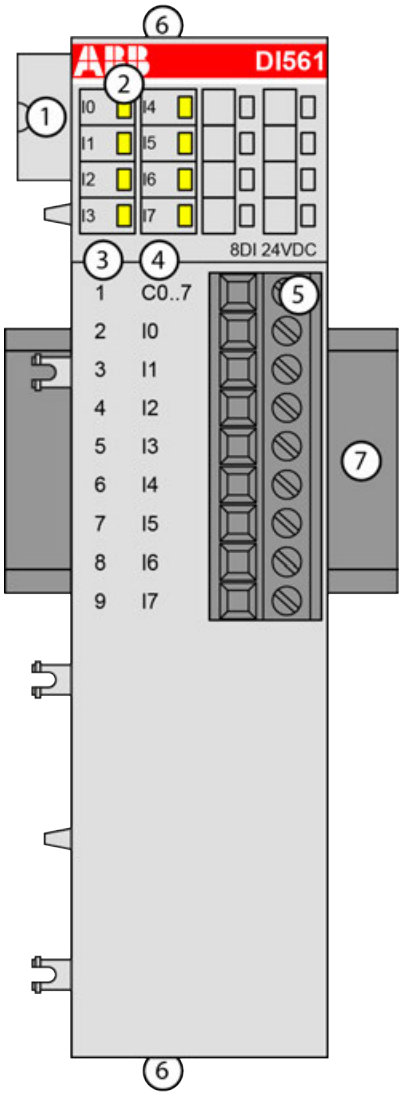
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 231 900 R0000	DC562, digital input/output module, 16 configurable inputs/outputs, transistor output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI561 - Digital input module

- 8 digital inputs 24 V DC / 24 V AC (I0 to I7) in 1 group
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using a removable 9-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs:

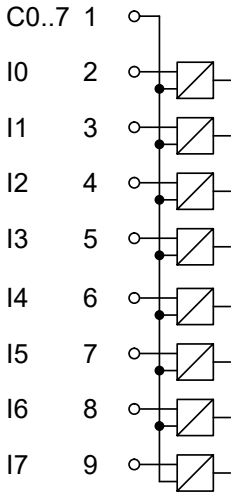


Table 469: Assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1

Terminal	Signal	Description
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI561.

An external power supply connection is not needed.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The digital inputs can be used as source inputs or as sink inputs.

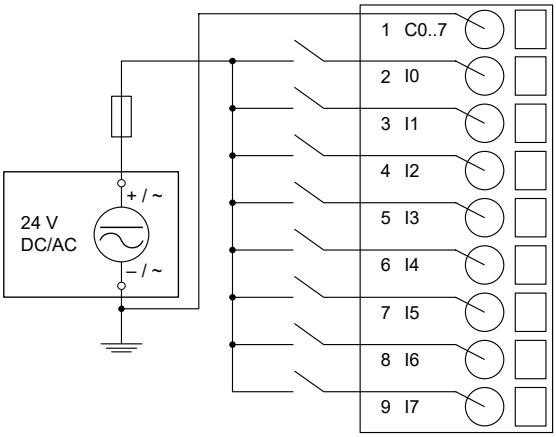
NOTICE!

Risk of malfunctions in the plant!

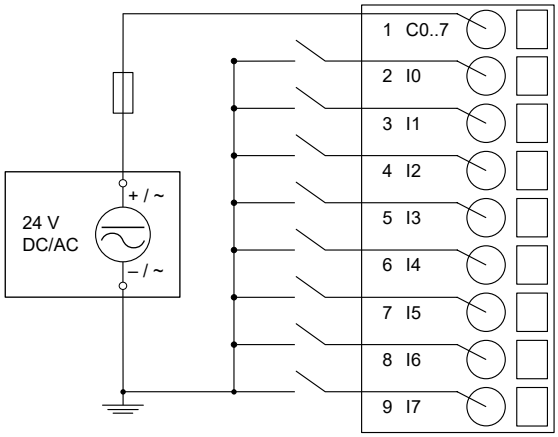
A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the digital input module DI561:



Connection of DI561 - sink inputs



Connection of DI561 - source inputs

The module provides several diagnosis functions *Chapter 1.6.3.6.1.1.3.6 "Diagnosis" on page 2592*.

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.3.6.1.1.3.7 "State LEDs" on page 2593*.

I/O Configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6105 [1])	WORD	6105 0x17D9	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) the module has no additional user-configurable parameters

[3]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xDA, 0x17, 0x00;

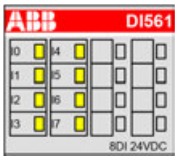
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON

In the undefined signal range, the state LED for the inputs can be ON although the input state detected by the module is OFF.

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input group and the rest of the module
Isolated groups	1 (8 channels per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	1.6 W
Weight	Ca. 110 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter	Value		
Number of channels per module	8 inputs (24 V DC / 24 V AC)		
Distribution of the channels into groups	1 (8 channels per group)		
Connections of the channels I0 to I7	Terminals 2 to 9		
Reference potential for the channels I0 to I7	Terminal 1 (plus or negative pole of the process supply voltage, signal name C0..7)		
Indication of the input signals	1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.		
Monitoring point of input indicator	LED is part of the input circuitry		
Input type according to EN 61131-2	Type 1 source	Type 1 sink	Type 1 AC [1])

Parameter		Value		
Input signal range		-24 V DC	+24 V DC	24 V AC 50/60 Hz
	Signal 0	-5 V...+3 V	-3 V...+5 V	0 V AC...5 V AC
	Undefined signal	-15 V...-5 V	+5 V...+15 V	5 V AC...14 V AC
	Signal 1	-30 V...-15 V	+15 V...+30 V	14 V AC...27 V AC
Input current per channel				
	Input voltage 24 V	Typ. 5 mA		Typ. 5 mA r.m.s.
	Input voltage 5 V	Typ. 1 mA		Typ. 1 mA r.m.s.
	Input voltage 14 V			Typ. 2.7 mA r.m.s.
	Input voltage 15 V	> 2.5 mA		
	Input voltage 27 V			Typ. 5.5 mA r.m.s.
	Input voltage 30 V	< 8 mA		
Max. permissible leakage current (at 2-wire proximity switches)		1 mA		Typ. 1 mA r.m.s.
Input delay (0->1 or 1->0)		Typ. 8 ms		
Input data length		1 byte		
Max. cable length				
	Shielded	500 m		
	Unshielded	300 m		

[1]) When inputs are used with 24 V AC, external surge limiting filters are required.

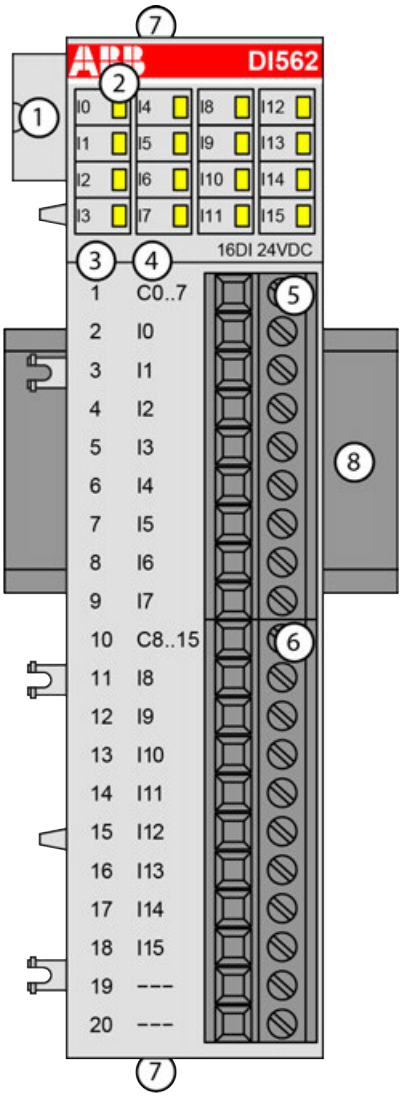
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2101	DI561, digital input module, 8 DI, 24 V DC / 24 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DI562 - Digital input module

- 16 digital inputs 24 V DC / 24 V AC (I0 to I15) in 2 groups
- Group-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the inputs I0 to I15
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

The other electronic circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

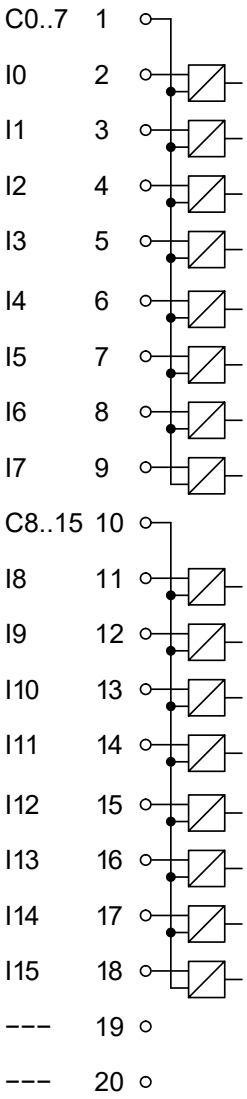
Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw-type terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs:



The assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7
10	C8...15	Input common for signals I8 to I15
11	I8	Input signal I8
12	I9	Input signal I9
13	I10	Input signal I10
14	I11	Input signal I11
15	I12	Input signal I12
16	I13	Input signal I13
17	I14	Input signal I14
18	I15	Input signal I15
19	---	Reserved
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI562.

An external power supply connection is not needed.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.3.6.1.1.4.6 “Diagnosis” on page 2600*.

The digital inputs can be used as source inputs or as sink inputs.

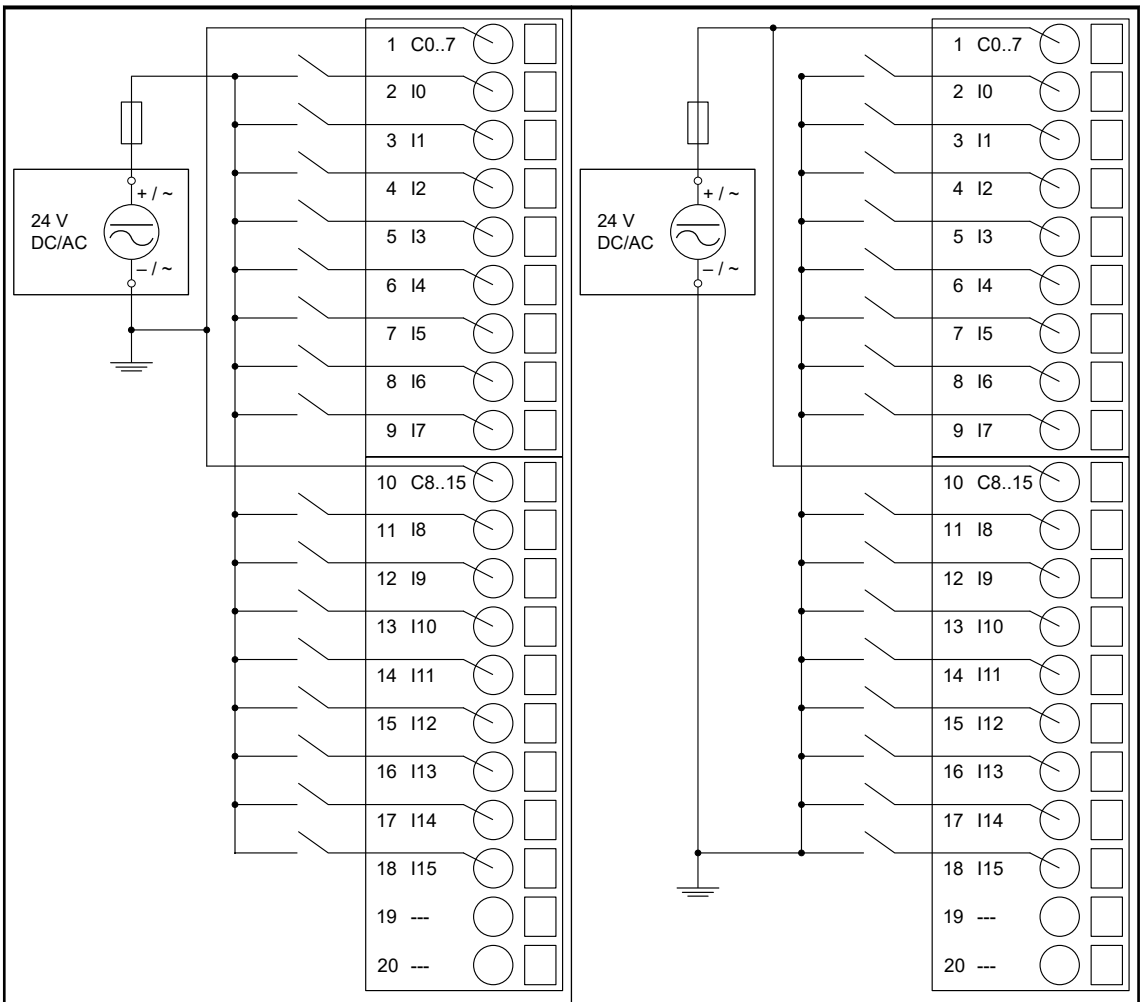
NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the digital input module DI562:



Connection of DI562 - sink inputs | Connection of DI562 - source inputs

The meaning of the LEDs is described in section State LEDs *Chapter 1.6.3.6.1.1.4.7 “State LEDs” on page 2600*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6110 [1])	WORD	6110 0x17DE	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

Remarks:

[1])	With CS31 and addresses less than 70, the value is increased by 1
[2])	The module has no additional user-configurable parameters
[3])	Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xDF, 0x17, 0x00;

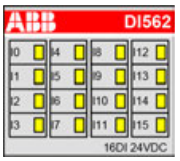
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error DI562							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Inputs I0...I15	Digital input	Yellow	Input is OFF	Input is ON

In the undefined signal range, the state LED for the inputs can be ON although the input state detected by the module is OFF.

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input groups and the rest of the module
Isolated groups	2 (8 channels per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	3.2 W
Weight	Ca. 115 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter	Value		
Number of channels per module	16 inputs (24 V DC / 24 V AC)		
Distribution of the channels into groups	2 (8 channels per group)		
Connections of the channels I0 to I7	Terminals 2 to 9		
Connections of the channels I8 to I15	Terminals 11 to 18		
Reference potential for the channels I0 to I7	Terminal 1 (positive or negative pole of the process supply voltage, signal name C0..7)		
Reference potential for the channels I8 to I15	Terminal 10 (positive or negative pole of the process supply voltage, signal name C8..15)		
Indication of the input signals	1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.		
Monitoring point of input indicator	LED is part of the input circuitry		
Input type according to EN 61131-2	Type 1 source	Type 1 sink	Type 1 AC [1])
Input signal range	-24 V DC	+24 V DC	24 V AC 50/60 Hz
Signal 0	-5 V...+3 V	-3 V...+5 V	0 V AC...5 V AC
Undefined signal	-15 V...-5 V	+5 V...+15 V	5 V AC...14 V AC

Parameter		Value		
	Signal 1	-30 V...-15 V	+15 V...+30 V	14 V AC...27 V AC
Input current per channel				
	Input voltage 24 V	Typ. 5 mA		Typ. 5 mA r.m.s.
	Input voltage 5 V	Typ. 1 mA		Typ. 1 mA r.m.s.
	Input voltage 14 V			Typ. 2.7 mA r.m.s.
	Input voltage 15 V	> 2.5 mA		
	Input voltage 27 V			Typ. 5.5 mA r.m.s.
	Input voltage 30 V	< 8 mA		
Max. permissible leakage current (at 2-wire proximity switches)		1 mA		Typ. 1 mA r.m.s.
Input delay (0->1 or 1->0)		Typ. 8 ms		
Input data length		2 bytes		
Max. cable length				
	Shielded	500 m		
	Unshielded	300 m		

[1]) When inputs are used with 24 V AC, external surge limiting filters are required.

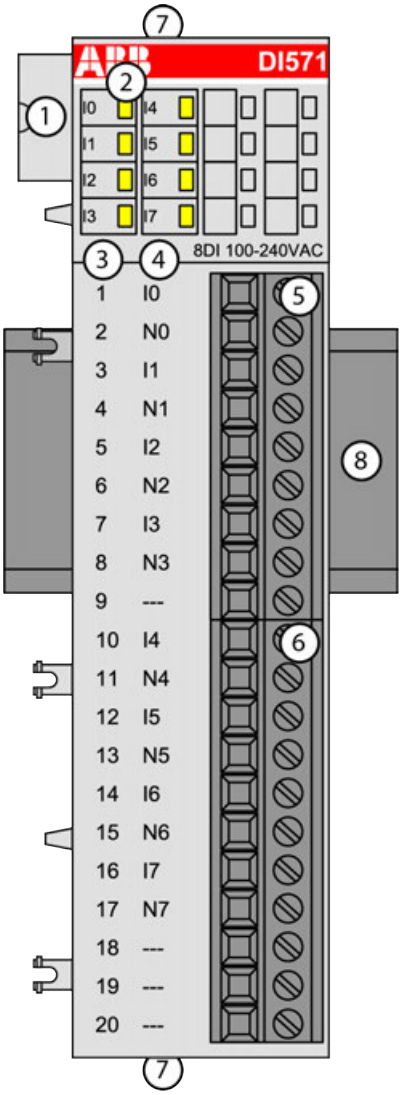
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2102	DI562, digital input module, 16 DI, 24 V DC / 24 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI571 - Digital input module

- 8 digital inputs 100-240 V AC (I0 to I7) in 8 groups
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs:

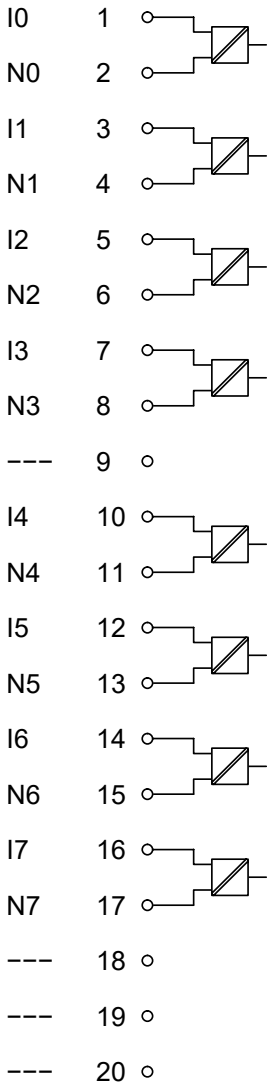


Table 470: Assignment of the terminals:

Terminal	Signal	Description
1	I0	Input signal I0
2	N0	Neutral conductor for the input signal I0
3	I1	Input signal I1
4	N1	Neutral conductor for the input signal I1
5	I2	Input signal I2
6	N2	Neutral conductor for the input signal I2
7	I3	Input signal I3
8	N3	Neutral conductor for the input signal I3
9	---	Reserved
10	I4	Input signal I4
11	N4	Neutral conductor for the input signal I4
12	I5	Input signal I5
13	N5	Neutral conductor for the input signal I5
14	I6	Input signal I6

Terminal	Signal	Description
15	N6	Neutral conductor for the input signal I6
16	I7	Input signal I7
17	N7	Neutral conductor for the input signal I7
18	---	Reserved
19	---	Reserved
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI571.

An external power supply connection is not needed.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

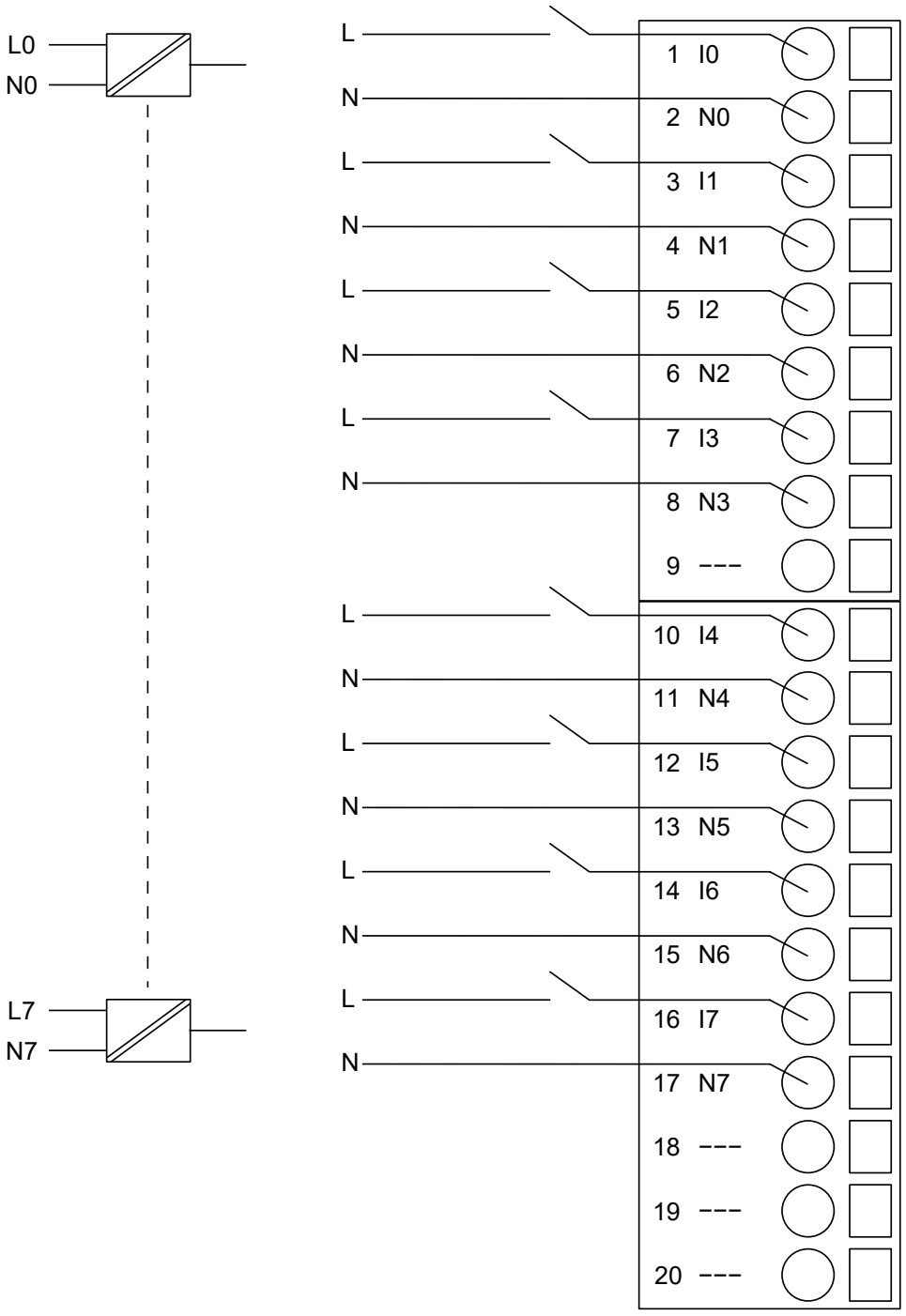
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the digital input module DI571:



NOTICE!

Risk of damaging the PLC modules!

The PLC modules will be irreparably damaged if a voltage > 240 V is connected.

Make sure that all inputs are fed from the same phase. The module must not be connected to a 400 V voltage.

The module provides several diagnosis functions ↳ *Chapter 1.6.3.6.1.1.5.7 “Diagnosis” on page 2609*.

The meaning of the LEDs is described in the section State LEDs ↳ *Chapter 1.6.3.6.1.1.5.8 “State LEDs” on page 2609*.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	0

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of the modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6115 [1])	WORD	6115 0x17E3	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) the module has no additional user-configurable parameters

[3]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xDF, 0x17, 0x00;

Diagnosis

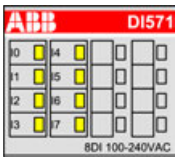
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON (the input voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Galvanic isolation		Yes, between the channels and the rest of the module
	Isolated groups	8 (1 channel per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Max. power dissipation within the module		On request
Weight		Ca. 135 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8 AC inputs (100-240 V AC)
Distribution of the channels into groups		8 (1 channel per group)
Input voltage range		0 V AC..264 V AC (47 Hz...63 Hz)
Input current per channel (typically at 25 °C)		<5 mA (at 40 V AC) >6 mA (at 159 V AC, 50 Hz) >7 mA (at 159 V AC, 60 Hz)
Connections of the channels I0 to I7		Terminals 1, 3, 5, 7, 10, 12, 14, 16
Reference potential for the channels I0 to I7		Terminals 2, 4, 6, 8, 11, 13, 15, 17
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)
Input type according to EN 61131-2		Type 1
Input signal range		
	Signal 0 (max.)	20 V AC
	Undefined signal	20 V AC < U < 79 V AC
	Signal 1 (min.)	79 V AC
Input delay		
	Signal 0 -> 1	Typ. 15 ms
	Signal 1 -> 0	Typ. 30 ms
Input data length		1 byte
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Max. cable length		

Parameter		Value
	Shielded	500 m
	Unshielded	300 m

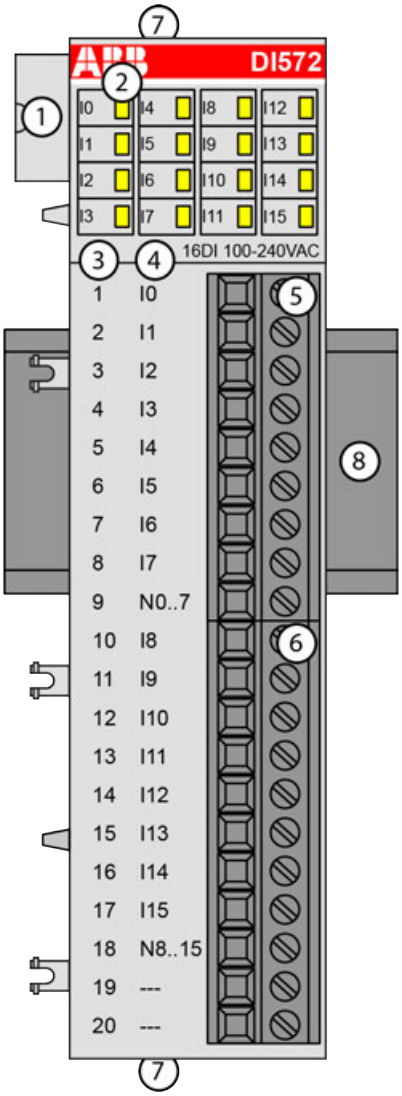
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2103	DI571, digital input module, 8 DI, 100 V AC...240 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI572 - Digital input module

- 16 digital inputs 100-240 V AC (I0 to I15) in 2 groups
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the inputs I0 to I15
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

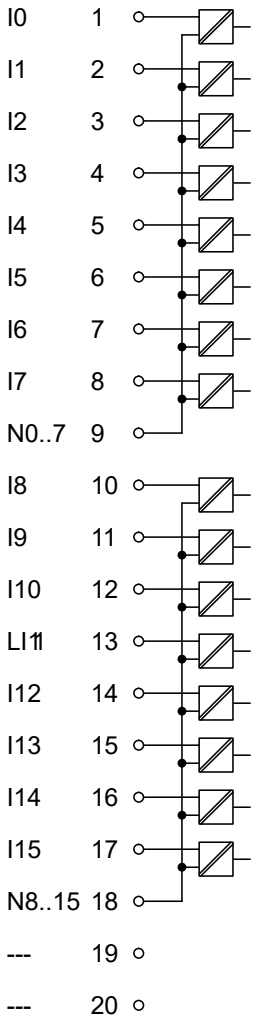


Fig. 104: Block diagram for the internal construction of the digital inputs.

Table 471: Assignment of the terminals

Terminal	Signal	Description
1	I0	Input signal I0
2	I1	Input signal I1
3	I2	Input signal I2
4	I3	Input signal I3
5	I4	Input signal I4
6	I5	Input signal I5
7	I6	Input signal I6
8	I7	Input signal I7
9	N0...7	Neutral conductor for the input signals I0...I7
10	I8	Input signal I8
11	I9	Input signal I9
12	I10	Input signal I10
13	I11	Input signal I11
14	I12	Input signal I12
15	I13	Input signal I13
16	I14	Input signal I14
17	I15	Input signal I15
18	N8...15	Neutral conductor for the input signals I8...I15
19	---	Reserved
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI572.

An external power supply connection is not needed.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

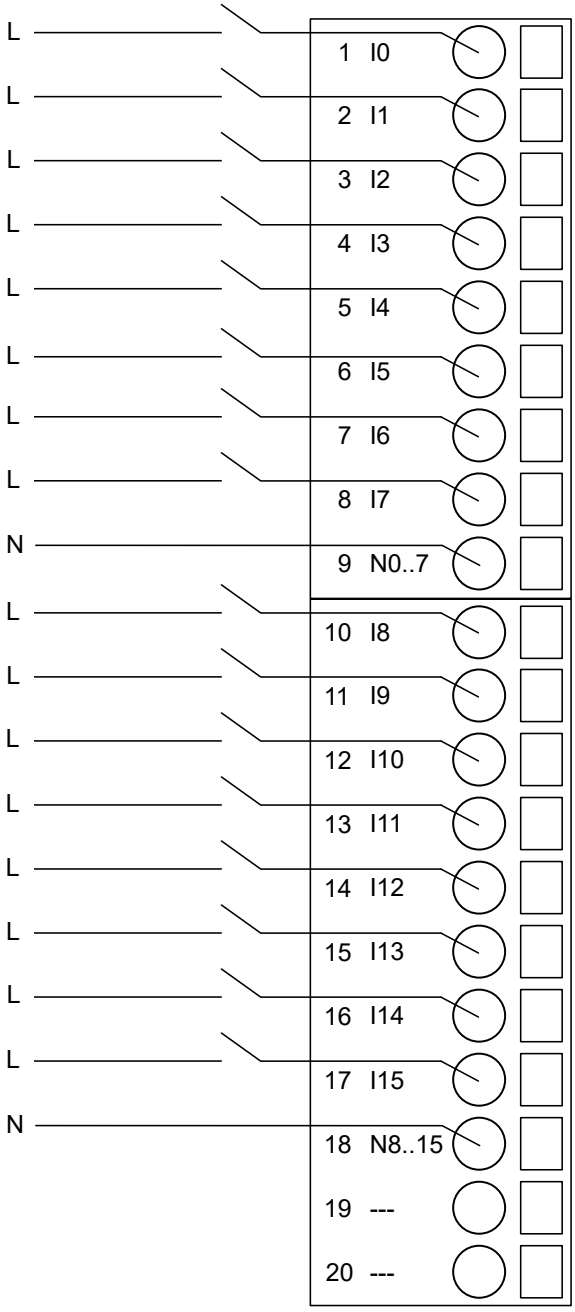
The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.



NOTICE!

Risk of damaging the PLC modules!

The PLC modules will be irreparably damaged if a voltage > 240 V is connected.

Make sure that all inputs are fed from the same phase. The module must not be connected to a 400 V voltage.

The module provides several diagnosis functions ↬ *Chapter 1.6.3.6.1.1.6.6 “Diagnosis” on page 2618*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Parameter name	Value	Internal value	Data type of internal value	Default value	Min.	Max.	EDS Slot Index
Module ID	Internal	6160 [1])	WORD	6160 0x1810	0	65535	xx01 [2])
Ignore module	No	0	BYTE	No 0x00	-	-	-
	Yes	1					
Parameter length	Internal	3	BYTE	3	0	255	xx02 [2])
Input delay	20 ms	0	BYTE	20 ms 0x00	0	1	-
	100 ms	1					

[1]) With CS31 and addresses less than 70, the value is increased by 1.

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n).

GSD file:

Ext_Module_Prm_Data_Len =	7
Ext_User_Prm_Data_Const(0) =	0x18, 0x11, 0x00, 0x03, 0x00, 0x00, 0x00;

Diagnosis

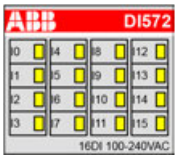
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block	

Class	Interface	Device	Module	Channel	Error Identifier	Error message	Remedy
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

Parameter	Remark
[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I15	Digital input	Yellow	Input is OFF	Input is ON (the input voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input groups and the rest of the module
Isolated groups	2 (8 channels per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	6 W
Weight	Ca. 222 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		16 AC inputs (100-240 V AC)
Distribution of the channels into groups		2 (8 channels per group)
Input voltage range		0 V AC...264 V AC (47 Hz...63 Hz)
Input current per channel (typically at 25 °C)		< 3 mA (at 40 V AC) > 6 mA (at 164 V AC) > 8 mA (at 240 V AC)
Connections of the channels I0..I7		Terminals 1...8
Connections of the channels I8...I15		Terminals 10...17
Reference potential for the channels I0...I7		Terminal 9
Reference potential for the channels I8...I15		Terminal 18
Indication of the input signals		1 yellow LED per channel. The LED is on when the input signal is high (signal 1).
Input type according to EN 61131-2		Type 1
Input signal range		
	Signal 0 (max.)	40 V AC
	Undefined signal	40 V AC < U < 79 V AC
	Signal 1 (min.)	79 V AC
Input delay		
	Signal 0 -> 1	Typ. 24 ms
	Signal 1 -> 0	Typ. 24 ms
Input data length		2 bytes

Parameter		Value
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

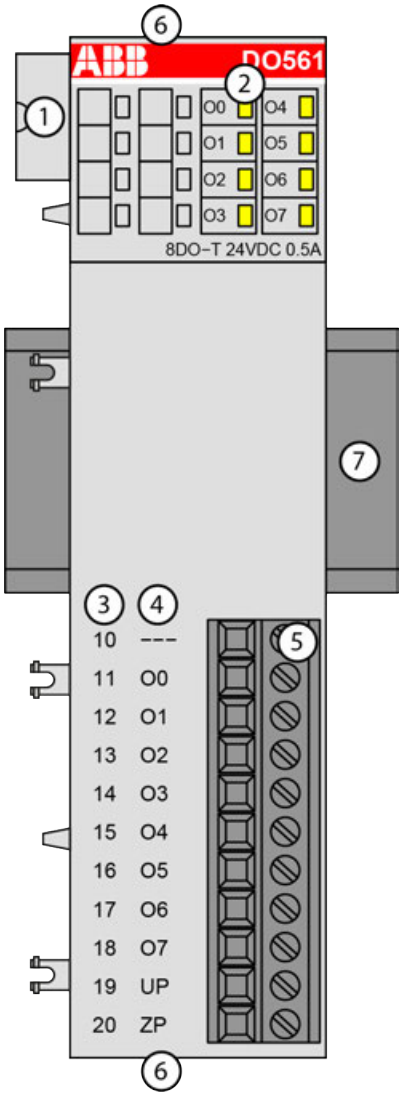
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 230 500 R0000	DI572, digital input module, 16 DI, 100 V AC...240 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO561 - Digital output module

- 8 digital outputs 24 V DC (O0 to O7) in 1 group
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the outputs O0 to O7
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital outputs:

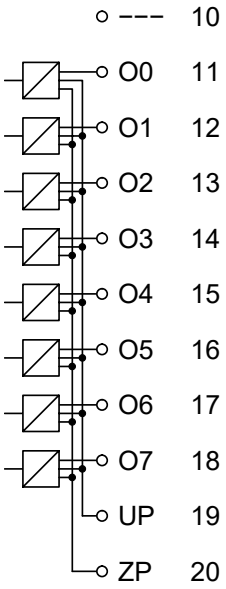


Table 472: Assignment of the terminals:

Terminals	Signal	Description
10	---	Reserved
11	O0	Output signal O0
12	O1	Output signal O1
13	O2	Output signal O2
14	O3	Output signal O3
15	O4	Output signal O4

Terminals	Signal	Description
16	O5	Output signal O5
17	O6	Output signal O6
18	O7	Output signal O7
19	UP	Process supply voltage UP +24 V DC
20	ZP	Process supply voltage ZP 0 V

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DO561.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

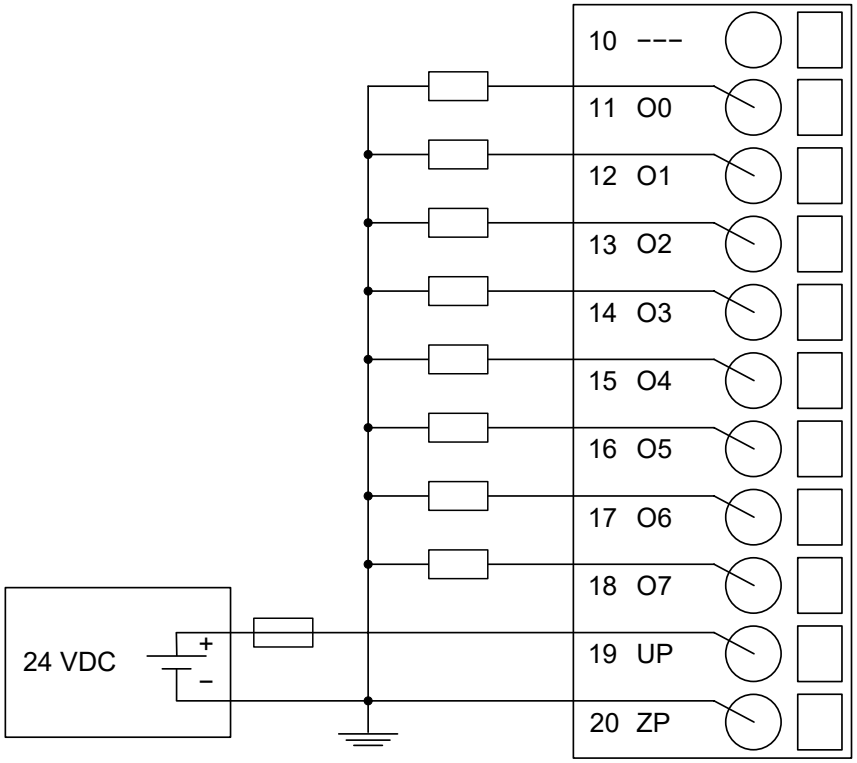
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the digital output module DO561:



NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external 3 A fast-protection fuse for the outputs.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.3.6.1.1.7.6 “Diagnosis” on page 2625*).

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.3.6.1.1.7.7 “State LEDs” on page 2626*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6120 [1])	WORD	6120 0x17E8	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xE9, 0x17, 0x00;

Diagnosis

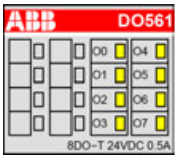
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error DO561								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error DO561								
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	5 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by an 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		1 (8 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Power dissipation within the module (max.)		1.6 W
Weight		Ca. 115 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 (8 channels per group)
Connection of the channels O0 to O7	Terminals 11 to 18
Common power supply voltage	Terminal 19 (positive pole of the process voltage, signal name UP)
Reference potential for the channels O0 to O7	Terminal 20 (negative pole of the process voltage, signal name ZP)
Indication of the output signals	1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus

Parameter		Value
Way of operation		Non-latching type
Min. output voltage at signal 1		20 V DC at max. current consumption
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	4 A
	Lamp load (max.)	5 W
Max. leakage current with signal 0		0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

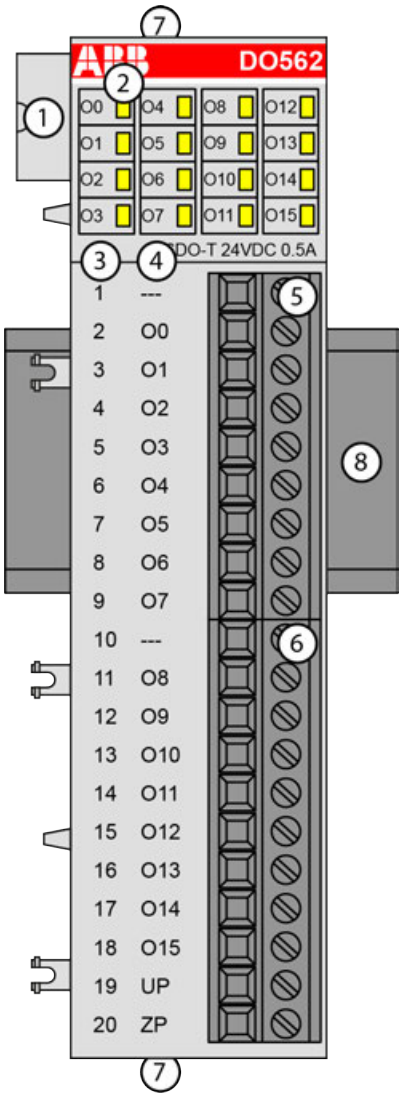
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2201	DO561, digital output module, 8 DO, transistor output	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DO562 - Digital output module

- 16 digital outputs 24 V DC (O0 to O15) in 1 group
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the outputs O0 to O15
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital outputs:

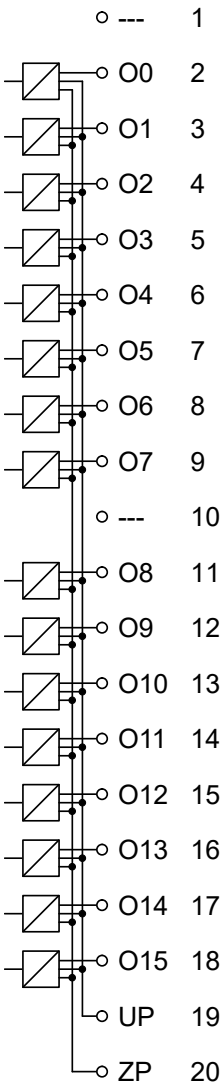


Table 473: Assignment of the terminals:

Terminal	Signal	Description
1	---	Reserved
2	O0	Output signal O0
3	O1	Output signal O1
4	O2	Output signal O2
5	O3	Output signal O3
6	O4	Output signal O4
7	O5	Output signal O5
8	O6	Output signal O6
9	O7	Output signal O7
10	---	Reserved
11	O8	Output signal O8
12	O9	Output signal O9
13	O10	Output signal O10
14	O11	Output signal O11

Terminal	Signal	Description
15	O12	Output signal O12
16	O13	Output signal O13
17	O14	Output signal O14
18	O15	Output signal O15
19	UP	Process voltage UP (24 V DC)
20	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DO562.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

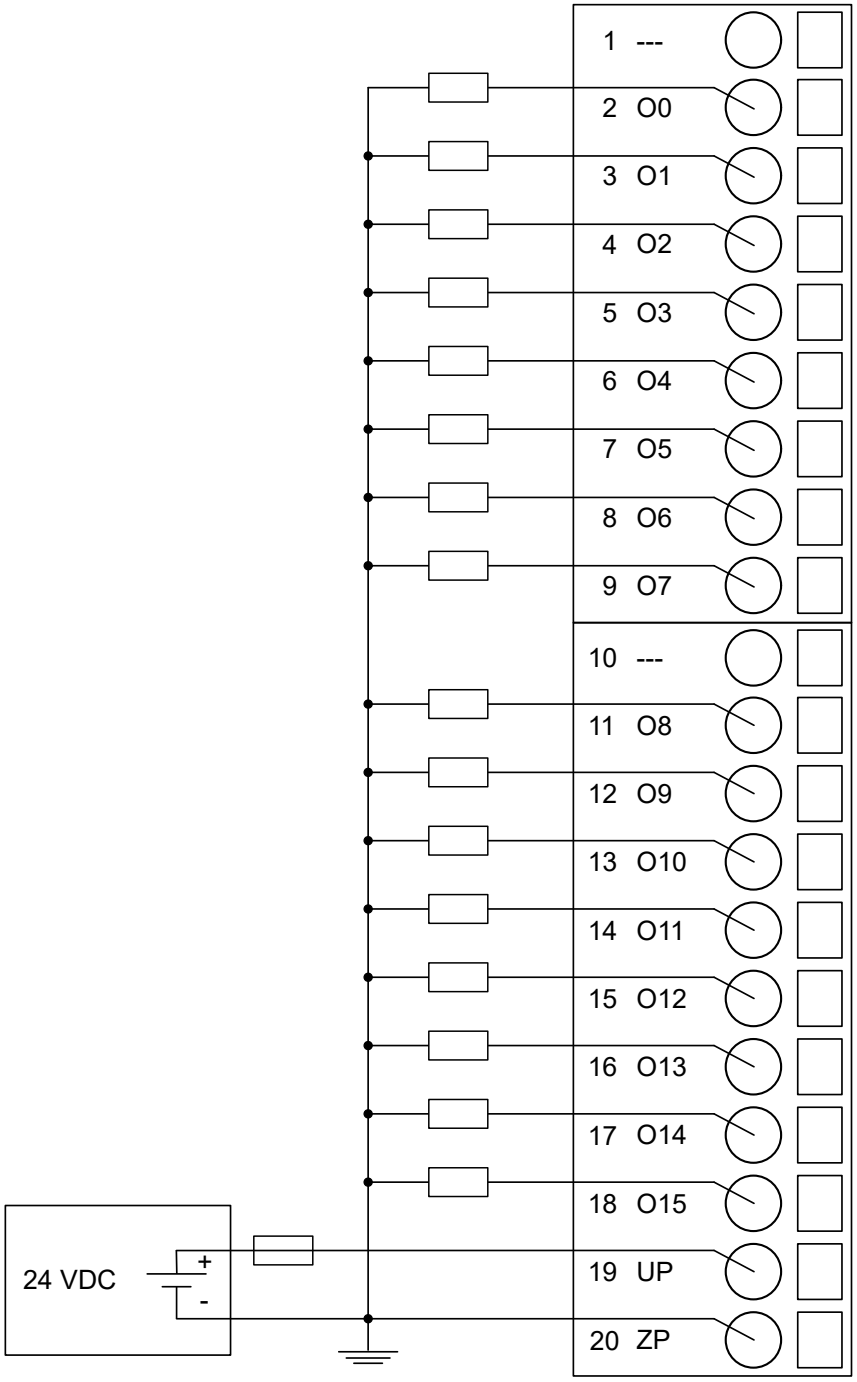
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the digital output module DO562:



NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

– Never short-circuit or overload the outputs.
– Never connect the outputs to other voltages.
– Use an external 3 A fast-protection fuse for the outputs.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.3.6.1.1.8.6 “Diagnosis” on page 2635*).

The meaning of the LEDs is described in the section Status LEDs *Chapter 1.6.3.6.1.1.8.7 “State LEDs” on page 2635*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6145 [1])	WORD	6145 0x1801	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x06
Ext_User_Prm_Data_Const(0) =	0x18, 0x02, 0x00, 0x02, 0x00, 0x00;

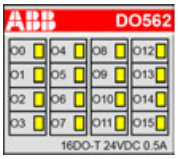
Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Inter- face**	**Device**	**Module**	**Channel**	**Error-Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = Module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O15	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	20 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by an 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		1 (16 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		1.4 W
Weight		Ca. 125 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter	Value
Number of channels per module	16 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 (16 channels per group)
Connection of the channels O0 to O7	Terminals 1 to 9
Connection of the channels O8 to O15	Terminals 11 to 18
Common power supply voltage	Terminal 19 (positive pole of the process voltage, signal name UP)
Reference potential for the channels O0 to O15	Terminal 20 (negative pole of the process voltage, signal name ZP)

Parameter		Value
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Way of operation		Non-latching type
Min. output voltage at signal 1		UP -0.3 V at max. current consumption
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		2 bytes
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	8 A
	Lamp load (max.)	5 W
Max. leakage current with signal 0		0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Ordering data

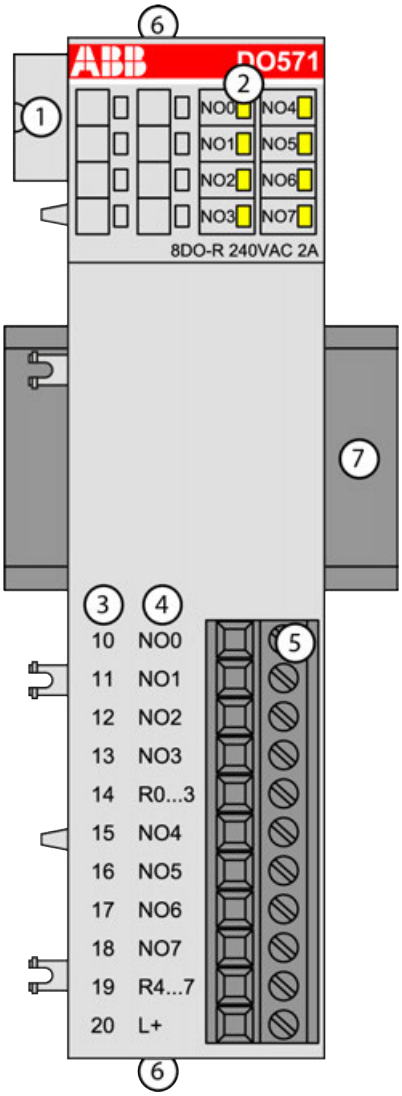
Part no.	Description	Product life cycle phase *)
1SAP 230 900 R0000	DO562, digital output module, 16 DO, transistor output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO571 - Digital output module

- 8 digital normally open relay outputs 24 V DC / 24 V AC or 100-240 V AC, 2 A max. (NO0 to NO7) in 2 groups
- Group-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the outputs O0 to O7
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminal L+ (process voltage 24 V DC). The negative pole is provided by the I/O bus.

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital outputs:

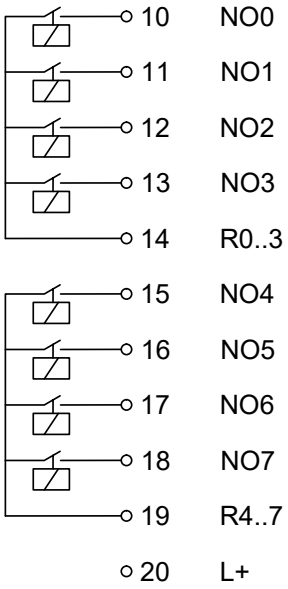


Table 474: Assignment of the terminals:

Terminal	Signal	Description
10	NO0	Normally-open contact of the output NO0
11	NO1	Normally-open contact of the output NO1
12	NO2	Normally-open contact of the output NO2
13	NO3	Normally-open contact of the output NO3
14	R0..3	Output common for signals NO0 to NO3
15	NO4	Normally-open contact of the output NO4

Terminal	Signal	Description
16	NO5	Normally-open contact of the output NO5
17	NO6	Normally-open contact of the output NO6
18	NO7	Normally-open contact of the output NO7
19	R4..7	Output common for signals NO4 to NO7
20	L+	Process voltage L+ +24 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per DO571.

The external power supply connection is carried out via the L+ (+24 V DC) terminal. The negative pole of the external power supply is realized via the I/O bus. Therefore, the CPU/ communication interface module and the DO571 must have a common power supply.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For screw-type terminals only:

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..3 and R4..7) does not exceed 8 A.

Never connect total currents > 8 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

The following figure shows the connection of the module:

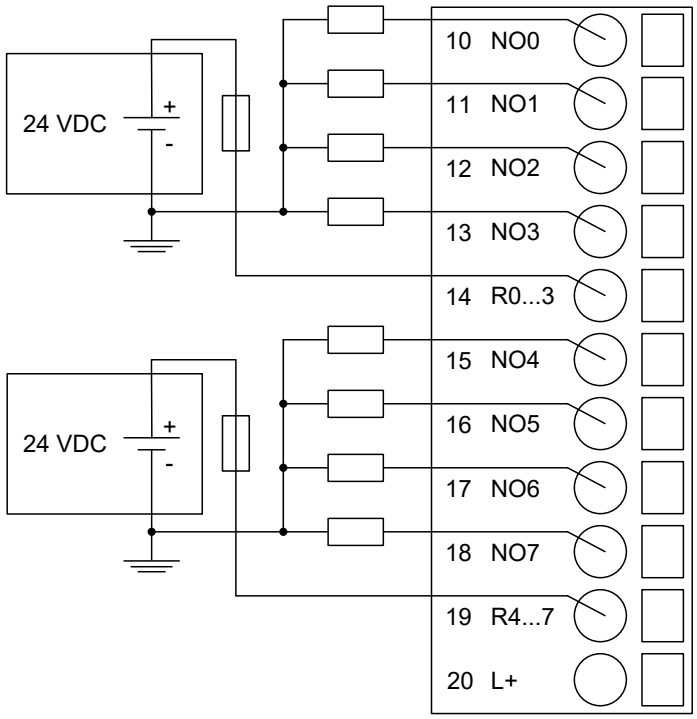


Fig. 105: Connection of 24 V DC actuators

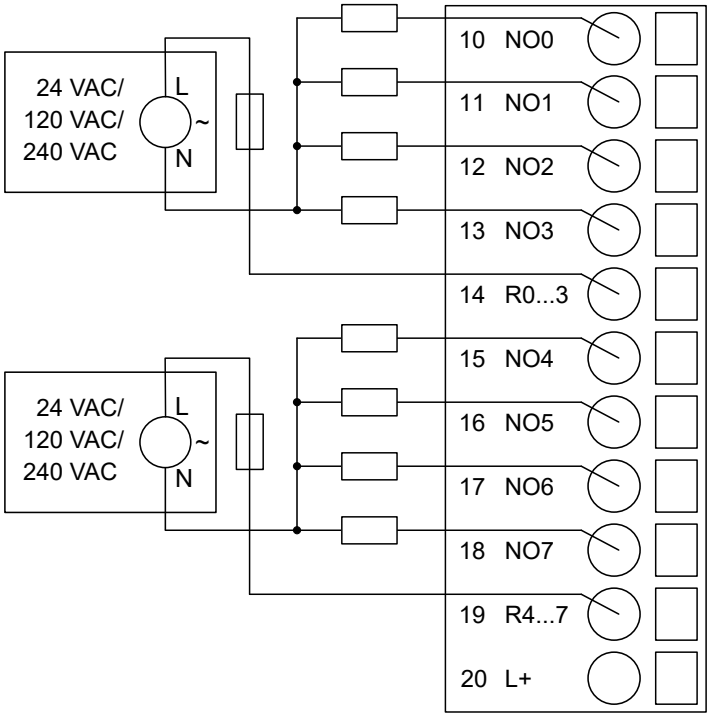


Fig. 106: Connection of 24 V AC or 100-240 V AC actuators

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be supplied from the same phase.
- Use an external 5 A fast protection fuse for the outputs.

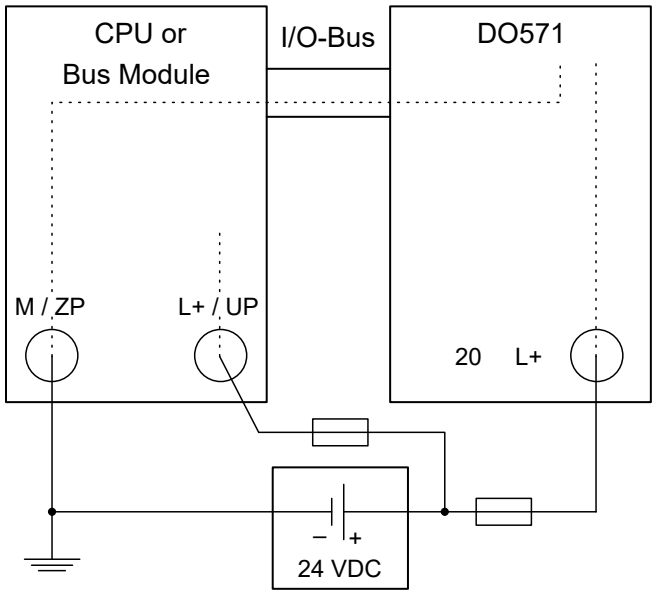


Fig. 107: Power supply - the negative connection is realized via the I/O bus

The L+ connection of the DO571 and the 24 V supply of the CPU/communication interface module must be connected to the same 24 V power supply.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.3.6.1.1.9.6 "Diagnosis" on page 2645*).

The meaning of the LEDs is described in the section Status LEDs *Chapter 1.6.3.6.1.1.9.7 "State LEDs" on page 2646*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6125 [1])	WORD	6125 0x17ED	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])
Check supply	Off On	0 1	BYTE	On 0x01			

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x04
Ext_User_Prm_Data_Const(0) =	0xEF, 0x17, 0x00,\ 0x01;

Diagnosis

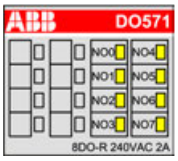
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = Hardware address (e. g. of the DC551-CS31)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1..10 = expansion 1..10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 20 for L+ (+24 V DC). The negative pole is provided by the I/O bus.
	Rated value	24 V DC
	Current consumption via L+	50 mA
	Inrush current (at power-up)	0.0035 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by a 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		2 (4 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.0 W
Weight		Ca. 150 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 normally-open relay outputs
Distribution of the channels into groups		2 (4 channels per group)
Connection of the channels O0 to O3		Terminals 10 to 13
Connection of the channels O4 to O7		Terminals 15 to 18
Reference potential for the channels O0 to O3		Terminal 14 (signal name R0..3)
Reference potential for the channels O4 to O7		Terminal 19 (signal name R4..7)
Relay coil power supply		Terminal 20 (positive pole of the process supply voltage, signal name L+). The negative pole is provided by the I/O bus.
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Way of operation		Non-latching type
Relay output voltage		
	Rated value	24 V DC / 24 V AC or 120/240 V AC
Output delay		
	Switching 0 to 1 (max.)	Typ. 10 ms
	Switching 1 to 0 (max.)	Typ. 10 ms
Output data length		1 byte
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	8 A
	Lamp load (max.)	200 W (230 V AC), 30 W (24 V DC)
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	On Request
	With lamp loads	Max. 1 Hz
Output type		Non-protected
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No

Parameter		Value
	Output current limitation	No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

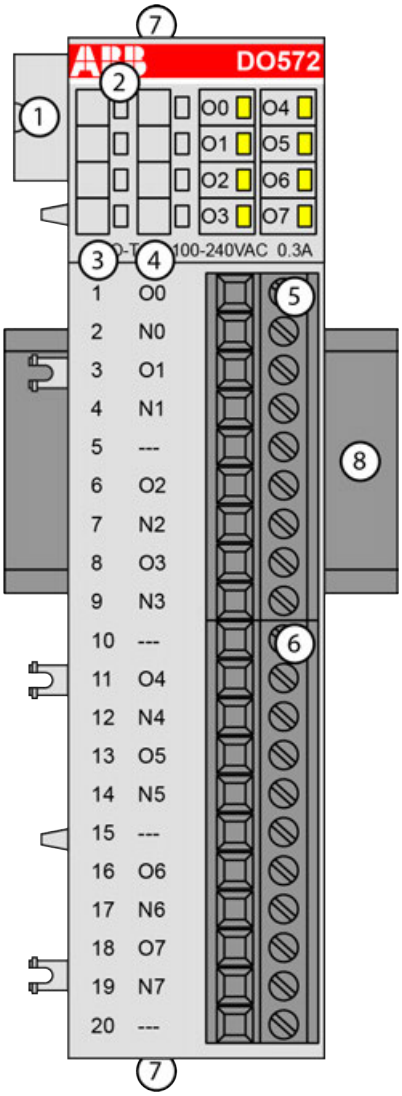
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2202	DO571, digital output module, 8 DO, relay output	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO572 - Digital output module

- 8 digital triac outputs (O0 to O7) in 8 groups
- 240 V AC
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the outputs O0 to O7
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital outputs:

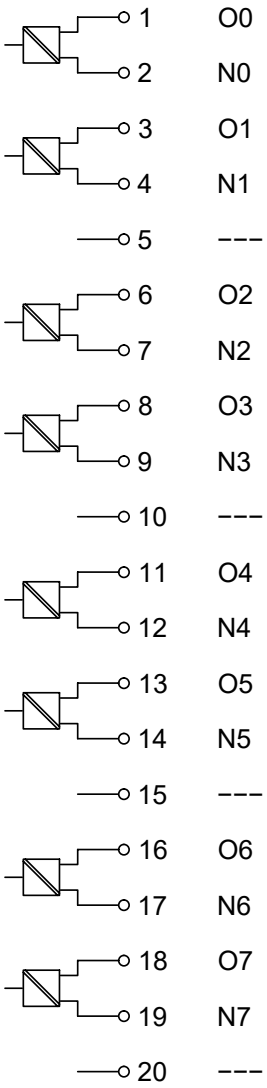


Table 475: Assignment of the terminals:

Terminal	Signal	Description
1	O0	Output signal O0
2	N0	Neutral conductor for the output signal O0
3	O1	Output signal O1
4	N1	Neutral conductor for the output signal O1
5	---	Reserved
6	O2	Output signal O2
7	N2	Neutral conductor for the output signal O2
8	O3	Output signal O3
9	N3	Neutral conductor for the output signal O3
10	---	Reserved
11	O4	Output signal O4

Terminal	Signal	Description
12	N4	Neutral conductor for the output signal O4
13	O5	Output signal O5
14	N5	Neutral conductor for the output signal O5
15	---	Reserved
16	O6	Output signal O6
17	N6	Neutral conductor for the output signal O6
18	O7	Output signal O7
19	N7	Neutral conductor for the output signal O7
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DO572.

An external power supply connection is not needed.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

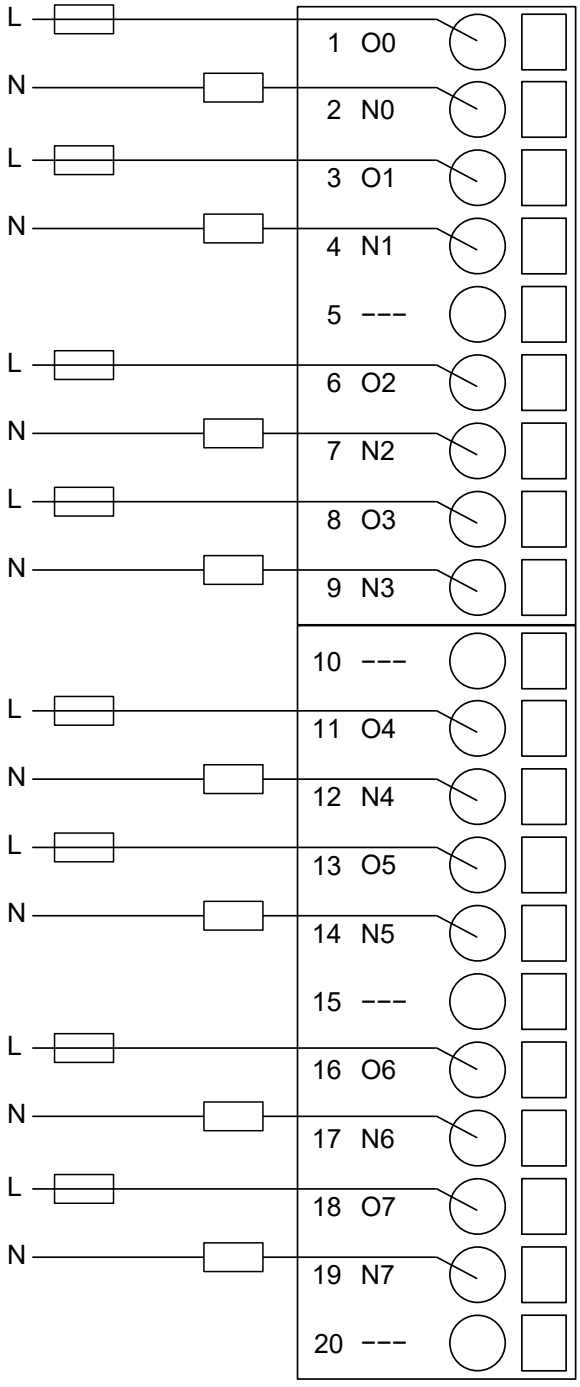
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the module:



NOTICE!

Risk of damaging the PLC modules!

The PLC modules will be irreparably damaged if a voltage > 240 V is connected.

Make sure that all inputs are fed from the same phase. The module must not be connected to a 400 V voltage.

The module provides several diagnosis functions (see chapter Diagnosis *Chapter 1.6.3.6.1.1.10.6 “Diagnosis” on page 2655*).

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.3.6.1.1.10.7 “State LEDs” on page 2656*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6130 [1])	WORD	6130 0x17F2	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1])	With CS31 and addresses smaller than 70, the value is increased by 1
[2])	The module has no additional user-configurable parameters
[3])	Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xF3, 0x17, 0x00;

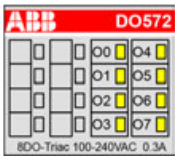
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the channels and the rest of the module
Isolated groups	8 (1 channel per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	On Request
Weight	ca. 120 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 triac outputs
Distribution of the channels into groups		8 groups (1 channel per group)
Connection of the channels O0 to O7		Terminals 1, 3, 5, 7, 10, 12, 14, 16
Reference potential for the channels O0 to O7		Terminals 2, 4, 6, 8, 11, 13, 15, 17
Output voltage for signal 1		On Request
Max. leakage current with signal 0		1.1 mA root mean square at 132 V AC and 1.8 mA root mean square at 264 V AC
Output voltage		
	Rated value	120 V AC or 240 V AC
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus

Parameter		Value
Way of operation		Non-latching type
Output delay		On Request
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.3 A
	Rated current per group (max.)	0.3 A
Surge current (max.)		On request
Lamp load (max.)		On request
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 10 Hz
	With inductive loads	Not applicable
	With lamp loads	Max. 10 Hz
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse		2 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No
	Output current limitation	No
Resistance to feedback against 230 V AC		No
Connection of 2 outputs in parallel		Not applicable
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Ordering data

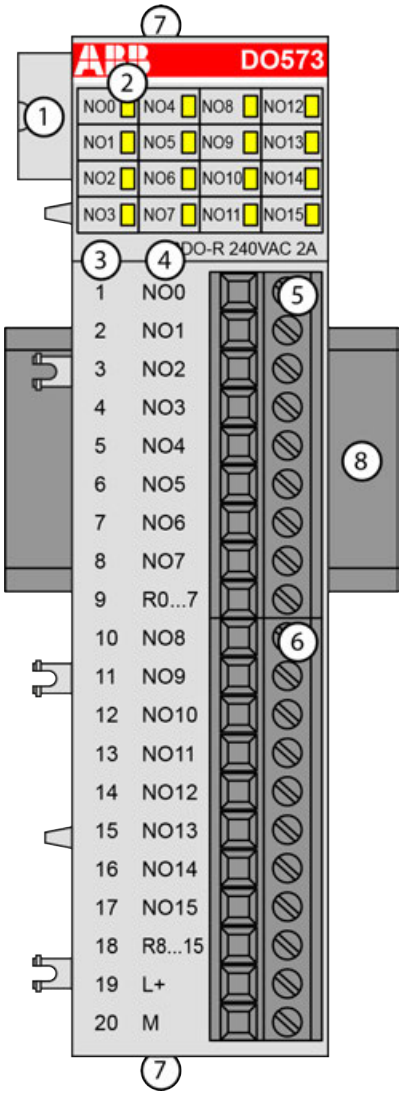
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2203	DO572, digital output module, 8 DO, triac output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO573 - Digital output module

- 16 digital normally open relay outputs 24 V DC or 100-240 V AC (NO0 to NO15) in 2 groups, 2 A max.
- Group-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the outputs O0 to O15
3 Terminal number

4 Allocation of signal name
5 Terminal block for output signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital outputs:

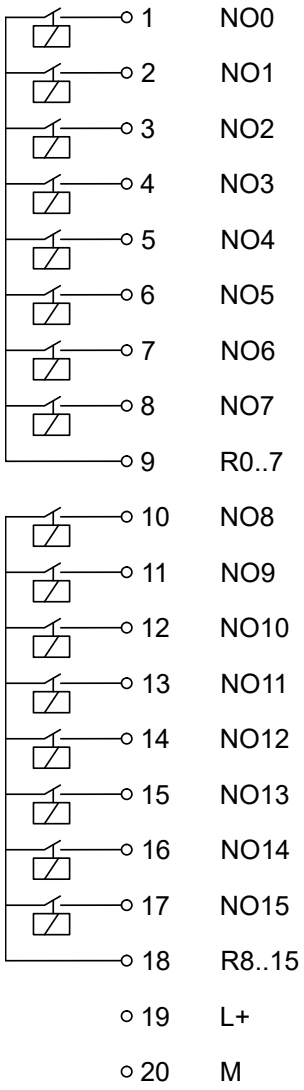


Table 476: Assignment of the terminals:

Terminal	Signal	Description
1	NO0	Normally-open contact of the output NO0
2	NO1	Normally-open contact of the output NO1
3	NO2	Normally-open contact of the output NO2
4	NO3	Normally-open contact of the output NO3
5	NO4	Normally-open contact of the output NO4
6	NO5	Normally-open contact of the output NO5
7	NO6	Normally-open contact of the output NO6
8	NO7	Normally-open contact of the output NO7
9	R0..7	Output common for signals NO0 to NO7
10	NO8	Normally-open contact of the output NO8
11	NO9	Normally-open contact of the output NO9
12	NO10	Normally-open contact of the output NO10
13	NO11	Normally-open contact of the output NO11
14	NO12	Normally-open contact of the output NO12

Terminal	Signal	Description
15	NO13	Normally-open contact of the output NO13
16	NO14	Normally-open contact of the output NO14
17	NO15	Normally-open contact of the output NO15
18	R8..15	Output common for signals NO8 to NO15
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per DO573.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is electrically interconnected to the M/ZP terminal of the CPU/ communication interface module.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For screw-type terminals only:

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be supplied from the same phase.
- Use an external 5 A fast protection fuse for the outputs.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..7 and R8..15) does not exceed 10 A.

Never connect total currents > 10 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

The following figure shows the connection of the module:

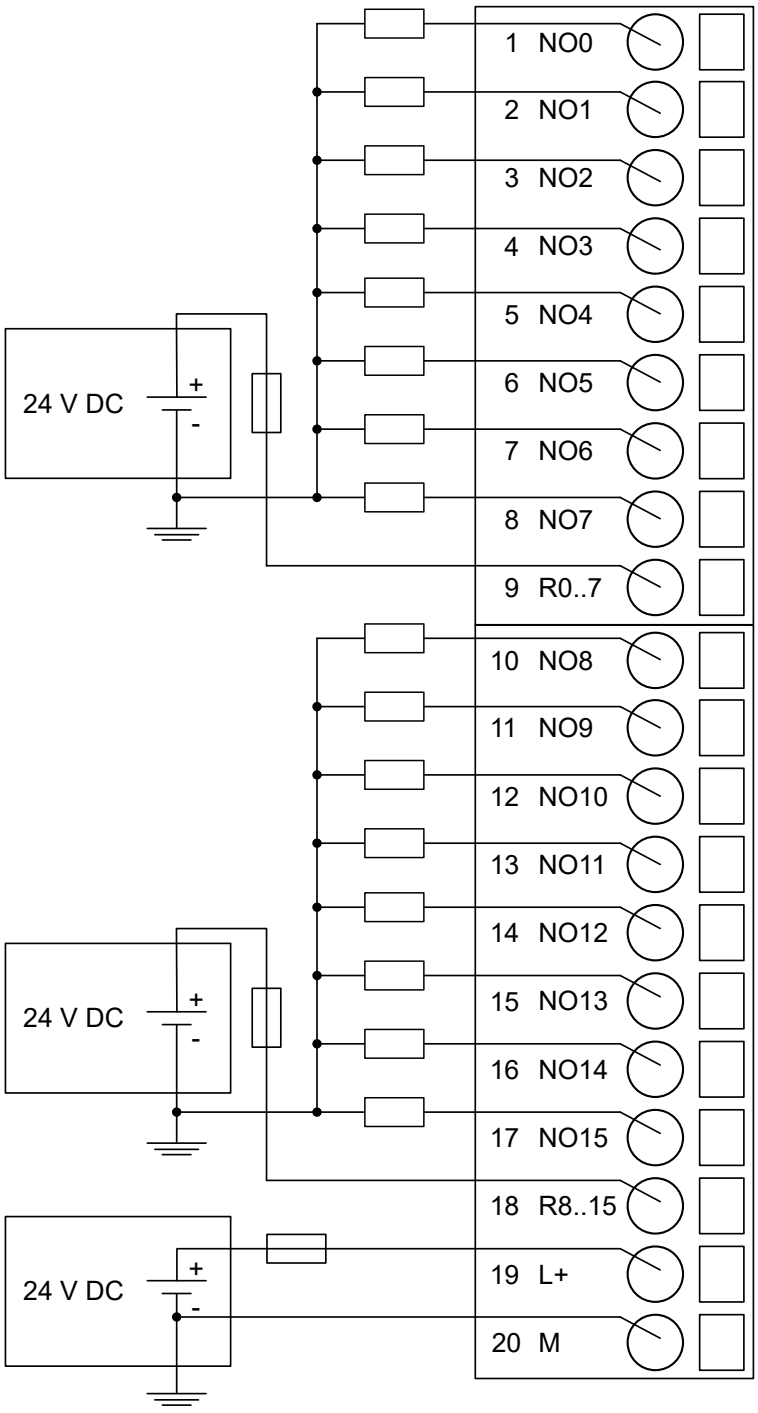


Fig. 108: Connection of 24 V DC actuators

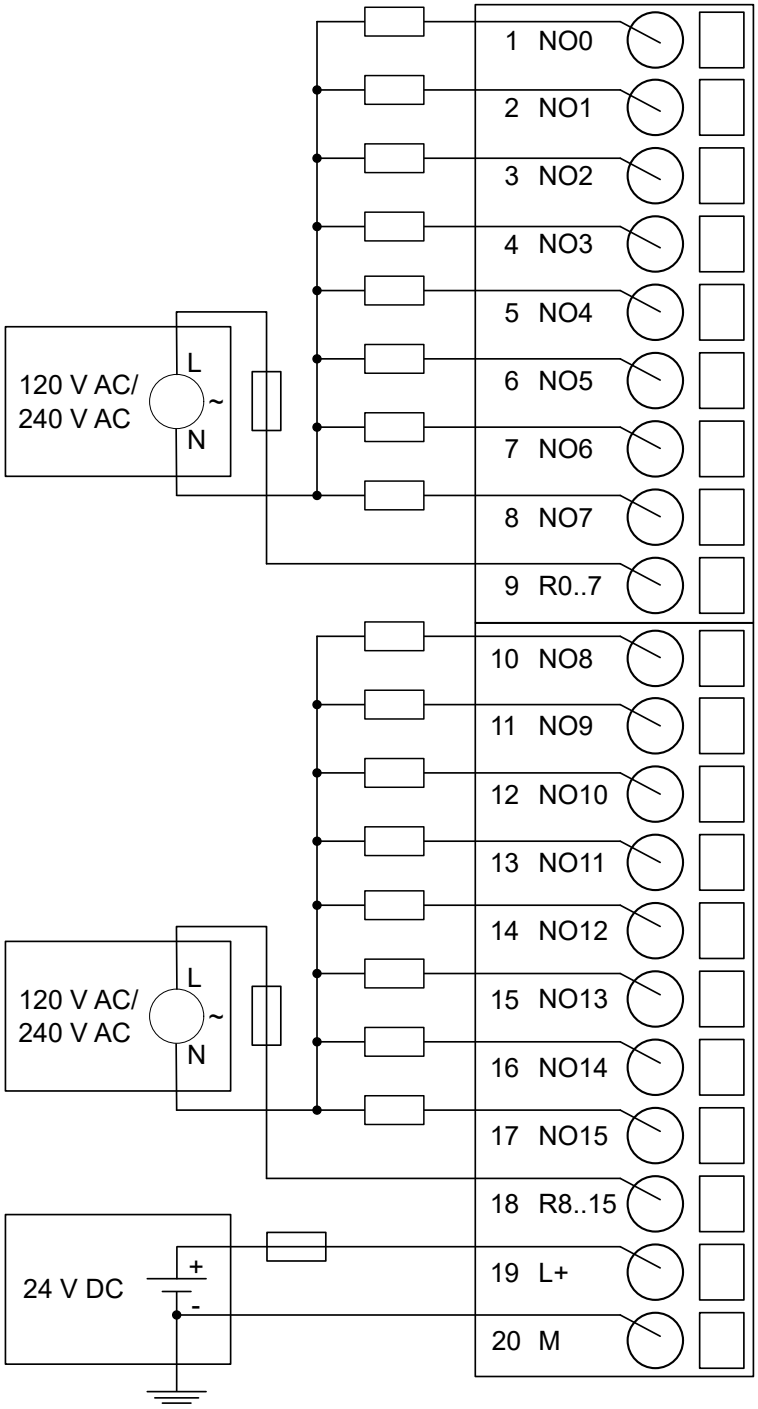


Fig. 109: Connection of 100-240 V AC actuators

The module provides several diagnosis functions (see section Diagnosis ↳ *Chapter 1.6.3.6.1.1.11.6 “Diagnosis” on page 2666*).

The meaning of the LEDs is described in the section State LEDs ↳ *Chapter 1.6.3.6.1.1.10.7 “State LEDs” on page 2656*.

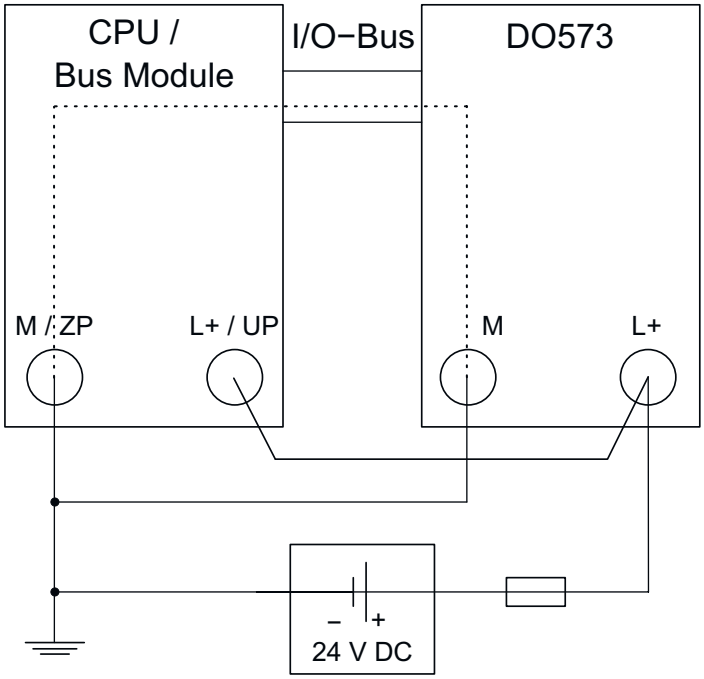


Fig. 110: Power supply - the negative connection is realized via the I/O bus

The L+ connection of the DO573 and the 24 V supply of the CPU/communication interface module must be connected to the same 24 V power supply .

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6150 [1])	WORD	6150 0x1806	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])
Check supply	Off On	0 1	BYTE	On 0x01			

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	0x07 0x18, 0x07, 0x00, 0x03, 0x01, 0x00, 0x00;

Diagnosis

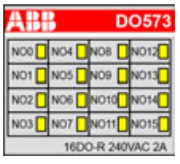
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...10 = decentralized communication interface module 1...10, ADR = Hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Outputs NO0...NO15	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminals 19 for L+ (+24 V DC) and 20 for M (0 V DC)
	Rated value	24 V DC
	Current consumption via L+	50 mA
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for L+	Recommended; the outputs must be protected by an 5 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the output groups and the rest of the module
Isolated groups		2 (8 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.0 W
Weight		Ca. 160 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter	Value
Number of channels per module	16 normally-open relay outputs
Distribution of the channels into groups	2 (8 channels per group)
Connection of the channels NO0 to NO7	Terminals 1 to 8
Connection of the channels NO8 to NO15	Terminals 10 to 17
Reference potential for the channels NO0 to NO7	Terminal 9 (signal name R0..7)
Reference potential for the channels NO8 to NO15	Terminal 18 (signal name R8..15)
Relay coil power supply	Terminals 19 and 20 (signal names L+ and M)
Indication of the output signals	1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Way of operation	Non-latching type

Parameter		Value
Relay output voltage		
	Rated value	24 V DC or 120/240 V AC
Output delay		
	Switching 0 to 1 (max.)	Typ. 10 ms
	Switching 1 to 0 (max.)	Typ. 10 ms
Output data length		2 bytes
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	10 A
Lamp load (max.)		200 W (230 V AC), 30 W (24 V DC)
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	On Request
	With lamp loads	Max. 1 Hz
Output type		Non-protected
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No
	Output current limitation	No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

Ordering data

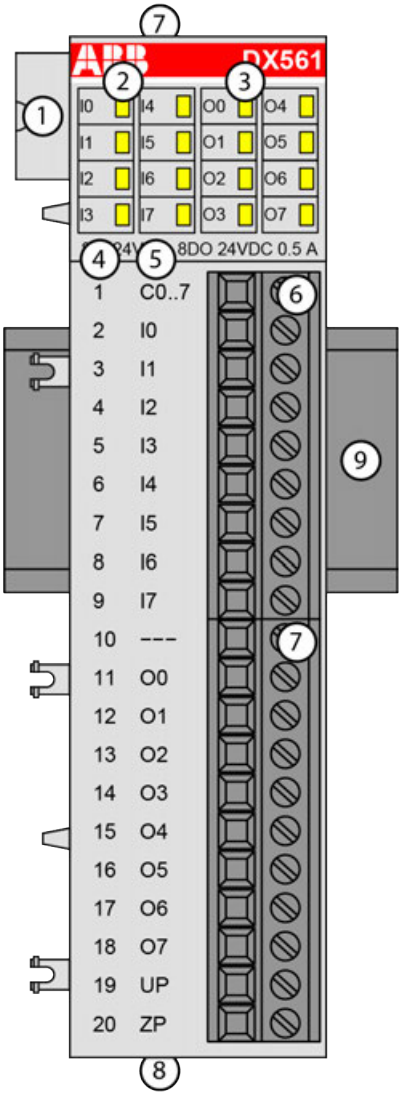
Part no.	Description	Product life cycle phase *)
1SAP 231 300 R0000	DO573, digital output module, 16 DO, relay output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DX561 - Digital input/output module

- 8 digital inputs 24 V DC (I0 to I7) in 1 group
- 8 digital transistor outputs 24 V DC (O0 to O7) in 1 group
- Group-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 8 yellow LEDs to display the signal states of the outputs O0 to O7
4 Terminal number
5 Allocation of signal name
6 Terminal block for input signals (9-pin)
7 Terminal block for output signals (11-pin)
8 2 holes for wall-mounting with screws
9 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs and outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs and outputs:

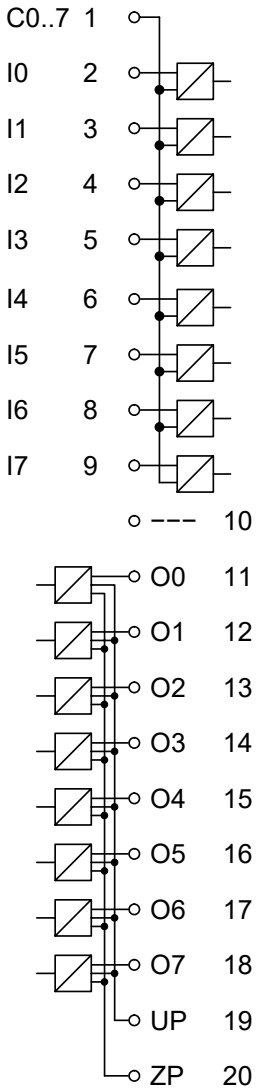


Table 477: Assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7
10	---	Reserved
11	O0	Output signal O0
12	O1	Output signal O1
13	O2	Output signal O2

Terminal	Signal	Description
14	O3	Output signal O3
15	O4	Output signal O4
16	O5	Output signal O5
17	O6	Output signal O6
18	O7	Output signal O7
19	UP	Process voltage UP +24 V DC
20	ZP	Process voltage ZP 0 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DX561.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The digital inputs can be used as source inputs or as sink inputs.

NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the inputs to the digital input/output module DX561:

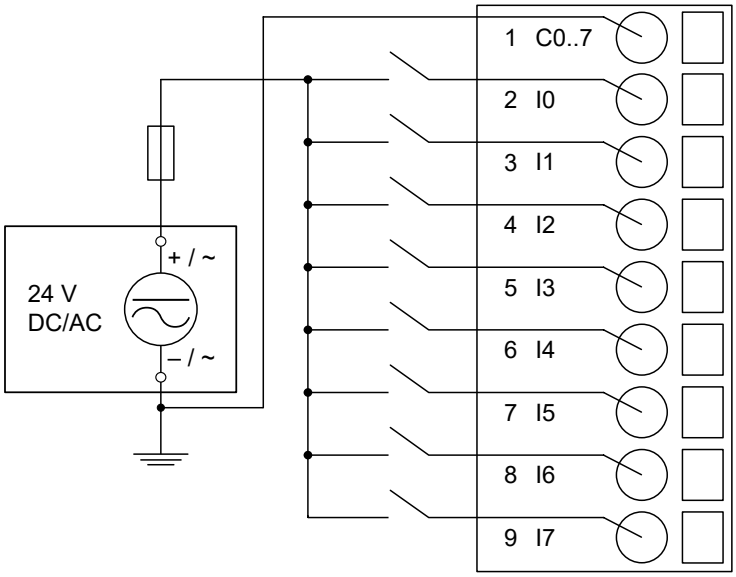


Fig. 111: Connection of inputs - sink inputs

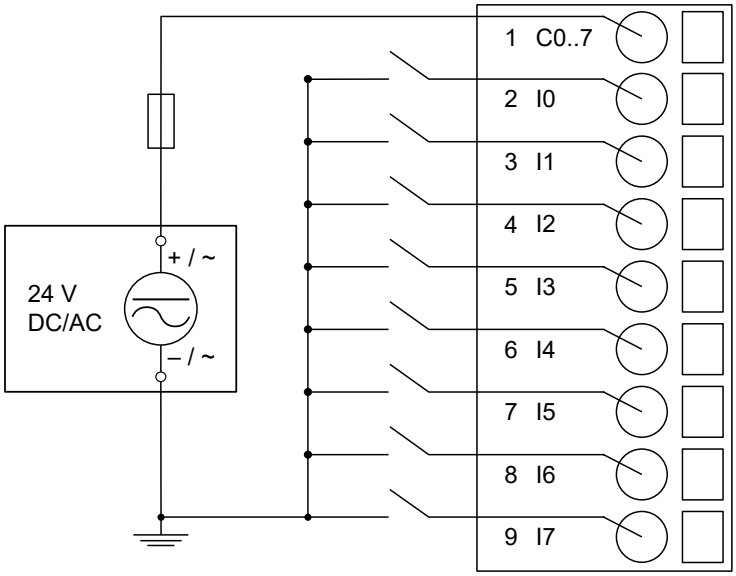


Fig. 112: Connection of inputs - source inputs

The following figure shows the connection of the outputs to the module:

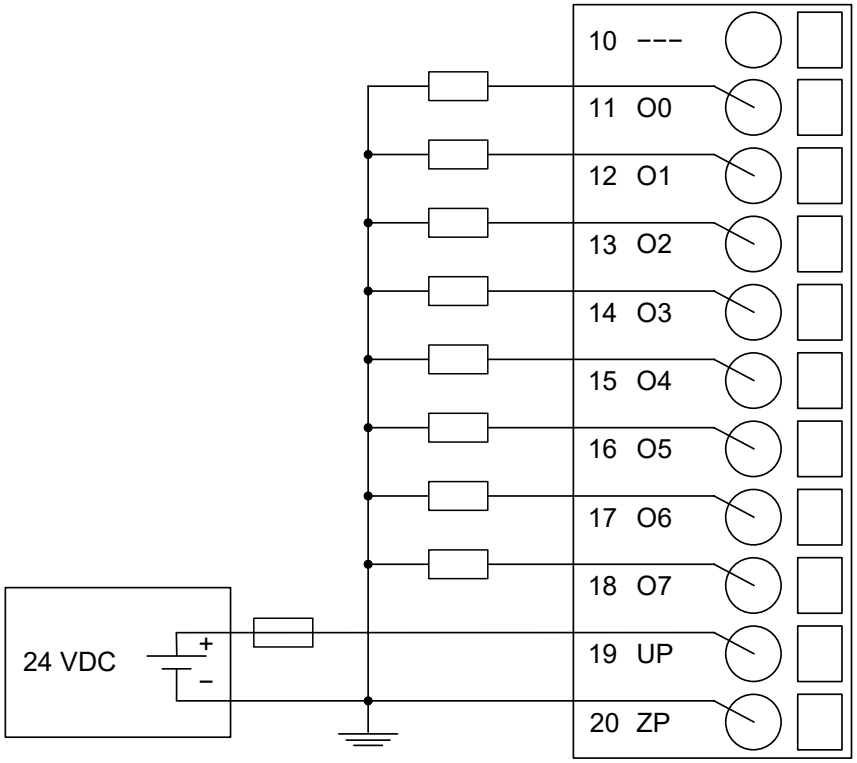


Fig. 113

NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external 3 A fast-protection fuse for the outputs.

The module provides several diagnosis functions (see chapter Diagnosis ↬ *Chapter 1.6.3.6.1.1.12.6 “Diagnosis” on page 2678*).

The meaning of the LEDs is described in the Displays section ↬ *Chapter 1.6.3.6.1.1.12.7 “State LEDs” on page 2679* chapter.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6135 [1])	WORD	6135 0x17F7	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xF8, 0x17, 0x00,\
(0) =	0x01;

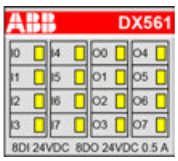
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	5 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by an 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the input group and the output group and the rest of the module
Isolated groups		2 groups (1 group for 8 input channels, 1 group for 8 output channels)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.3 W
Weight		ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter		Value	
Number of channels per module		8	
Distribution of the channels into groups		1 group for 8 channels	
Connections of the channels I0 to I7		Terminals 2 to 9	
Reference potential for the channels I0 to I7		Terminal 1	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)	
Monitoring point of input indicator		LED is part of the input circuitry	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
Signal 0		-5 V...+3 V	-3 V...+5 V
Undefined signal		-15 V...+ 5 V	+5 V...+15 V
Signal 1		-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		-5 V...+3 V	-3 V...+5 V
Ripple with signal 1		-30 V...-15 V	+15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 5 mA	
	Input voltage +5 V	Typ. 1 mA	
	Input voltage +15 V	> 2.5 mA	
	Input voltage +30 V	< 8 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 8 ms	
Input data length		1 byte	
Max. cable length			
	Shielded	500 m	
	Unshielded	300 m	

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 group of 8 channels
Connection of the channels O0 to O7	Terminals 11 to 18
Reference potential for the channels O0 to O7	Terminal 20 (negative pole of the process voltage, name ZP)
Common power supply voltage	Terminal 19 (positive pole of the process voltage, name UP)
Indication of the output signals	1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Monitoring point of output indicator	Controlled together with transistor

Parameter		Value
Way of operation		Non-latching type
Max. output voltage at signal 1		20 V DC at max. current consumption
Output delay		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	4 A
	Rated current (all channels together, max.)	4 A
	Lamp load (max.)	5 W
	Max. leakage current with signal 0	0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Ordering data

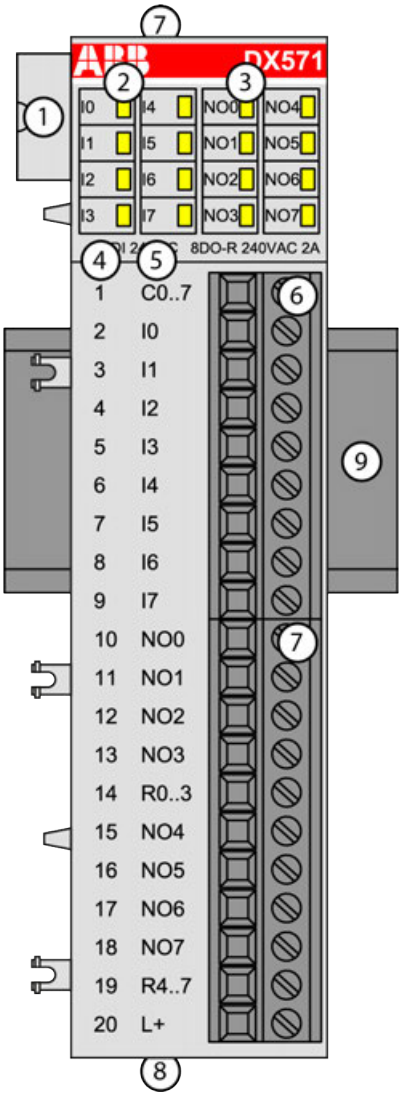
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2301	DX561, digital input/output module, 8 DI 24 V DC, 8 DO 24 V DC, transistor output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DX571 - Digital input/output module

- 8 digital inputs 24 V DC / 24 V AC (I0 to I7) in 1 group
- 8 digital normally open relay outputs 24 V DC / 24 V AC or 100-240 V AC, 2 A max. (NO0 to NO7) in 2 groups
- Group-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 8 yellow LEDs to display the signal states of the outputs NO0 to NO7
4 Terminal number
5 Allocation of signal name
6 Terminal block for input signals (9-pin)
7 Terminal block for output signals (11-pin)
8 2 holes for wall-mounting with screws
9 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs and outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminal L+ (process voltage 24 V DC). The negative pole is provided by the I/O bus.

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↪ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs and outputs:

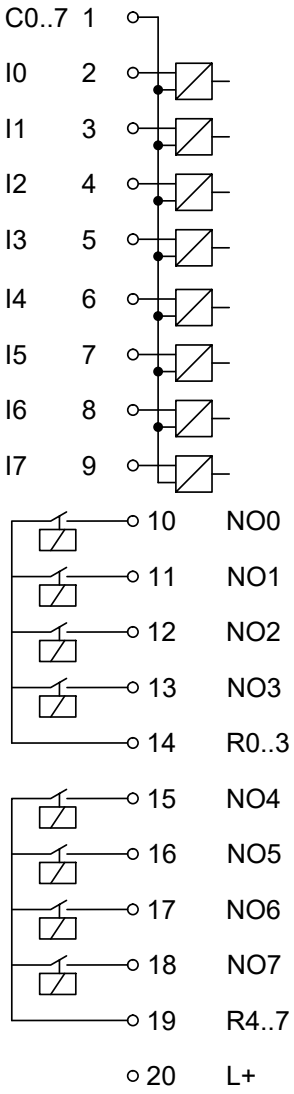


Table 478: Assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7
10	NO0	Normally-open contact of the output 0
11	NO1	Normally-open contact of the output 1
12	NO2	Normally-open contact of the output 2

Terminal	Signal	Description
13	NO3	Normally-open contact of the output 3
14	R0...3	Output common for signals O0 to O3
15	NO4	Normally-open contact of the output 4
16	NO5	Normally-open contact of the output 5
17	NO6	Normally-open contact of the output 6
18	NO7	Normally-open contact of the output 7
19	R4...7	Output common for signals O4 to O7
20	L+	Process voltage +24 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per DX571.

The external power supply connection is carried out via the L+ (+24 V DC) terminal. The negative pole of the external power supply is realized via the I/O bus. Therefore, the CPU/ communication interface module and the DX571 must have a common power supply.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..3 and R4..7) does not exceed 8 A.

Never connect total currents > 8 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.3.6.1.1.13.6 “Diagnosis” on page 2691*).

The digital inputs can be used as source inputs or as sink inputs.

NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figures show the connection of the inputs to the digital input/output module DX571:

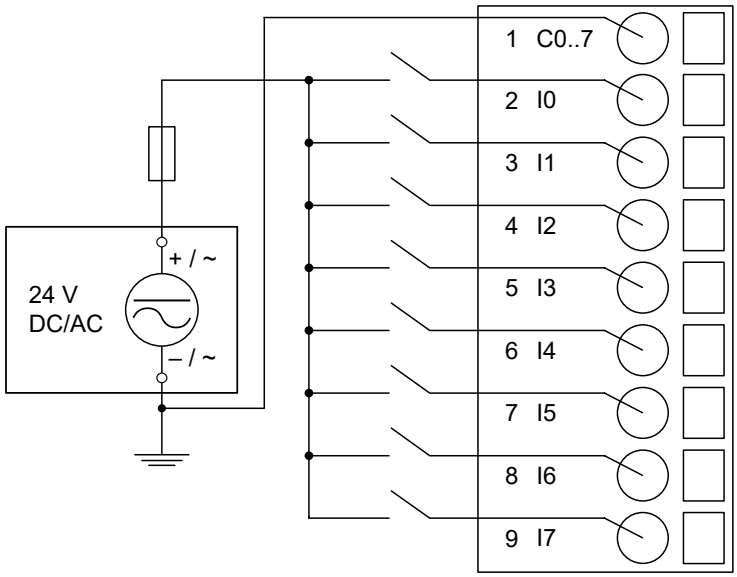


Fig. 114: Connection of inputs - sink inputs

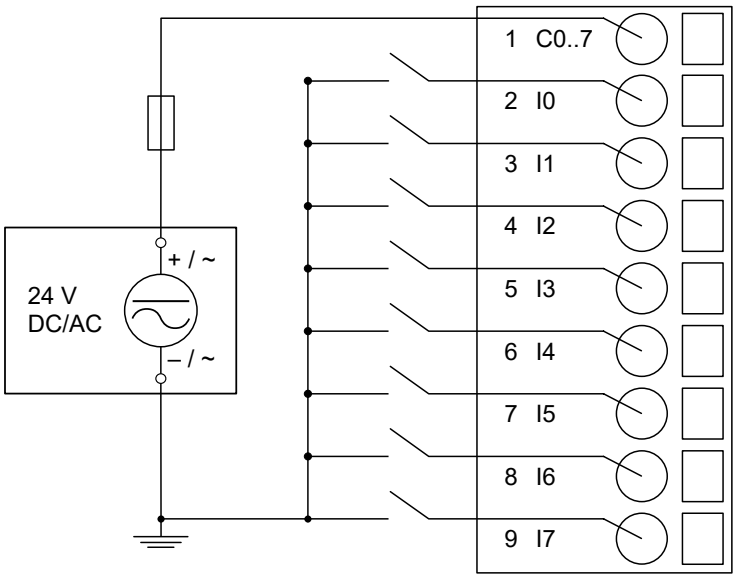


Fig. 115: Connection of inputs - source inputs

The following figures show the connection of the outputs to the module:

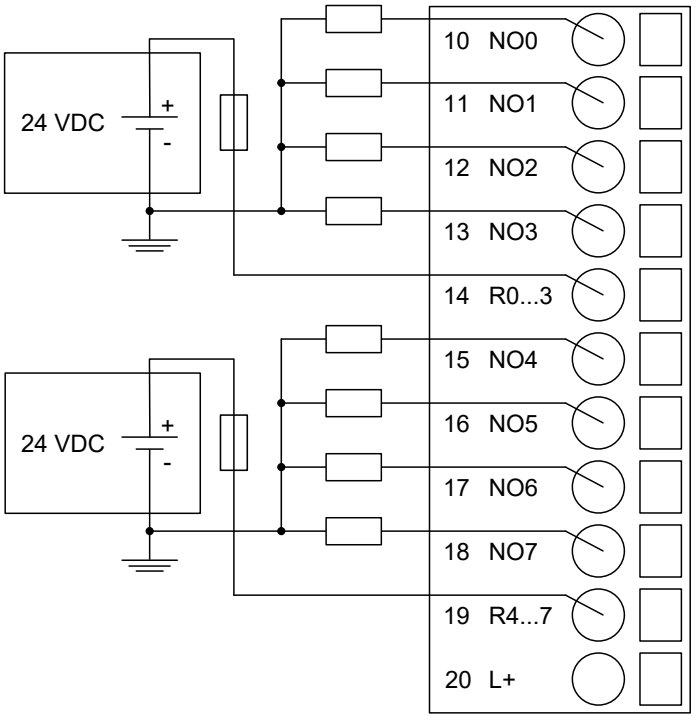


Fig. 116: Connection of 24 V DC actuators

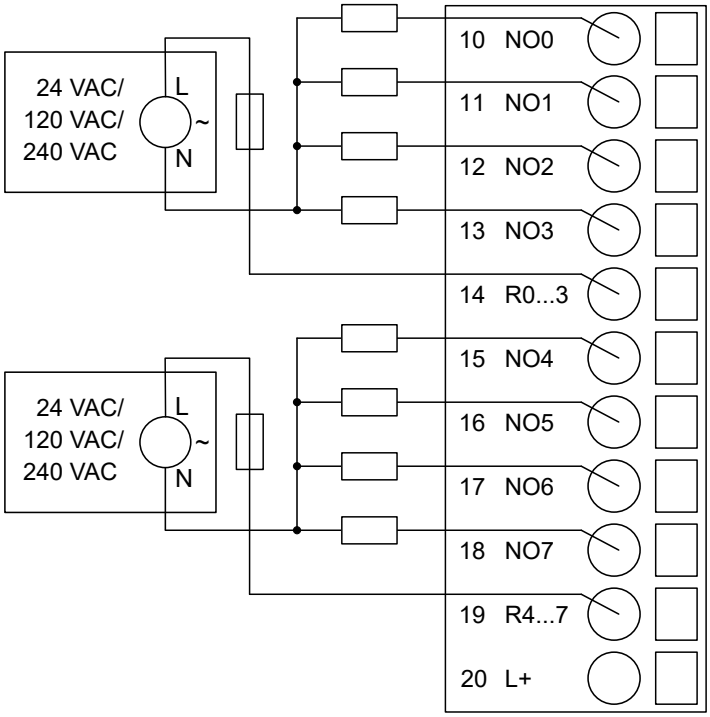


Fig. 117: Connection of 24 V AC or 100-240 V AC actuators

The L+ connection of the DX571 and the 24 V supply of the CPU/communication interface module must be connected to the same 24 V power supply.

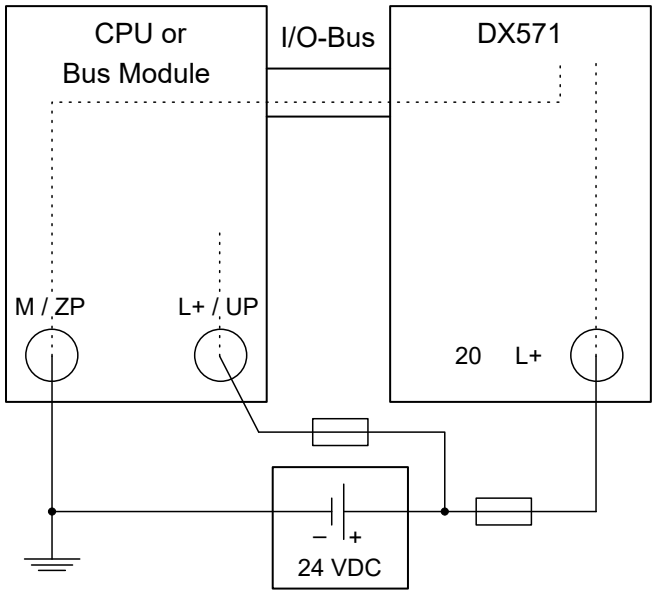


Fig. 118: Power supply - the minus connection is realized via the I/O bus

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For screw-type terminals only:

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be supplied from the same phase.
- Use an external 5 A fast protection fuse for the outputs.

The meaning of the LEDs is described in the Displays section *Chapter 1.6.3.6.1.1.13.7 “State LEDs” on page 2692*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6140 [1])	WORD	6140 0x17FC	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Check supply	Off On	0 1	BYTE	On 0x01			
[1]) with CS31 and addresses smaller than 70, the value is increased by 1							
[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)							

GSD file:

Ext_User_Prm_Data_Len =	0x04
Ext_User_Prm_Data_Const(0) =	0xFD, 0x17, 0x00,\
(0) =	0x01;

Diagnosis

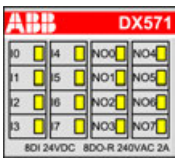
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<− Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block	

Class	Inter face	Device	Module	Channel	Error Identifier	Error message	Remedy
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
4	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = Module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = Module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON
	Outputs NO0...NO7	Digital output	Yellow	Output is OFF	Output is ON

In the undefined signal range, the state LED for the inputs can be ON although the input state detected by the module is OFF.

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 20 for L+ (+24 V DC). The negative pole is provided by the I/O bus.
	Rated value	24 V DC
	Current consumption via L+	50 mA
	Inrush current (at power-up)	0.0035 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for L+	Recommended; the outputs must be protected by a 3 A fast-acting fuse

Parameter	Value
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 5 mA
Galvanic isolation	Yes, between the input group and the output group and the rest of the module
Isolated groups	3 groups (1 group for 8 input channels, 2 groups for 8 output channels)
Surge-voltage (max.)	35 V DC for 0.5 s
Max. power dissipation within the module	2.3 W
Weight	Ca. 150 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter		Value		
Number of channels per module		8		
Distribution of the channels into groups		1 group for 8 channels		
Connections of the channels I0 to I7		Terminals 2 to 9		
Reference potential for the channels I0 to I7		Terminal 1		
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)		
Monitoring point of input indicator		LED is part of the input circuitry		
Input type according to EN 61131-2		Type 1 source	Type 1 sink	Type 1 AC [1])
Input signal range		-24 V DC	+24 V DC	24 V AC 50/60 Hz
Signal 0		-5 V...+3 V	-3 V...+5 V	0 V AC...5 V AC
Undefined signal		-15 V...+ 5 V	+5 V...+15 V	5 V AC...14 V AC
Signal 1		-30 V...-15 V	+15 V...+30 V	14 V AC...27 V AC
Input current per channel				
	Input voltage 24 V	Typ. 5 mA		Typ. 5 mA r.m.s.
	Input voltage 5 V	Typ. 1 mA		Typ. 1 mA r.m.s.
	Input voltage 14 V			Typ. 2.7 mA r.m.s.
	Input voltage 15 V	> 2.5 mA		
	Input voltage 27 V			Typ. 5.5 mA r.m.s.
	Input voltage 30 V	< 8 mA		
Max. permissible leakage current (at 2-wire proximity switches)		1 mA		Typ. 1 mA r.m.s.

Parameter		Value
Input delay (0->1 or 1->0)		Typ. 8 ms
Input data length		1 byte
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

[1]) When inputs are used with 24 V AC, external surge limiting filters are required.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 normally-open relay outputs
Distribution of the channels into groups		2 (4 channels per group)
Connection of the channels O0 to O3		Terminals 10 to 13
Connection of the channels O4 to O7		Terminals 15 to 18
Reference potential for the channels O0 to O3		Terminal 14 (signal name R0..3)
Reference potential for the channels O4 to O7		Terminal 19 (signal name R4..7)
Relay coil power supply		Terminal 20 (positive pole of the process supply voltage, signal name L+). The negative pole is provided by the I/O bus.
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered through the I/O bus
Monitoring point of output indicator		Controlled together with relay
Way of operation		Non-latching type
Relay output voltage		
	Rated value	24 V DC / 24 V AC or 120/240 V AC
Output delay		
	Switching 0 to 1 (max.)	Typ. 10 ms
	Switching 1 to 0 (max.)	Typ. 10 ms
Output data length		1 byte
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	8 A
Lamp load (max.)		200 W (230 V AC), 30 W (24 V DC)
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 1 Hz

Parameter		Value
	With inductive loads	On Request
	With lamp loads	Max. 1 Hz
Output type		Non-protected
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No
	Output current limitation	No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

Ordering data

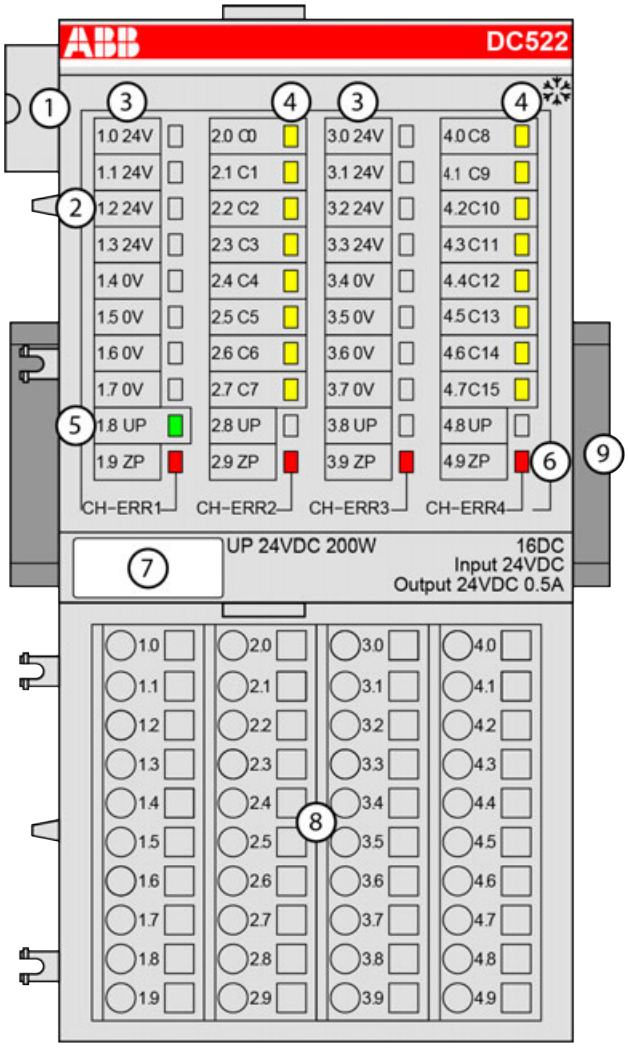
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2302	DX571, digital input/output module, 8 DI 24 V DC / 24 V AC, 8 DO, relay output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

S500

DC522 - Digital input/output module

- 16 configurable digital inputs/outputs
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 Sensor power supply 24 V DC / 0.5 A
4 16 yellow LEDs to display the signal states at the digital inputs/outputs (C0 - C15)
5 1 green LED to display the state of the process supply voltage UP
6 4 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input/output unit.

- 2 sensor supply voltages 24 V DC, 0.5 A, with short-circuit and overload protection
- 16 digital configurable inputs/outputs 24 V DC (C0 to C15) in 1 group (2.0...2.7 and 4.0...4.7), each of which can be used
  - as an input,
  - as a transistor output with short-circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.
- Optional with fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 ↬ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

The device is plugged on a terminal unit ↬ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

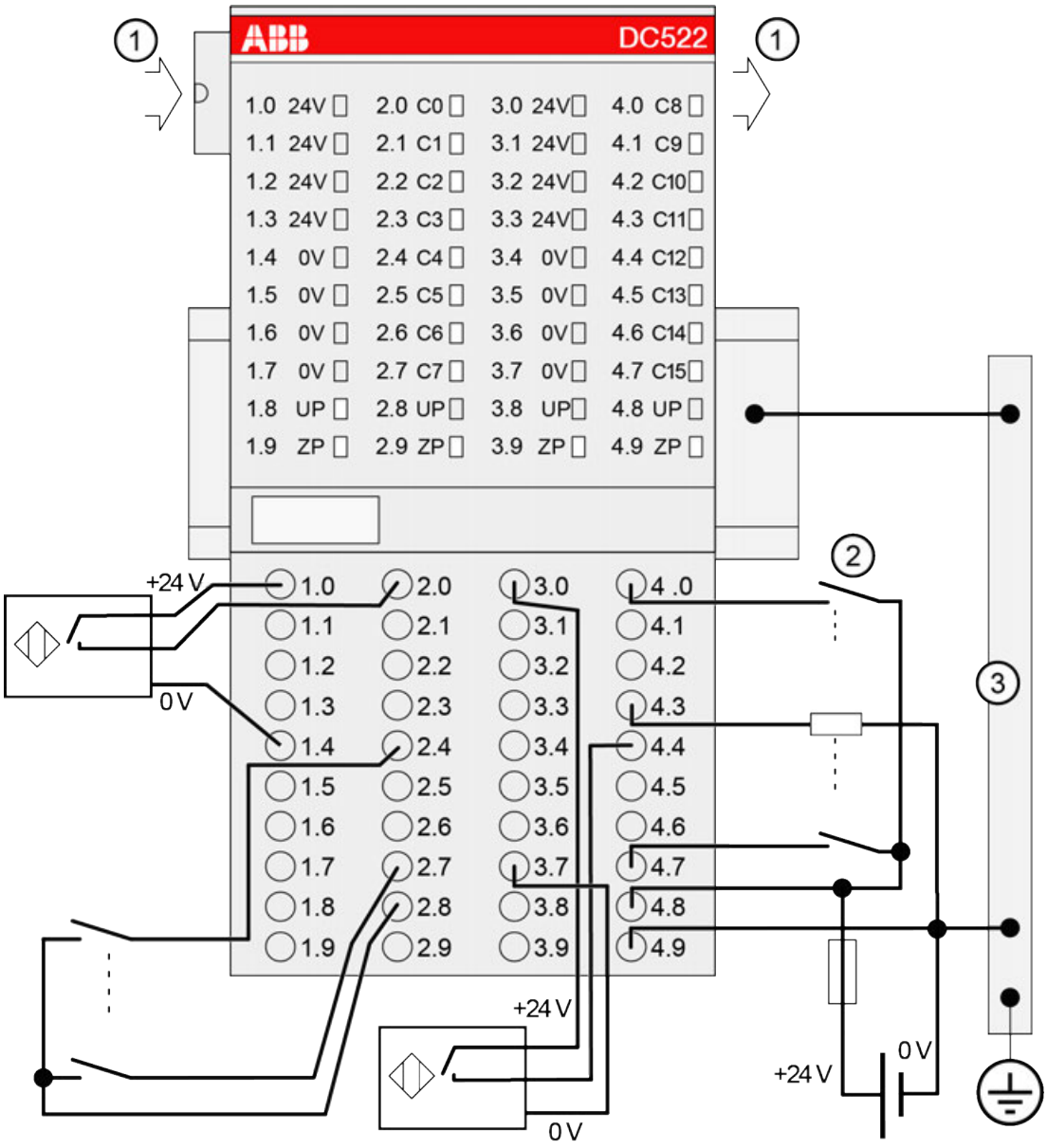
Connections

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.3	+24 V	4 x sensor power supply sources (loadable with 0.5 A in total)
1.4 to 1.7	0 V	0 V (reference potential)
2.0 to 2.7	C0 to C7	8 digital inputs/outputs
3.0 to 3.3	+24 V	4 x sensor power supply sources (loadable with 0.5 A in total)
3.4 to 3.7	0 V	0 V (reference potential)
4.0 to 4.7	C8 to C15	8 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DC522.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!
Removal/Insertion under power
Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!
Risk of damaging the PLC modules!
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!
Risk of influences to the connected sensors!
Some sensors may be influenced by the deactivated module outputs of DC522.

Connect a 470 Ω / 1 W resistor in series to inputs C8/C9 if they are used as fast counter inputs to avoid any influences.

The modules provide several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	2	4
Digital outputs (bytes)	2	4
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O Configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1220 [1])	Word	1220 0x04C4	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	7	Byte	7-CPU 6-FBP	0	255	0x0Y02
Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Short-circuit detection of output or sensor supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y05
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y06
Substitute value at outputs Bit 15 = Output 15 Bit 0 = Output 0	0... 65535	0... 0xffff	Word	0 0x0000	0	65535	0x0Y07

Remarks:

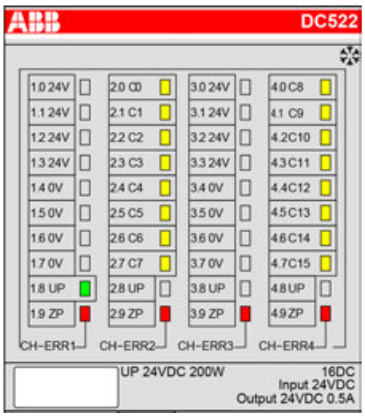
[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
[4])	With FBP or CS31 without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	9
Ext_User_Prm_Data_Const(0) =	0x04, 0xc5, 0x06, \ 0x01, 0x02, 0x01, 0x00, 0x00, 0x00;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs/ outputs C0...C15	Digital input or digital output	Yellow	Input/output = OFF	Input/output = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel Error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.005 A^2s

Parameter		Value
Max. power dissipation within the module		6 W (outputs unloaded)
Sensor power supply		
	Connections	Terminals 1.0...1.3 = +24 V, 1.4...1.7 = 0 V Terminals 3.0...3.3 = +24 V, 3.4...3.7 = 0 V
	Voltage	24 V DC with short-circuit and overload protection
	Loadability	Terminals 1.0...1.3, in total max. 0.5 A Terminals 3.0...3.3, in total max. 0.5 A
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		16 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
If the channels are used as inputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8...C15	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8 C15	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)

Parameter	Value
Monitoring point of input/output indicator	LED is part of the input circuitry
Galvanic isolation	From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		Max. 16 digital inputs
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter	Value
Number of channels per module	Max. 16 transistor outputs
Reference potential for all outputs	Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)

Parameter		Value
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

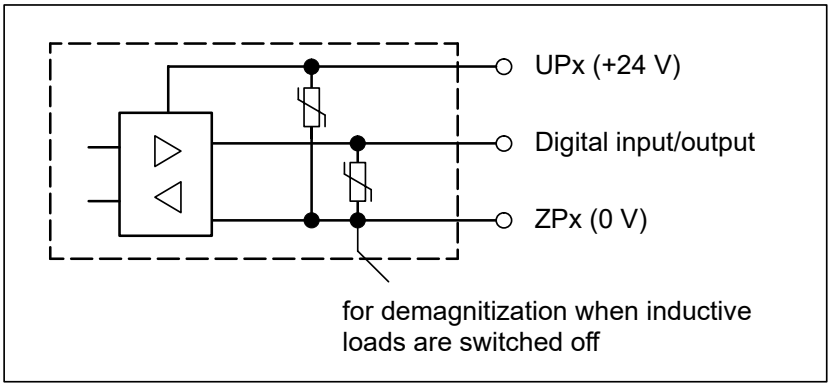


Fig. 119: Digital input/output (circuit diagram)

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	C8 / C9
Used outputs	C10
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

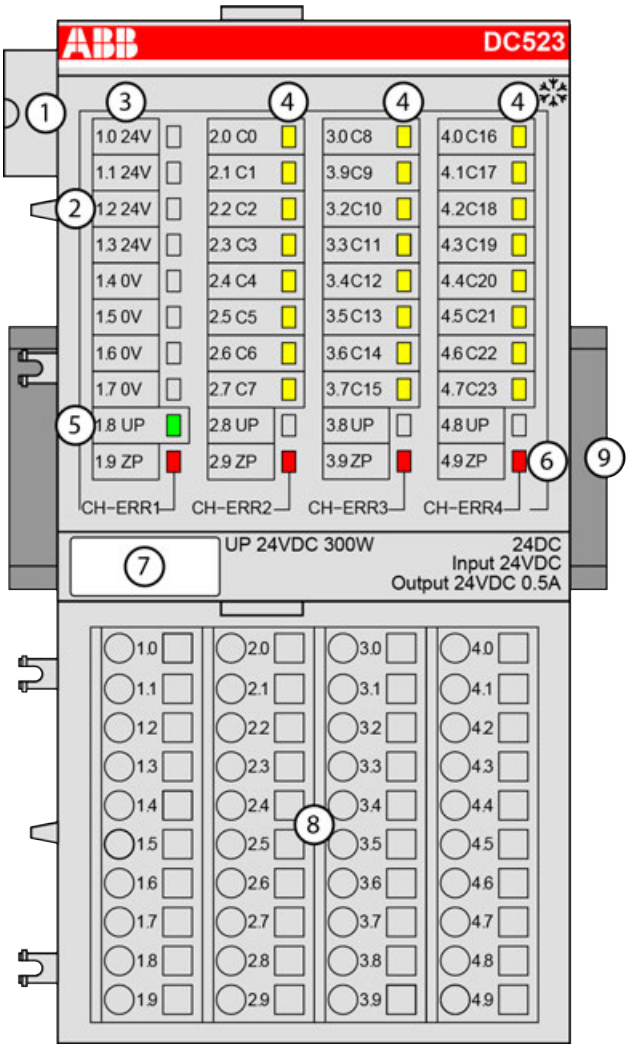
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 600 R0001	DC522, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires	Active
1SAP 440 600 R0001	DC522-XC, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC523 - Digital input/output module

- 24 configurable digital inputs/outputs
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 Sensor power supply 24 V DC / 0.5 A
4 24 yellow LEDs to display the signal states at the digital inputs/outputs (C0 - C23)
5 1 green LED to display the status of the process supply voltage UP
6 4 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input/output unit.

- 1 sensor supply voltage 24 V DC, 0.5 A, with short circuit and overload protection
- 24 digital configurable inputs/outputs 24 V DC (C0 to C23) in 1 group (2.0...2.7, 3.0...3.7 and 4.0...4.7), of which each can be used
  - as an input,
  - as a transistor output with short circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.
- Optional with fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

The device is plugged on a terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

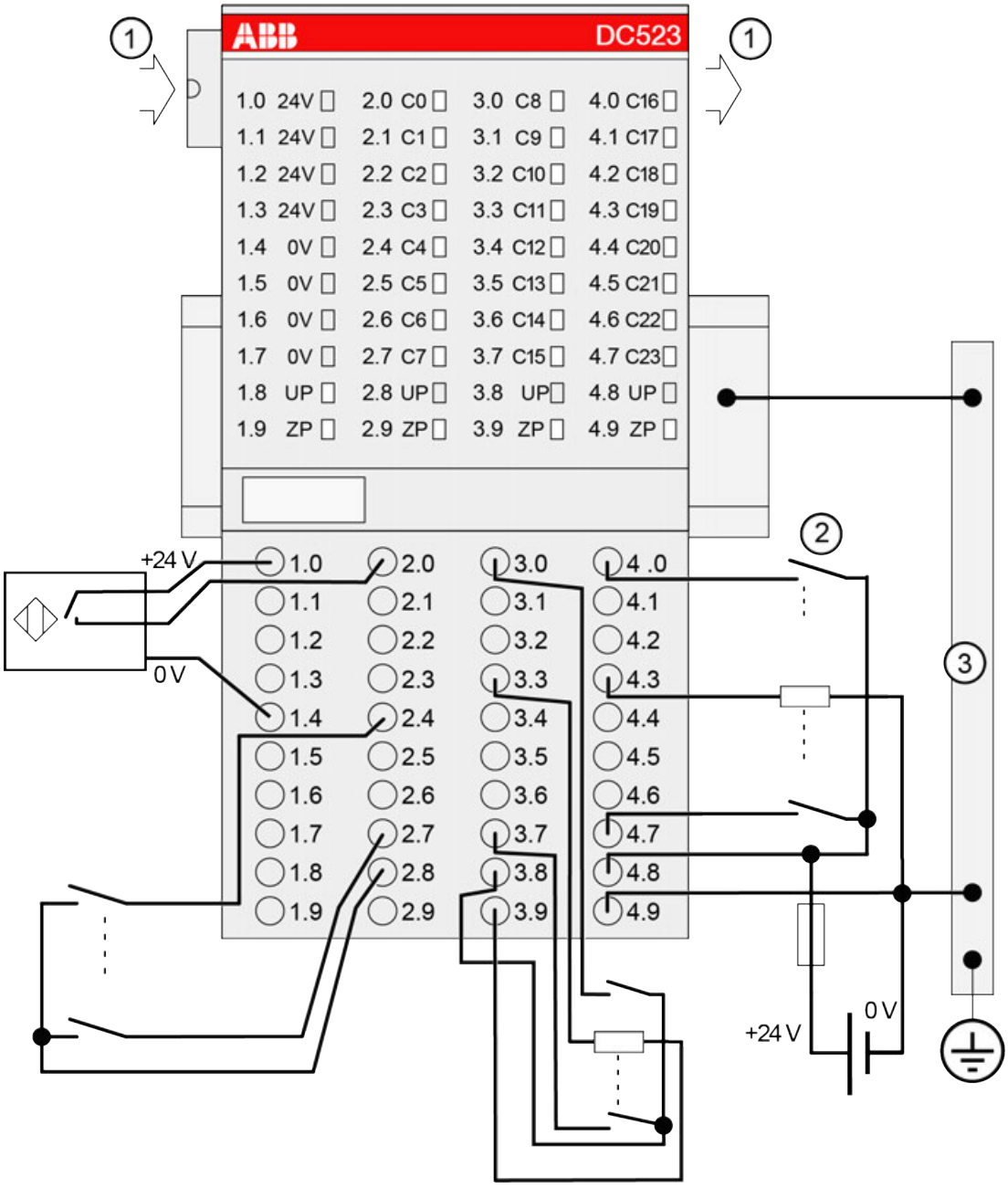
Connections

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.3	+24 V	4 x sensor power supply sources (loadable with 0.5 A in total)
1.4 to 1.7	0 V	0 V (reference potential)
2.0 to 2.7	C0 to C7	8 digital inputs/outputs

Terminals	Signal	Description
3.0 to 3.7	C8 to C15	8 digital inputs/outputs
4.0 to 4.7	C16 to C23	8 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DC523.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!
Removal/Insertion under power
Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!
Risk of damaging the PLC modules!
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!
Risk of influences to the connected sensors!
Some sensors may be influenced by the deactivated module outputs of DC523.

Connect a 470 Ω / 1 W resistor in series to inputs C16/C17 if they are used as fast counter inputs to avoid any influences.

The modules provide several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	3	5
Digital outputs (bytes)	3	5
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1215 [1])	Word	1215 0x04BF	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	9	Byte	9-CPU 8-FBP	0	255	0x0Y02

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Check supply	Off on	0 1	Byte	On 0x01	0	1	0x=Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast counter [4]	0 : 10 [3]	0 : 10	Byte	Mode 0 0x00			Not for FBP
Short circuit detection of output or sensor supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y05
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y06
Substitute value at outputs B23 = Output 23 Bit 0 = Output 0	0... 16777215	0... 0x00ff-ffff	DWord	0 0x0000 -0000	0	224−1	0x0Y07

Remarks:

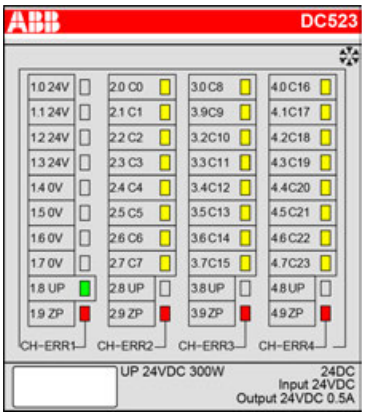
[1]	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2]	Not with FBP
[3]	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
[4]	With FBP or CS31 without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	11
Ext_User_Prm_Data_Const(0) =	0x04, 0xc0, 0x08, \ 0x01, 0x02, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs/ outputs C0...C23	Digital input or digital output	Yellow	Input/output = OFF	Input/output = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		

Parameter		Value
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.1 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.008 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Sensor power supply		
	Connections	Terminals 1.0...1.3 = +24 V, 1.4...1.7 = 0 V
	Voltage	24 V DC with short circuit and overload protection
	Loadability	Terminals 1.0...1.3, in total max. 0.5 A
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		24 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 24 channels
If the channels are used as inputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8...C15	Terminals 3.0...3.7
	Channels C16...C23	Terminals 4.0...4.7
If the channels are used as outputs		

Parameter		Value
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8 C15	Terminals 3.0...3.7
	Channels C16...C23	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Galvanic isolation		From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

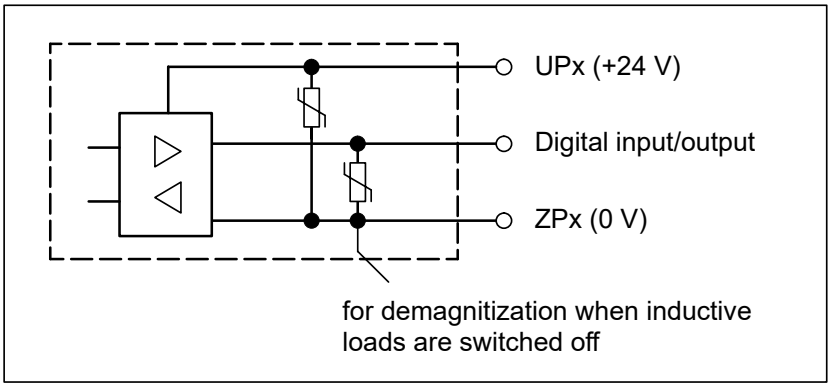
Parameter		Value
Number of channels per module		Max. 24 digital inputs
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 24 transistor outputs
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	C16 / C17
Used outputs	C18
Counting frequency	Max. 50 kHz

Chapter 1.6.5.1.12 “Fast counters” on page 3570

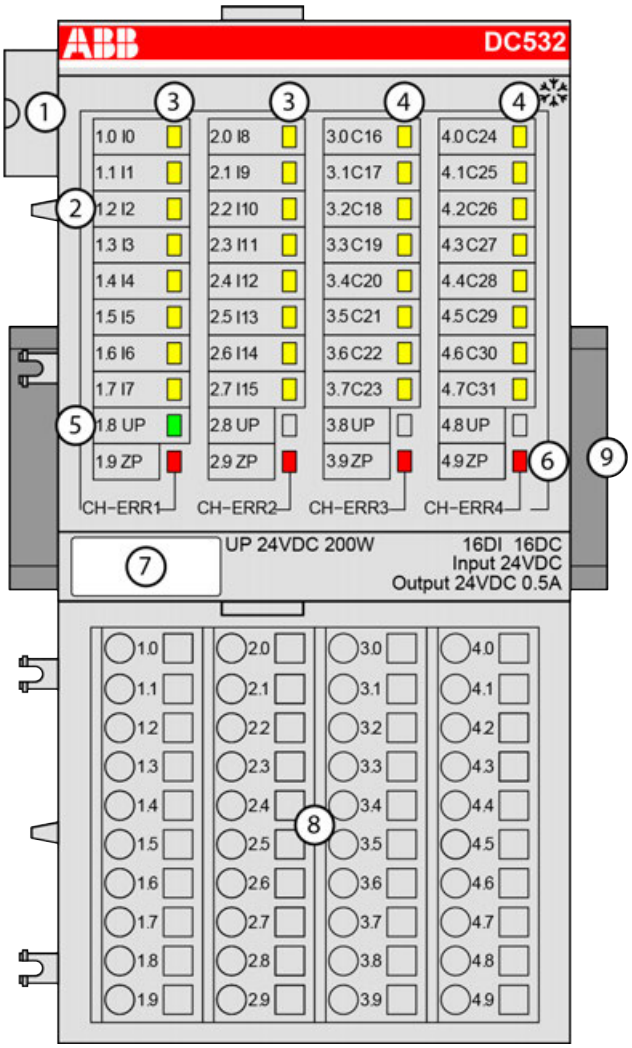
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 500 R0001	DC523, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire	Active
1SAP 440 500 R0001	DC523-XC, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC532 - Digital input/output module

- 16 digital inputs 24 V DC, 16 configurable digital inputs/outputs
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states at the digital inputs (I0 - I15)
4 16 yellow LEDs to display the signal states at the digital inputs/outputs (C16 - C31)
5 1 green LED to display the state of the process supply voltage UP
6 4 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input / output unit.

- 16 digital inputs 24 V DC in 2 groups (1.0...1.7 and 2.0...2.7)
- 16 digital configurable inputs/outputs 24 V DC (C16 to C31) in 1 group (3.0...3.7 and 4.0...4.7), of which each can be used
  - as an input,
  - as a transistor output with short circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.
- Optional with fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Digital inputs	16 (24 V DC)
Digital inputs/outputs	16 (24 V DC)
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

The device is plugged on a terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

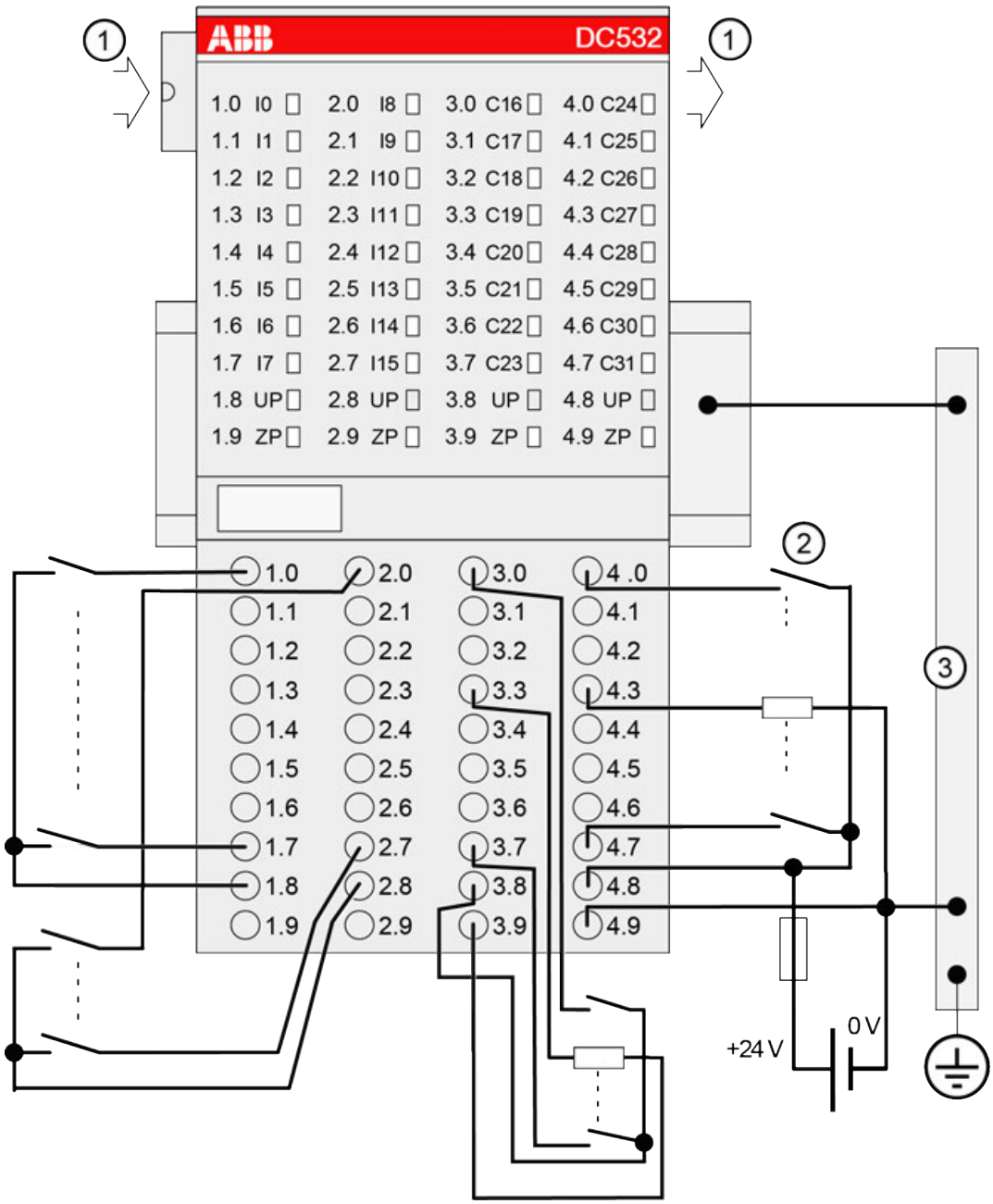
Connections

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0 to I7	8 digital inputs
2.0 to 2.7	I8 to I15	8 digital inputs
3.0 to 3.7	C16 to C23	8 digital inputs/outputs
4.0 to 4.7	C24 to C31	8 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DC532.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!
Removal/Insertion under power
Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!
Risk of damaging the PLC modules!
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of DC532.

Connect a 470 Ω / 1 W resistor in series to inputs C24/C25 if using them as fast counter inputs to avoid any influences.

The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	4	6
Digital outputs (bytes)	2	4
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	
Module ID	Internal	1200 [1])	Word	1200 0x04B0	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	7	Byte	7-CPU 6-FBP	0	255	0x0Y02
Check supply	Off on	0 1	Byte	On 0x01	0	1	0x0Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Output short circuit detection	Off On	0 1	Byte	On 0x01	0	1	0x0Y05
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y06
Substitute value at outputs Bit 15 = Output 15 Bit 0 = Output 0	0... 65535	0... 0xffff	Word	0 0x0000	0	65535	0x0Y07

Remarks:

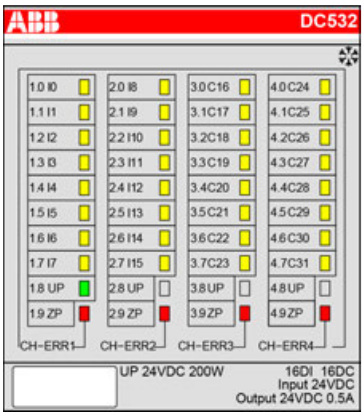
[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
[4])	With FBP or CS31 without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	9
Ext_User_Prm_Data_Const(0) =	0x04, 0xb1, 0x06, \ 0x01, 0x02, 0x01, 0x00, 0x00, 0x00;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I15	Digital input	Yellow	Input = OFF	Input = ON [1])	--
	Inputs/ outputs C16...C31	Digital input/ output	Yellow	Input/output = OFF	Input/output = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel Error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module Error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)

Parameter		Value
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.007 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	16
Distribution of the channels into groups	1 group of 16 channels
Terminals of the channels I0 to I7	1.0 to 1.7
Terminals of the channels I8 to I15	2.0 to 2.7
Reference potential for all inputs	Terminals 1.9, 2.8, 3.8 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation	From the rest of the module (I/O bus)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)

Parameter		Value
Monitoring point of input indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	> +5 V...< +15 V Parameter
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		16 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
If the channels are used as inputs		
	Channels I16...I23	Terminals 3.0...3.7
	Channels I24...I31	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels Q16...Q23	Terminals 3.0...3.7
	Channels Q24...Q31	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Galvanic isolation		From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		Max. 16 digital inputs
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Input current, per channel		See Technical Data of the Digital Inputs ↳ *Chapter 1.6.3.6.1.2.3.8.1 “Technical data of the digital inputs” on page 2725*
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
Signal 0		-3 V...+5 V *)
undefined signal		> +5 V...< +15 V
Signal 1		+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

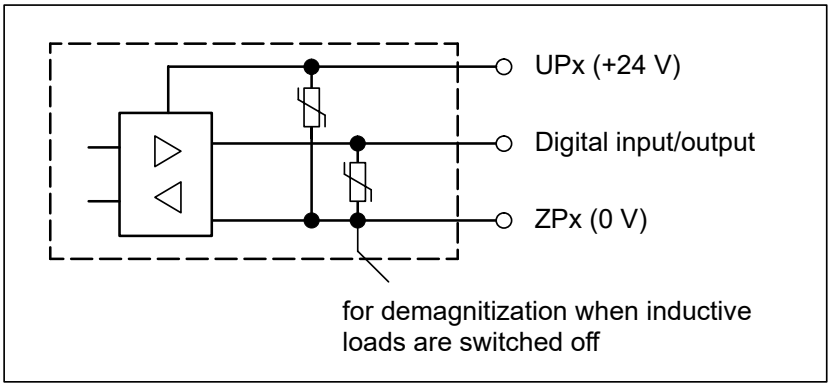
*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 16 transistor outputs
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request

Parameter		Value
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	C24 / C25
Used outputs	C26
Counting frequency	Max. 50 kHz

Chapter 1.6.5.1.12 “Fast counters” on page 3570

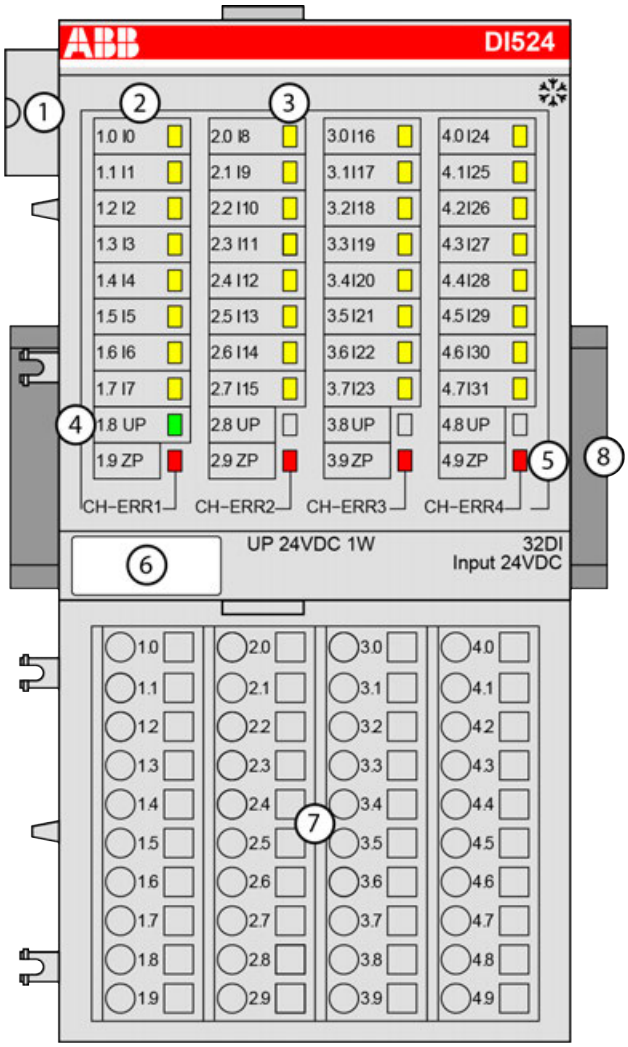
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 100 R0001	DC532, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire	Active
1SAP 440 100 R0001	DC532-XC, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire, XC version	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI524 - Digital input module

- 32 digital inputs 24 V DC in 4 groups (1.0...1.7, 2.0...2.7, 3.0...3.7 and 4.0...4.7)
- Fast counter
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name

3 32 yellow LEDs to display the signal states at the digital inputs (I0 - I31)
4 1 green LED to display the state of the process supply voltage UP
5 4 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Via the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal units	TU515 or TU516 ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

The device is plugged on a terminal unit ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and have always the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC

Table 479: Assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0 to I7	8 digital inputs
2.0 to 2.7	I8 to I15	8 digital inputs
3.0 to 3.7	I16 to I23	8 digital inputs
4.0 to 4.7	I24 to I31	8 digital inputs

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DI524.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter ↳ *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

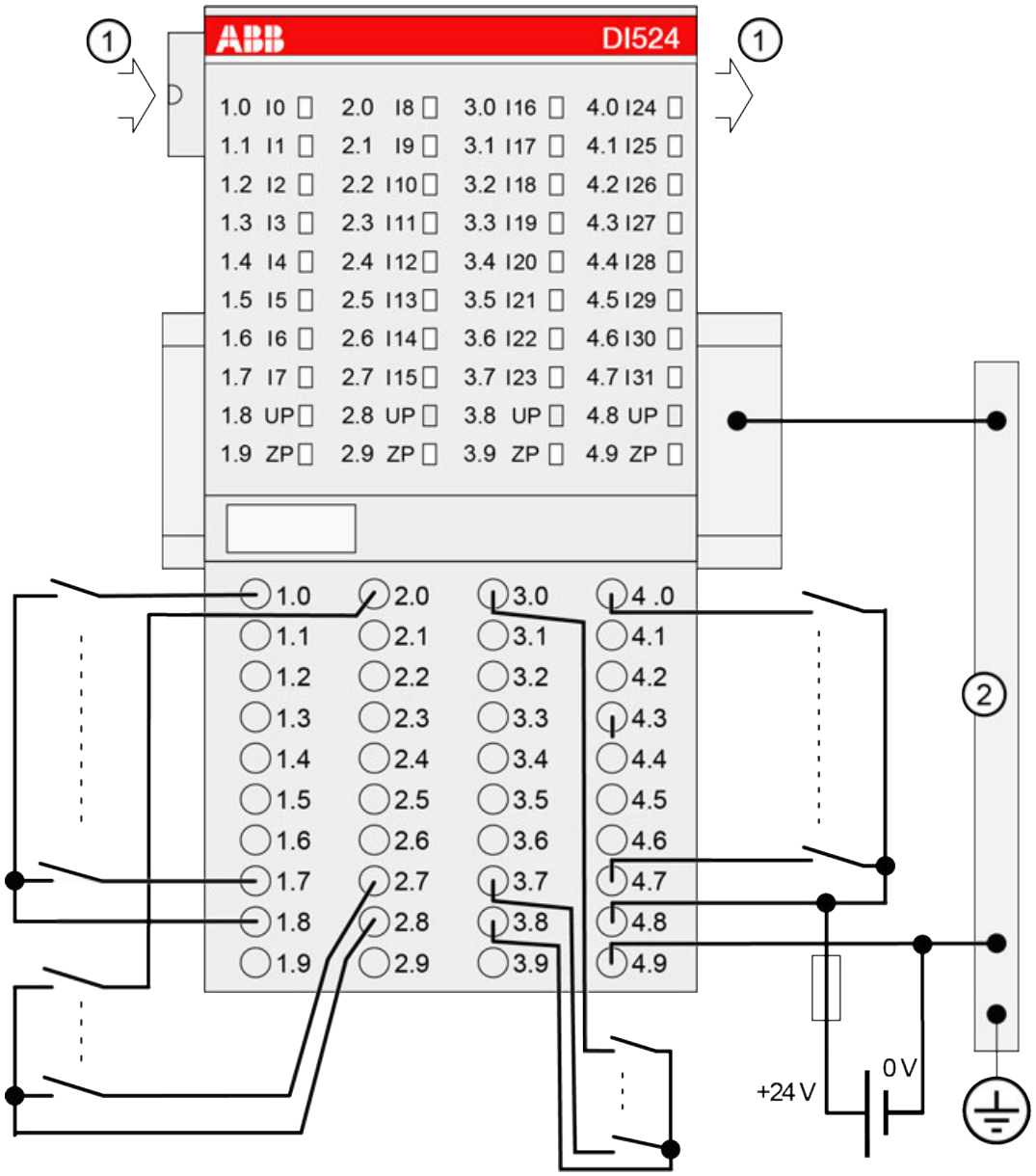
The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.



1 I/O bus
2 switchgear cabinet earth

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	4	6
Digital outputs (bytes)	0	2

	Without the fast counter	With the fast counter (only with AC500)
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1000 [1])	Word	1000 0x03E8	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
3	Parameter length	Internal	3-CPU 2-FBP	Byte	3 2	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
5	Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
6	Fast counter [4)]	0 : 10 [3)]	0 : 10	Byte	Mode 0 0x00			Not for FBP

Remarks:

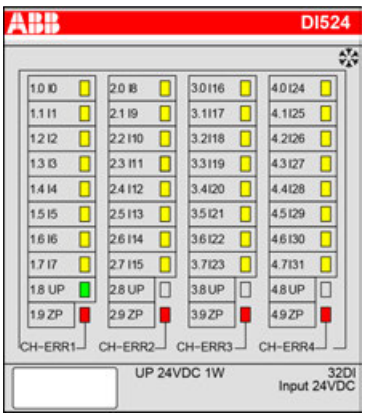
[1)]	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2)]	Not with FBP
[3)]	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
[4)]	With FBP or CS31 without the parameter Fast counter

GSD file:

Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	5 0x03, 0xe9, 0x02, \ 0x01, 0x02;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I31	Digital input	Yellow	Input = OFF	Input = ON [1)]	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--

LED		State	Color	LED = OFF	LED = ON	LED flashes
	CH-ERR1	Channel error, error messages in groups (digital inputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module	ca. 2 mA
	From UP at normal operation	0.15 A
	Inrush current from UP (at power up)	0.008 A^2s
Weight (without terminal unit)		ca. 105 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		32
Distribution of the channels into groups		1 group of 32 channels
Terminals of the channels I0 to I7		1.0 to 1.7
Terminals of the channels I8 to I15		2.0 to 2.7
Terminals of the channels I16 to I23		3.0 to 3.7
Terminals of the channels I24 to I31		4.0 to 4.7
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module (I/O bus)
Indication of the input signals		One yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0 -> 1 or 1 -> 0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	I24 / I25
Used outputs	None
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

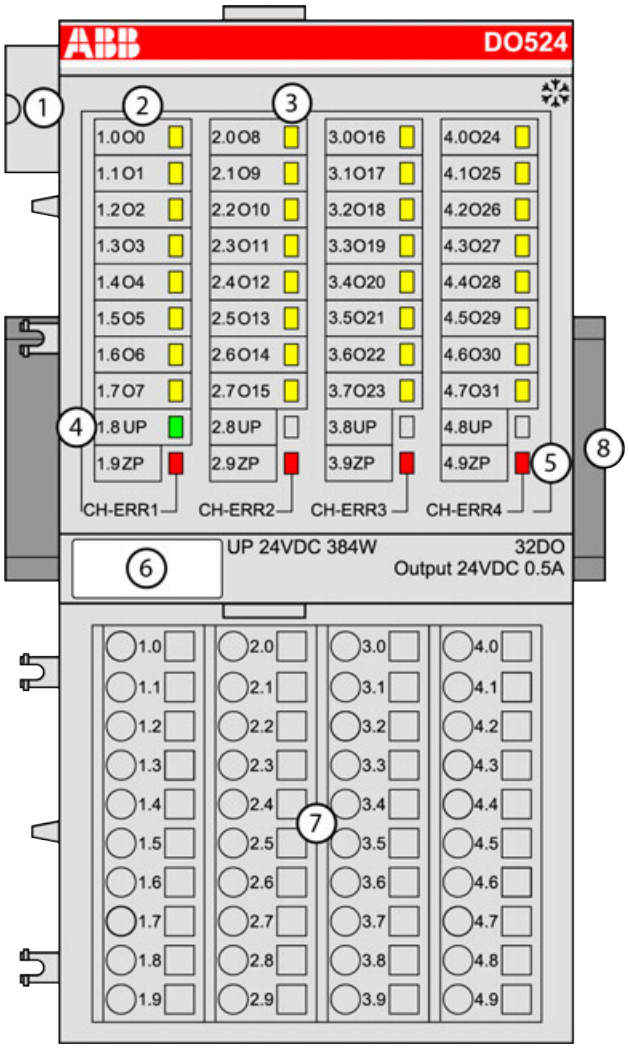
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 000 R0001	DI524, digital input module, 32 DI, 24 V DC, 1-wire	Active
1SAP 440 000 R0001	DI524-XC, digital input module, 32 DI, 24 V DC, 1-wire, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO524 - Digital output module

- 32 digital outputs 24 V DC / 0.5 A in 4 groups (1.0...4.7) with short circuit and overload protection
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 32 yellow LEDs to display the signal states at the digital outputs (O0 - O31)
4 1 green LED to display the state of the process supply voltage UP
5 4 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels.

Functionality

Parameter	Value
LED displays	For signal states, errors and supply voltage
Internal power supply	Via the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

The device is plugged on a terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	O0 to O7	8 digital outputs
2.0 to 2.7	O8 to O15	8 digital outputs
3.0 to 3.7	O16 to O23	8 digital outputs
4.0 to 4.7	O24 to O31	8 digital outputs

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DO524.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

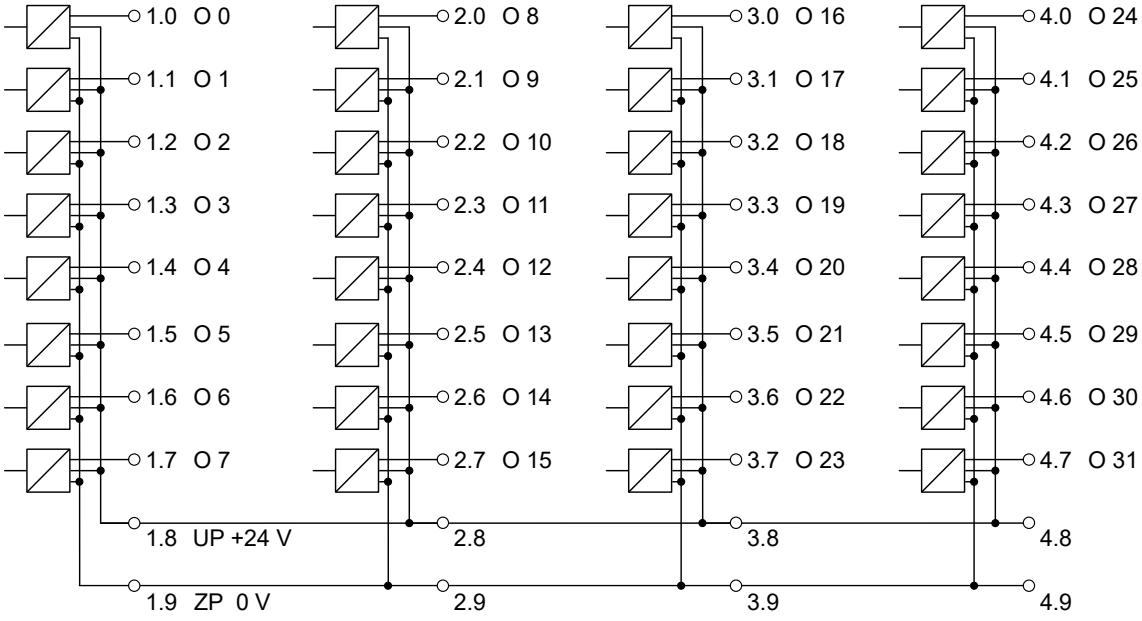
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following block diagram shows the internal construction of the digital outputs:



The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	4

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Module ID	Internal	1101 [1])	WORD	1101 0x044D	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	BYTE	No 0x00			not for FBP
Parameter length	Internal	7	BYTE	7-CPU 7-FBP	0	255	0x0Y02
Check supply	Off on	0 1	BYTE	On 0x01	0	1	0x0Y03
Output short circuit detection	Off On	0 1	BYTE	On 0x01	0	1	0x0Y04

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	BYTE	Off 0x00	0	2	0x0Y05
Substitute value at outputs Bit 31 = Output 31 Bit 0 = Output 0	0... 4294967295	0... 0xffffffff	DWORD	0 0x00000000	0	4294967295	0x0Y06

[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1

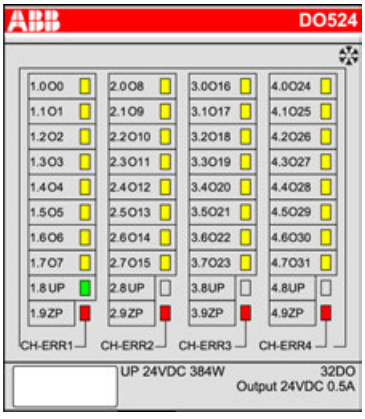
[2]) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	10
Ext_User_Prm_Data_Const(0) =	0x04, 0x4d, 0x07, \ 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Outputs O0...O31	Digital output	Yellow	Output = OFF	Output = ON	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.10 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.005 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		Ca. 100 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

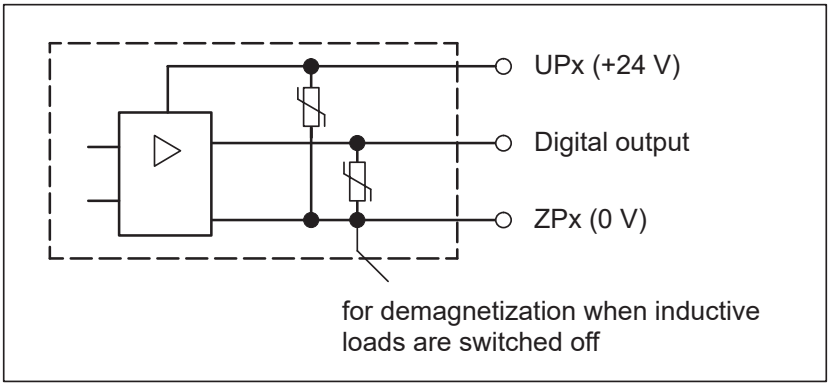
Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		32 outputs (with transistors)
Distribution of the channels into groups		1 group of 32 channels
Connection of the channels		
	O0 to O7	Terminals 1.0 to 1.7
	O8 to O15	Terminals 2.0 to 2.7
	O16 to O23	Terminals 3.0 to 3.7
	O24 to O31	Terminals 4.0 to 4.7
Indication of the output signals		1 yellow LED per channel, the LED is ON if the output signal is high (signal 1)
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0 -> 1 or 1 -> 0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (channels O0 to O15)	4 A
	Maximum value (channels O16 to O31)	4 A
	Maximum value (all channels together)	8 A
Max. leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit proof / overload proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short-circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital output with the varistors for demagnetization when inductive loads are switched off.



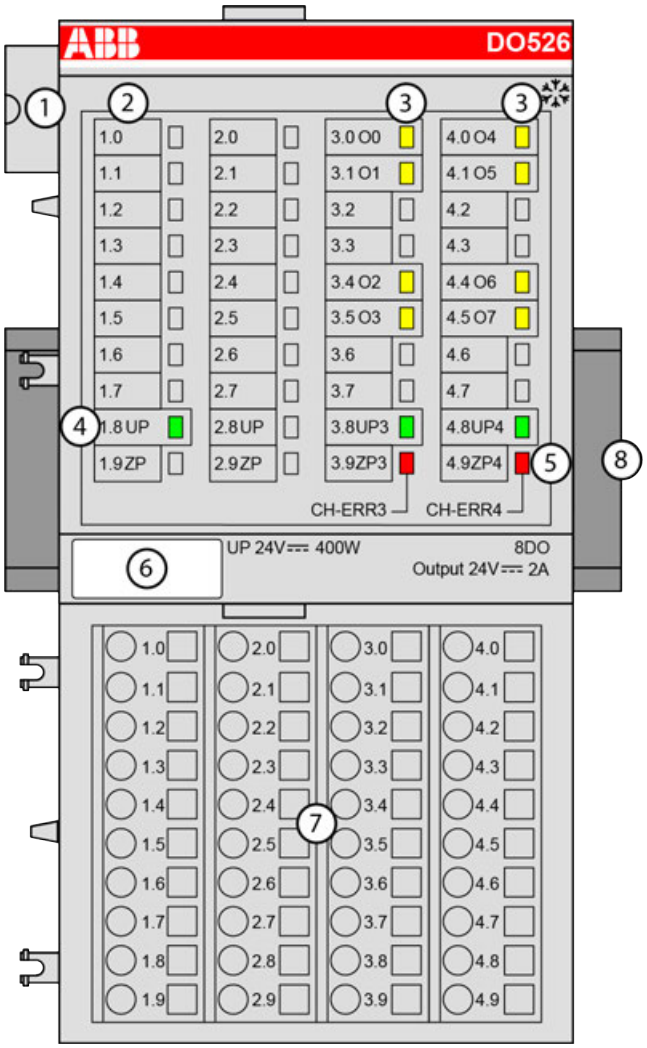
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 700 R0001	DO524, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire	Active
1SAP 440 700 R0001	DO524-XC, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO526 - Digital output module

- 8 digital outputs 24 V DC (O0 to O7) in 2 groups without short circuit and without overload protection.
- Module and group-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the outputs O0 to O7
4 3 green LEDs to display the states of the process supply voltage UP, UP3 and UP4
5 2 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN-rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

Potential separation between the channel groups.

Functionality

Parameter	Value
LED displays	For signal states, errors and supply voltages
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP, ZP3, ZP4, UP, UP3 and UP4 (process voltage 24 V DC)
Required terminal unit	TU542 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

The output module is plugged on the terminal unit TU542. Properly position the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 2.8 and 1.9 to 2.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 2.8:	Process voltage UP = +24 V DC
Terminals 1.9 to 2.9:	Process voltage ZP = 0 V
Terminal 3.8:	Process voltage UP3 = +24 V DC
Terminal 3.9:	Process voltage ZP3 = 0 V
Terminal 4.8:	Process voltage UP4 = +24 V DC
Terminal 4.9:	Process voltage ZP4 = 0 V

Terminals	Signal	Description
3.0, 3.1, 3.4, 3.5	O0 to O3	4 digital outputs
4.0, 4.1, 4.4, 4.5	O4 to O7	4 digital outputs

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DO526.

The external power supply connection is carried out via the UP, UP3, UP4 (+24 V DC) and the ZP, ZP3, ZP4 (0 V DC) terminals.

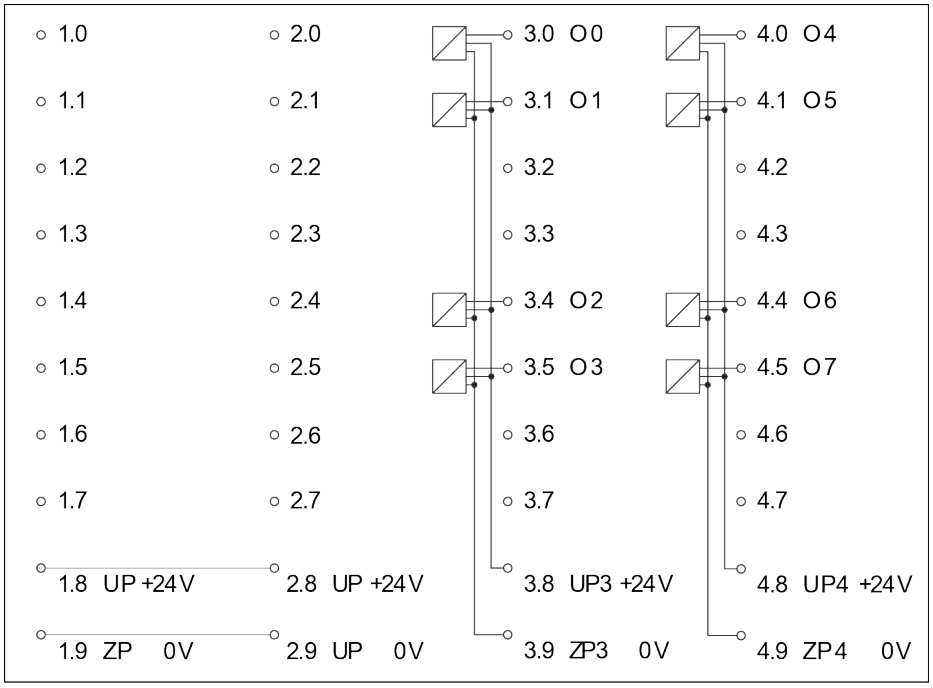
NOTICE!

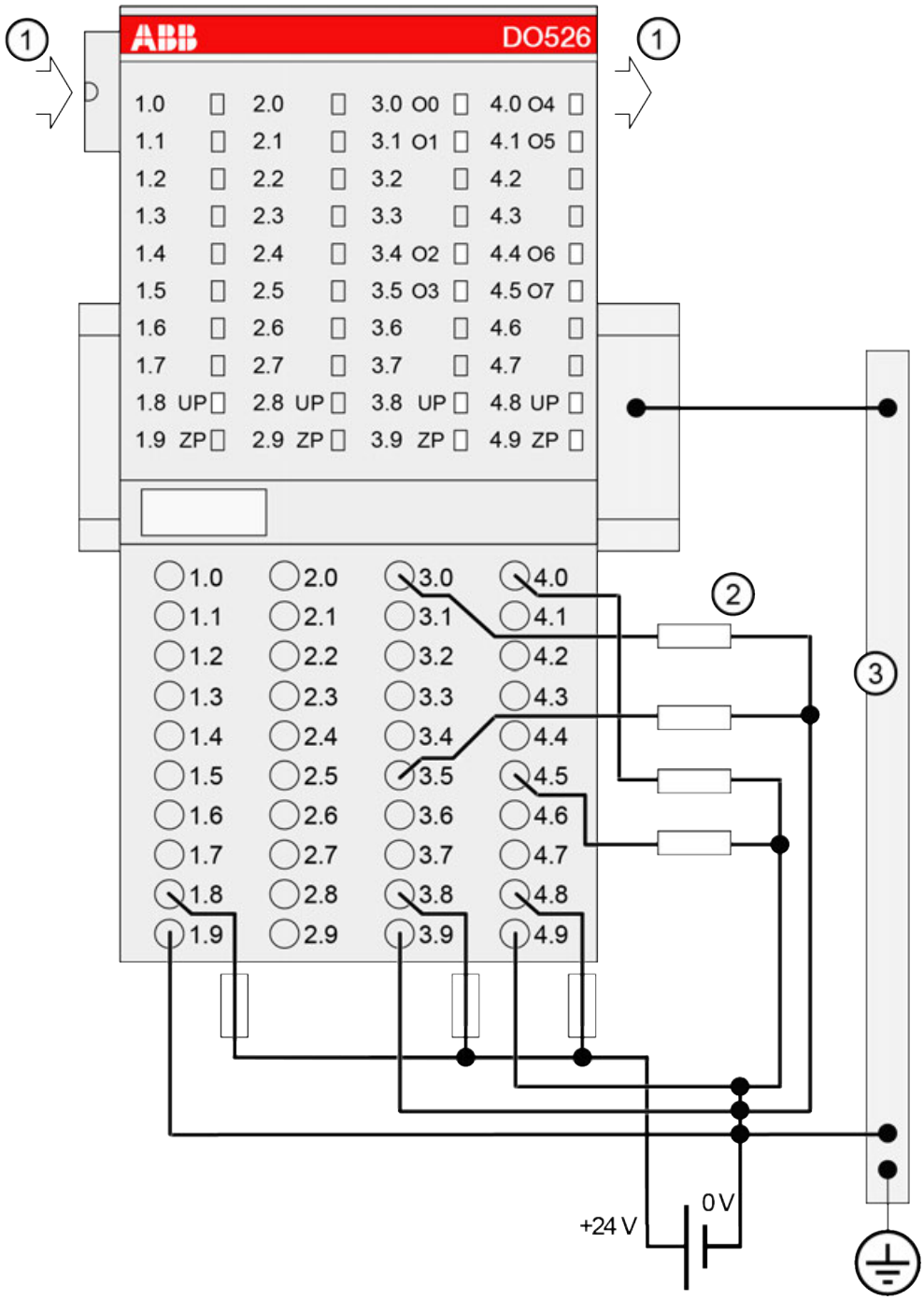
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following block diagram shows the internal construction of the digital outputs:





1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	1

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software, versions ≥ 1.2.3.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...7

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Module ID	Internal	1105 [1])	WORD	1105 0x0451	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	BYTE	No 0x00			not for FBP
Parameter length	Internal	6	BYTE	6-CPU 6-FBP	0	6	0x0Y02
Check supply	Off on	0 1	BYTE	On 0x01	0	1	0x0Y03
Reserve	0...255	0...0xff	BYTE	On 0x01	0	1	0x0Y04
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	BYTE	Off 0x00	0	2	0x0Y05

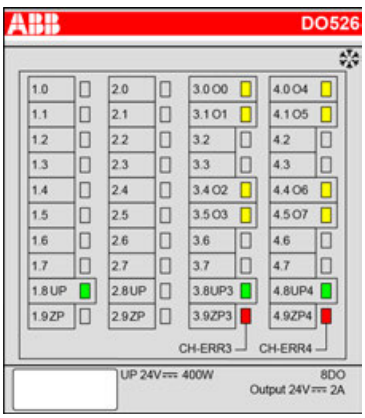
Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Substitute value at outputs Bit 7 = Output 7 Bit 0 = Output 0	0...255	0...0xff	BYTE	0x00	0	255	0x0Y06
Reserve	0...255	0...0xff	BYTE	0x00	0	255	0x0Y07
Reserve	0...255	0...0xff	BYTE	0x00	0	255	0x0Y08
[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1 [2]) Not with FBP							

GSD file:

Ext_User_Prm_Data_Len =	10
Ext_User_Prm_Data_Const(0) =	0x04, 0x51, 0x00, 0x06, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Outputs O0...O7	Digital output	Yellow	Output = OFF	Output = ON [2])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	UP3	Process supply voltage outputs 0...3 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	UP4	Process supply voltage outputs 4...7 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--

LED		State	Color	LED = OFF	LED = ON	LED flashes
	CH-ERR3	Channel Error, error messages in groups (digital outputs combined into the groups 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on in the corresponding group
	CH-ERR4		Red			
	CH-ERR [1])	Module Error	Red	--	Internal error	--
	[1]) All of the LEDs CH-ERR3 to CH-ERR4 light up together [2]) The state of the LEDs corresponds to the logic state of the output. In case of missing or low process supply voltage UP3 or UP4, the signal on the output terminal is off even though the LED is on.					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP, UP3 and UP4		
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) as well as 1.9 and 2.9 0 V (ZP) Terminals 3.8 for +24 V (UP3) as well as 3.9 for 0 V (ZP3) Terminals 4.8 for +24 V (UP4) as well as 4.9 for 0 V (ZP4)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP, UP3 and UP4	10 A fast (for each process supply voltage)
	Galvanic isolation	Yes, per module and per output channel groups
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	Ca. 20 mA + 1.5 mA per output
	From UP3 or UP4 at normal operation / with outputs	Ca. 0.01 A + max. 2 A per output
	Inrush current from UP (at power up)	0.015 A^2s

Parameter		Value
	Inrush current from UP3 or UP4 (at power up)	0.005 A^2s (without output load)
Max. power dissipation within the module		6 W
Weight (without terminal unit)		Ca. 135 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply and continuous overvoltage up to 30 V DC.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 outputs (with transistors, non-latching type)
Distribution of the channels into groups		2 groups of 4 channels
Connection of the channels		
	O0 to O3	Terminals 3.0, 3.1, 3.4, 3.5
	O4 to O7	Terminals 4.0, 4.1, 4.4, 4.5
Indication of the output signals		1 yellow LED per channel, the LED is ON if the output signal is high (signal 1)
Power supply voltage for the module		Terminals 1.8 and 2.8 (positive pole of the process supply voltage, signal name UP)
Reference potential for module power supply		Terminals 1.9 and 2.9 (negative pole of the process supply voltage, signal name ZP)
Power supply voltage for the outputs O0 to O3		Terminal 3.8 (positive pole of the process supply voltage, signal name UP3)
Reference potential for the outputs O0 to O3		Terminal 3.9 (negative pole of the process supply voltage, signal name ZP3)
Power supply voltage for the outputs O4 to O7		Terminal 4.8 (positive pole of the process supply voltage, signal name UP4)
Reference potential for the outputs O4 to O7		Terminal 4.9 (negative pole of the process supply voltage, signal name ZP4)
Output voltage for signal 1		UP (-0.4 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	2 A at UP3 or UP4 = 24 V

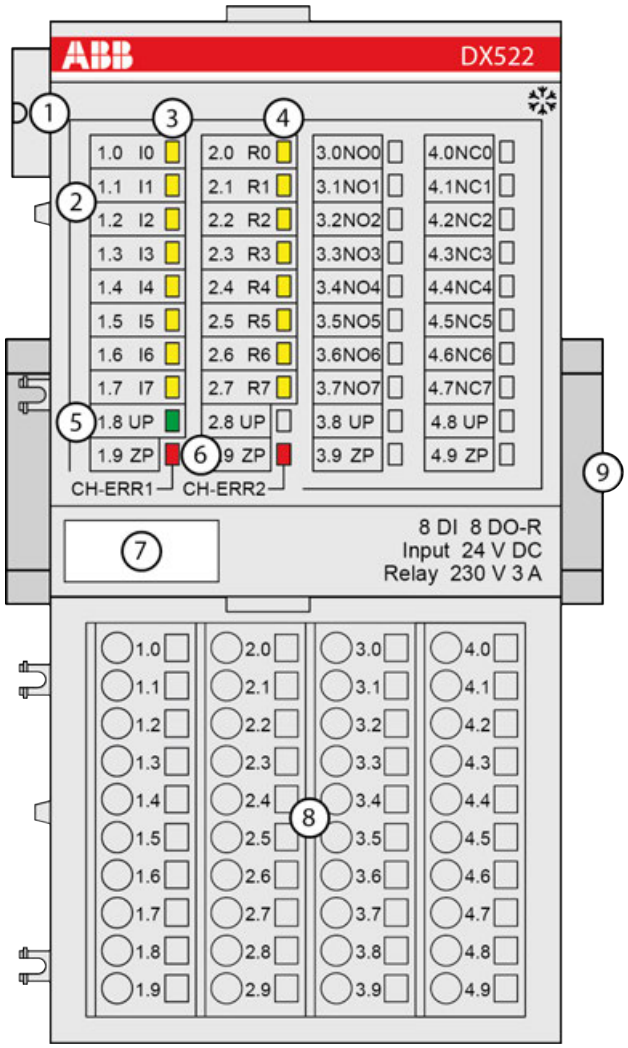
Parameter		Value
	Maximum value (channels O0 to O3)	8 A
	Maximum value (channels O4 to O7)	8 A
Leakage current with signal 0		< 0.1 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With clamp diode in output high side driver
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 2 Hz
	With lamp loads	Max. 11 Hz with max. 48 W
Short-circuit proof / overload proof		No (should be done externally)
Overload message		No
Output current limitation		No (should be done externally)
Resistance to feedback against 24 V signals		Yes to UP3 or UP4. No to outputs in same group.
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 800 R0001	DO526, digital output module, 8 DO, 24 V DC / 2 A, 1-wire	Active
1SAP 440 800 R0001	DO526-XC, digital output module, 8 DO, 24 V DC / 2 A, 1-wire, XC version	Active
1SAP 213 200 R0001	TU542, I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 413 200 R0001	TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version	Active

DX522 - Digital input/output module

- 8 digital inputs 24 V DC, module-wise galvanically isolated
- 8 relay outputs
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the digital inputs (I0 - I7)
4 8 yellow LEDs to display the signal states at the digital relay outputs (R0 - R7)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input/output unit.

- 8 digital inputs 24 V DC in 1 group (1.0...1.7)
- 8 digital relay outputs with one change-over contact each (R0...R7). All output channels are galvanically isolated from each other.
- Fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)
Required terminal units	TU531 or TU532 ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*

The device is plugged on a terminal unit ↳ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

Connections

WARNING!
Risk of death by electric shock!
Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and have always the same assignment, irrespective of the inserted module:

- Terminals 1.8 to 4.8: process supply voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process supply voltage ZP = 0 V DC

Table 480: Assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0 to I7	Input signals of the 8 digital inputs
1.8 to 4.8	UP	Process supply voltage +24 V DC
1.9 to 4.9	ZP	Reference potential for the 8 digital inputs and the process supply voltage
2.0	R0	Common contact of the first relay output
3.0	NO 0	Normally-open contact of the first relay output
4.0	NC 0	Normally-closed contact of the first relay output
2.1	R1	Common contact of the second relay output
3.1	NO 1	Normally-open contact of the second relay output
4.1	NC 1	Normally-closed contact of the second relay output
:	:	:
2.7	R7	Common contact of the eighth relay output
3.7	NO 7	Normally-open contact of the eighth relay output
4.7	NC 7	Normally-closed contact of the eighth relay output

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DX522.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions (see Diagnosis and State LEDs *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*).

The following figure shows the connection of the digital input/output module DX522.

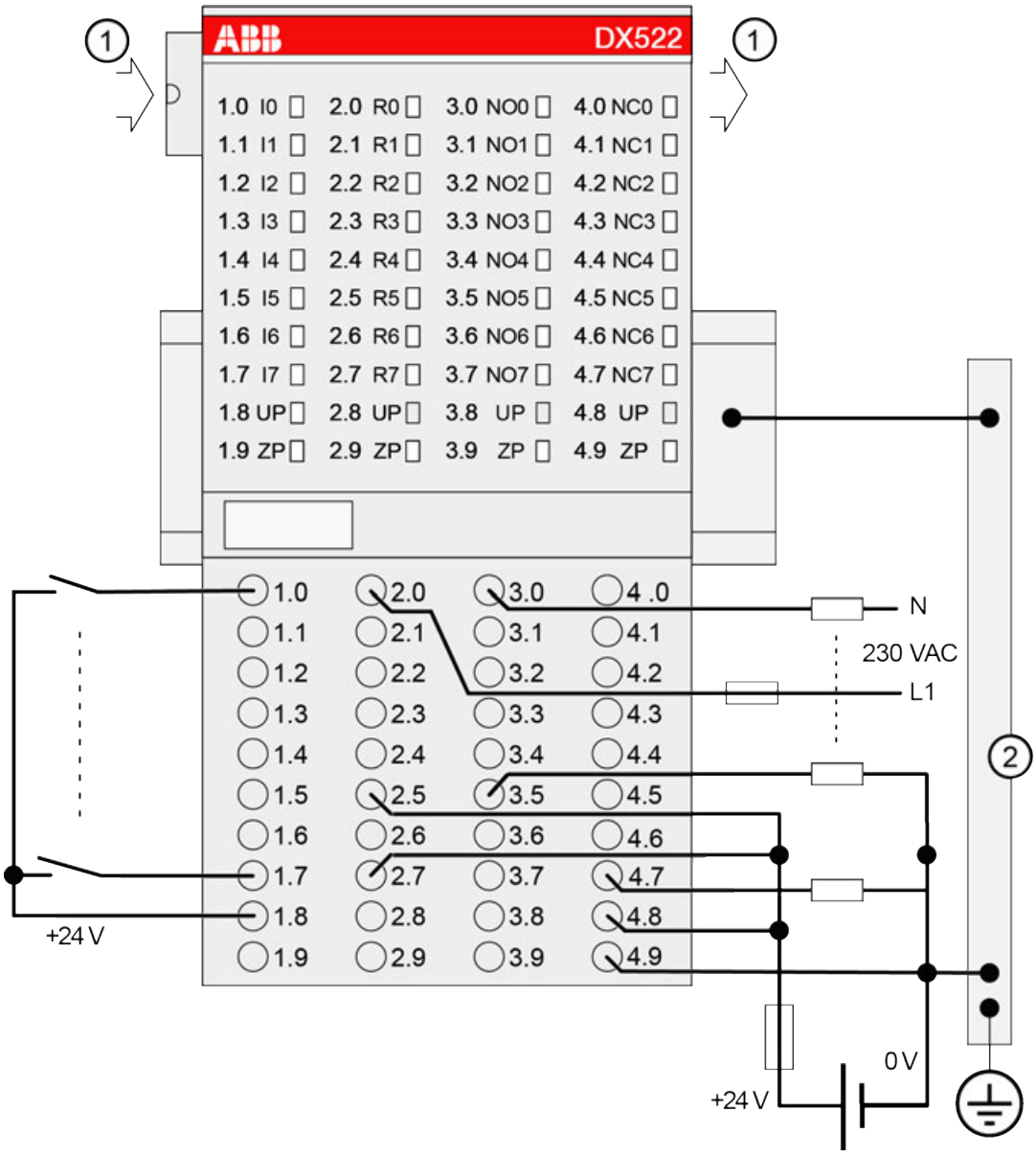


Fig. 120: Connection of the module

1 I/O bus
2 Switchgear cabinet earth

NOTICE!

– If the relay outputs have to switch inductive **DC loads**, free-wheeling diodes must be circuited in parallel to these loads.
– If the relay outputs have to switch inductive **AC loads**, spark suppressors are required.

CAUTION!

The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

NOTICE!

Risk of damaging the PLC module!

The following has to be considered when connecting input and output voltages to the module:

- All 230 V AC feeds must be single-phase from the same supply system.
- Connection of 2 or more relay contacts in series is possible; however, voltages above 230 V AC and 3-phase loads are not allowed.
- The 8 change-over contacts of the relays are galvanically isolated from channel to channel. This allows to connect loads of 24 V DC and 230 V AC to relay outputs of the same module. In such cases it is necessary that both supply voltages are grounded to prevent unsafe floating grounds.

NOTICE!

Risk of damaging the PLC module!

There is no internal short-circuit or overload protection for the relay outputs.

Protect the relay contacts by back-up fuses of 6 A max. (characteristic gG/gL). Depending on the application, fuses can be used for single channels or module-wise.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	1	3
Digital outputs (bytes)	1	3
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1210 [1])	Word	1210 0x04BA	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	5	Byte	5-CPU 4-FBP	0	255	0x0Y02
Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast Counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
Substitute value at outputs) Bit 7 = Output 7 Bit 0 = Output 0	0... 255	0... 0xff	Byte	0 0x00	0	255	0x0Y06

Remarks:

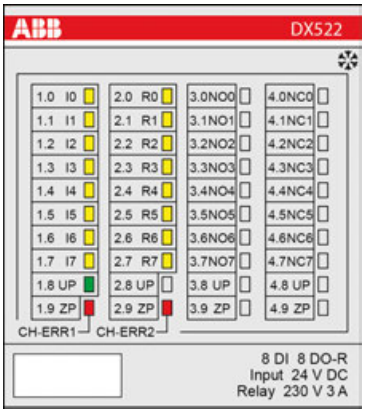
[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section ↬ *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
[4])	With FBP and without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	7
Ext_User_Prm_Data_Const (0) =	0x04, 0xbb, 0x04, \ 0x01, 0x02, 0x00, 0x00;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7	Digital input	Yellow	Input = OFF	Input = ON [1])	--
	Outputs R0...R7 (relays)	Digital output	Yellow	Relay output = OFF	Relay output = ON	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel Error, error messages in groups (digital inputs/ outputs combined into the groups 1 and 2)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group
	CH-ERR2		Red			
	CH-ERR [2])	Module Error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR2 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module	ca. 2 mA
	From UP at normal operation / with outputs	0.05 A + output loads
	Inrush current from UP (at power up)	0.010 A^2s
Max. power dissipation within the module		6 W (outputs OFF)
Weight (without terminal unit)		ca. 300 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels I0 to I7	1.0 to 1.7
Reference potential for all inputs	Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation	From the rest of the module (I/O bus)
Indication of the input signals	One yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator	LED is part of the input circuitry
Input type acc. to EN 61131-2	Type 1

Parameter		Value
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
Signal 0		-3 V...+5 V
Undefined signal		> +5 V...< +15 V
Signal 1		+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the relay outputs

Parameter		Value
Number of channels per module		8 relay outputs
Distribution of channels into groups		8 groups of 1 channel each
Connection of the channel R0		Terminal 2.0 (common), 3.0 (NO) and 4.0 (NC)
Connection of the channel R1		Terminal 2.1 (common), 3.1 (NO) and 4.1 (NC)
Connection of the channel R6		Terminal 2.6 (common), 3.6 (NO) and 4.6 (NC)
Connection of the channel R7		Terminal 2.7 (common), 3.7 (NO) and 4.7 (NC)
Galvanic isolation		Between the channels and from the rest of the module
Indication of the output signals		One yellow LED per channel, the LED is ON when the relay coil is energized
Monitoring point of output indicator		LED is controlled by process CPU
Way of operation		Non-latching type
Output delay (0->1 or 1->0)		On request
Relay power supply		By UP process supply voltage
Relay outputs		
	Output short circuit protection	Should be provided externally with a fuse or circuit breaker
Rated protection fuse		6 A gL/gG per channel
Min. switching current		10 mA
Output switching capacity		
	Resistive load, max.	3 A; 3 A (230 V AC), 2 A (24 V DC)
	Inductive load, max.	1.5 A; 1.5 A (230 V AC), 1.5 A (24 V DC)
	Lamp load	60 W (230 V AC), 10 W (24 V DC)

Parameter		Value
Output switching capacity (XC version above 60 °C)		On request
Lifetime (cycles)		Mechanical: 300 000; Under load: 300 000 (24 V DC at 2 A), 200 000 (120 V AC at 2 A), 100 000 (230 V AC at 3 A)
Spark suppression with inductive AC load		Must be performed externally according to driven load specifications
Demagnetization with inductive DC load		A free-wheeling diode must be circuited in parallel to the inductive load
Switching frequency		
	With resistive load	Max. 10 Hz
	With inductive load	Max. 2 Hz
	With lamp load	On request
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	I0 / I1
Used outputs	None
Counting frequency	50 kHz max.
Detailed description	See *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

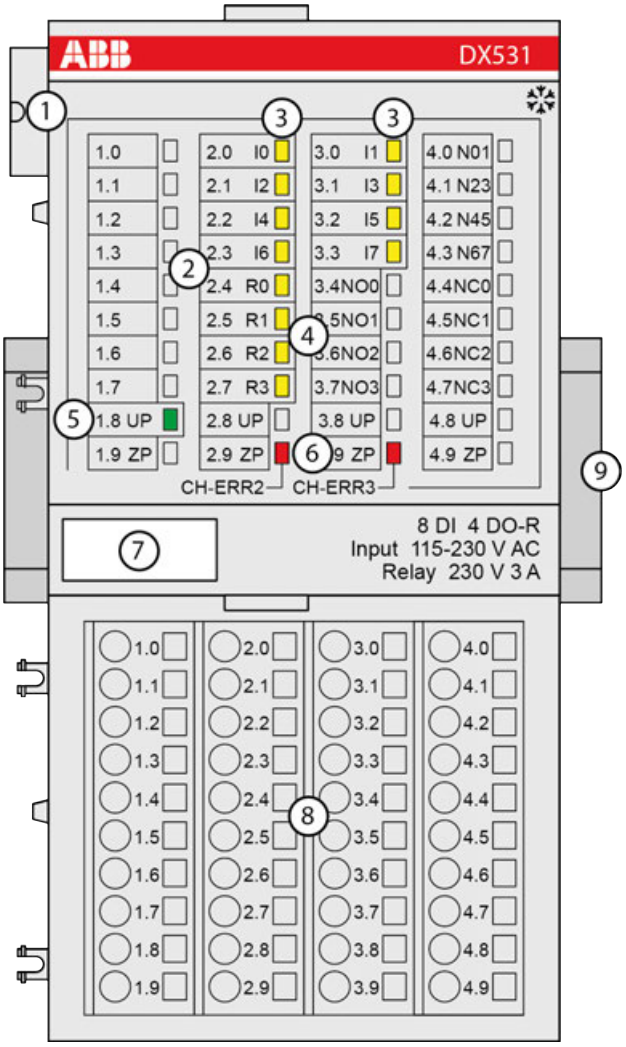
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 245 200 R0001	DX522, digital input/output module, 8 DI, 24 V DC, 8 DO relays	Active
1SAP 445 200 R0001	DX522-XC, digital input/output module, 8 DI, 24 V DC, 8 DO relays, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DX531 - Digital input/output module

- 8 digital inputs 120/230 V AC
- 4 relay outputs with one change-over contacts each
- Module-wise galvanically isolated



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the digital inputs (I0 - I7)
4 4 yellow LEDs to display the signal states at the digital relay outputs (R0 - R3)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input / output unit.

- 8 digital inputs 120/230 V AC in 1 group (2.0...2.3 and 3.0...3.3)
- 4 digital relay outputs with one change-over contact each (R0...R3). All output channels are galvanically isolated from each other.

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)
Required terminal units	TU531 or TU532 ↬ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*

The device is plugged on a terminal unit ↬ *Chapter 1.6.3.5.4 “TU531 and TU532 for I/O modules” on page 2562*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

Connections

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

- Terminals 1.8 to 4.8: process supply voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process supply voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	unused	
2.0 and 3.0	I0 and I1	Input signals for the digital inputs I0 and I1
4.0	N01	Neutral conductor for the digital inputs I0 and I1
2.1 and 3.1	I2 and I3	Input signals for the digital inputs I2 and I3
4.1	N23	Neutral conductor for the digital inputs I2 and I3
2.2 and 3.2	I4 and I5	Input signals for the digital inputs I4 and I5
4.2	N45	Neutral conductor for the digital inputs I4 and I5
2.3 and 3.3	I6 and I7	Input signals for the digital inputs I6 and I7
4.3	N67	Neutral conductor for the digital inputs I6 and I7
2.4	R0	Common contact of the first relay output
3.4 and 4.4	NO0 and NC0	NO and NC contacts of the first relay output
2.5	R1	Common contact of the second relay output
3.5 and 4.5	NO1 and NC1	NO and NC contacts of the second relay output
2.6	R2	Common contact of the third relay output
3.6 and 4.6	NO2 and NC2	NO and NC contacts of the third relay output
2.7	R3	Common contact of the fourth relay output
3.7 and 4.7	NO3 and NC3	NO and NC contacts of the fourth relay output

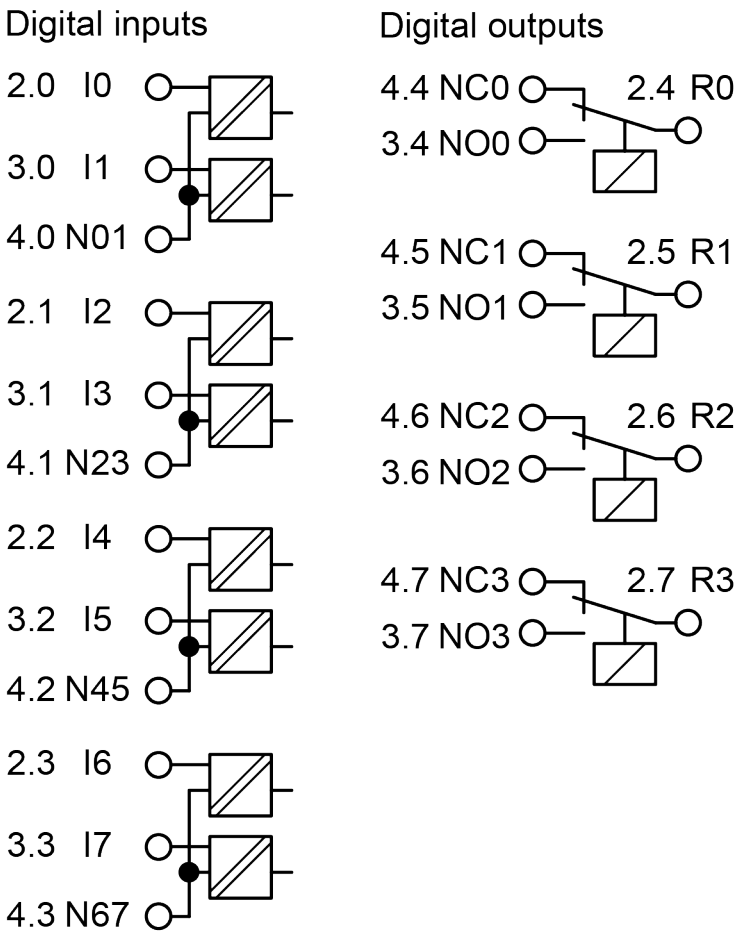


Fig. 121: Internal construction

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DX531. The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

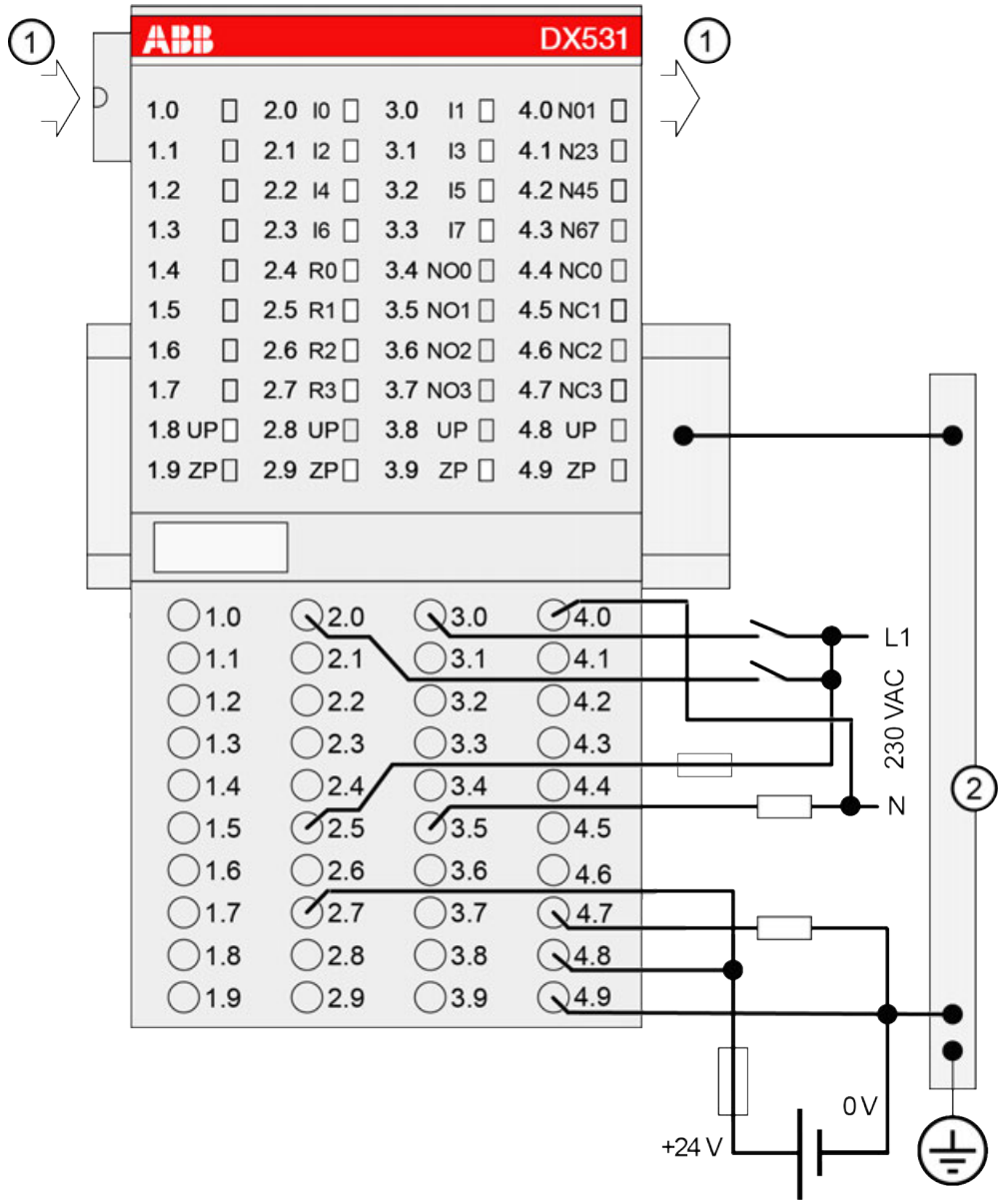
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the module:



1 I/O bus
2 Switchgear cabinet earth

NOTICE!

- If the relay outputs have to switch inductive **DC loads**, free-wheeling diodes must be circuited in parallel to these loads.
- If the relay outputs have to switch inductive **AC loads**, spark suppressors are required.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

NOTICE!
Risk of damaging the PLC module!
The following has to be considered when connecting input and output voltages to the module:

- All 230 V AC feeds must be single phase from the same supply system.
- Connection of 2 or more relay contacts in series is possible; however, voltages above 230 V AC and 3-phase loads are not allowed.
- The 4 change-over contacts of the relays are galvanically isolated from channel to channel. This allows to connect loads of 24 V DC and 230 V AC to relay outputs of the same module. In such cases it is necessary that both supply voltages are grounded to prevent unsafe floating grounds.
- All input signals must come from the same phase of the same supply system (together with the used neutral conductor). The module is designed for 120/230 V AC max., not for 400 V AC, not even between two input terminals.
- All neutral conductor connections must be common to the same supply system, since the terminals 4.0 to 4.3 are interconnected within the module. Otherwise, accidental energization could occur.

NOTICE!
Risk of damaging the PLC module!
There is no internal short-circuit or overload protection for the relay outputs.

Protect the relay contacts by back-up fuses of 6 A max. (characteristic gG/gL). Depending on the application, fuses can be used for single channels or module-wise.

The module provides several diagnosis functions (see chapter Diagnosis and State LEDs ↳ *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 4065*).

Internal data exchange

Digital inputs (bytes)	1
Digital outputs (bytes)	1
Counter input data (words)	0
Counter output data (words)	0

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

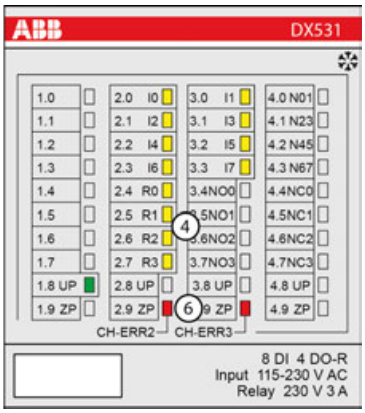
Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1205 [1])	Word	1205 0x04B5	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			not for FBP
Parameter length	Internal	4	Byte	4-CPU 4-FBP	0	255	0x0Y02
Check supply	Off on	0 1	Byte	On 0x01	0	1	0x0Y03
Input delay	20 ms 100 ms	0 1	Byte	20 ms 0x00	0	1	0x0Y04
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
Substitute value at outputs Bit 3 = Output 3 Bit 0 = Output 0	0...15	0... 0x0f	Byte	0 0x00	0	15	0x0Y06
[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1 [2]) Not with FBP							

GSD file:

Ext_User_Prm_Data_Len =	7
Ext_User_Prm_Data_Const (0) =	0x04, 0xb6, 0x04, \ 0x01, 0x00, 0x00, 0x00;

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7	Digital input	Yellow	Input = OFF	Input = ON	--
	Outputs R0...R3 (relays)	Digital output	Yellow	Relay output = OFF	Relay output = ON	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR2	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 2 and 3)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group
	CH-ERR3		Red			
	CH-ERR *)	Module Error	Red	--	Internal error	--
	*) All of the LEDs CH-ERR2 to CH-ERR3 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V DC (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V DC (ZP)
	Rated value	24 V DC

Parameter		Value
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + output loads
Inrush current from UP (at power up)		0.004 A^2s
Max. power dissipation within the module		6 W (outputs OFF)
Weight (without terminal unit)		Ca. 300 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	4 groups of 2 channels each
Terminals of the channels I0 to I7	*Chapter 1.6.3.6.1.2.8.3 “Connections” on page 2767*
Galvanic isolation	2500 V AC from the rest of the module (I/O bus)
Indication of the input signals	1 yellow LED per channel The LEDs are only operating if the module is initialized
Monitoring point of input indicator	LED is controlled by process CPU
Input type acc. to EN 61131-2	Type 2
Input delay (0->1 or 1->0)	Typ. 20 ms
Input signal voltage	230 V AC or 120 V AC

Parameter		Value
Input signal range		0 V AC...265 V AC
Input signal frequency		47 Hz...63 Hz
Input characteristic		According EN 61132-2 Type 2
Signal 0		0 V AC...40 V AC
Undefined signal		> 40 V AC...< 74 V AC
Signal 1		74 V AC...265 V AC
Input current per channel		
	Input voltage = 159 V AC	> 7 mA
	Input voltage = 40 V AC	< 5 mA
Overvoltage protection		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the relay outputs

Parameter		Value
Number of channels per module		4 relay outputs
Distribution of channels into groups		4 groups of 1 channel each
Connection of the four relays		*Chapter 1.6.3.6.1.2.8.3 “Connections” on page 2767*
Galvanic isolation		Between the channels and from the rest of the module
Indication of the output signals		1 yellow LED per channel, the LED is ON when the relay coil is energized
Monitoring point of output indicator		LED is controlled by process CPU
Way of operation		Non-latching type
Output delay (0->1 or 1->0)		On request
Relay power supply		By UP process supply voltage
Relay outputs		
	Output short circuit protection	Must be provided externally with a fuse or circuit breaker
	Rated protection fuse	6 A gL/gG per channel
Output switching capacity		
	Resistive load, max.	3 A; 3 A (230 V AC), 2 A (24 V DC)
	Inductive load, max.	1.5 A; 1.5 A (230 V AC), 1.5 A (24 V DC)
	Lamp load	60 W (230 V AC), 10 W (24 V DC)
Lifetime (cycles)		Mechanical: 300 000; Under load: 300 000 (24 V DC at 2 A), 200 000 (120 V AC at 2 A), 100 000 (230 V AC at 3 A)
Spark suppression with inductive AC load		Must be performed externally according to driven load specifications

Parameter		Value
Demagnetization with inductive DC load		A free-wheeling diode must be circuited in parallel to the inductive load
Switching frequency		
	With resistive load	Max. 10 Hz
	With inductive load	Max. 2 Hz
	With lamp load	On request
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 245 000 R0001	DX531, digital input/output module, 8 DI, 230 V AC, 4 DO relays, 2-wires	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Fast counter

More information can be found in the Automation Builder chapter, *“Fast counters in AC500 devices”*.

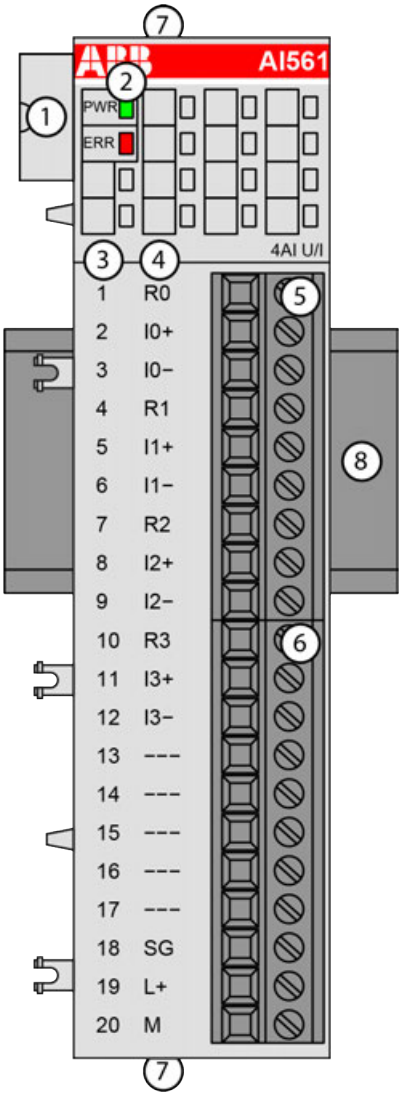
↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

1.6.3.6.2 Analog I/O modules

S500-eCo

AI561 - Analog input module

- 4 configurable analog inputs (I0 to I3) in 1 group
- Resolution: 11 bits plus sign or 12 bits



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are not galvanically isolated from each other.

All other circuitry of the module is not galvanically isolated from the inputs or from the I/O bus.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

4 analog inputs, individually configurable for

- Not used (default setting)
- -2.5 V...+2.5 V
- -5 V...+5 V
- 0 V...+5 V
- 0 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

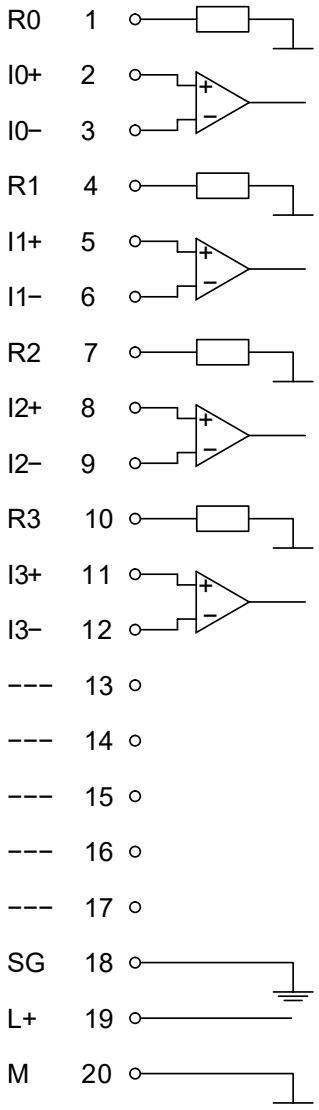
Parameter		Value
Resolution of the analog channels		
	Voltage bipolar (-2.5 V...+2.5 V; -5 V...+5 V)	11 bits plus sign
	Voltage unipolar (0 V...5 V; 0 V...10 V)	12 bits
	Current (0 mA...20 mA; 4 mA...20 mA)	12 bits
LED displays		2 LEDs for process voltage and error messages
Internal supply		Via I/O bus
External supply		Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the analog inputs:



The assignment of the terminals:

Terminal	Signal	Description
1	R0	Burden resistor for input signal 0 for current sensing
2	I0+	Positive pole of input signal 0
3	I0-	Negative pole of input signal 0
4	R1	Burden resistor for input signal 1 for current sensing
5	I1+	Positive pole of input signal 1
6	I1-	Negative pole of input signal 1
7	R2	Burden resistor for input signal 2 for current sensing
8	I2+	Positive pole of input signal 2
9	I2-	Negative pole of input signal 2
10	R3	Burden resistor for input signal 3 for current sensing
11	I3+	Positive pole of input signal 3

Terminal	Signal	Description
12	I3-	Negative pole of input signal 3
13	---	Reserved
14	---	Reserved
15	---	Reserved
16	---	Reserved
17	---	Reserved
18	SG	Shield grounding
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per AI561.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is interconnected to the M/ZP terminal of the CPU/communication interface module.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

NOTICE!

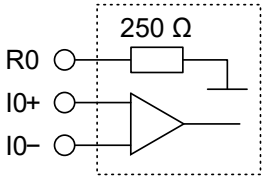
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

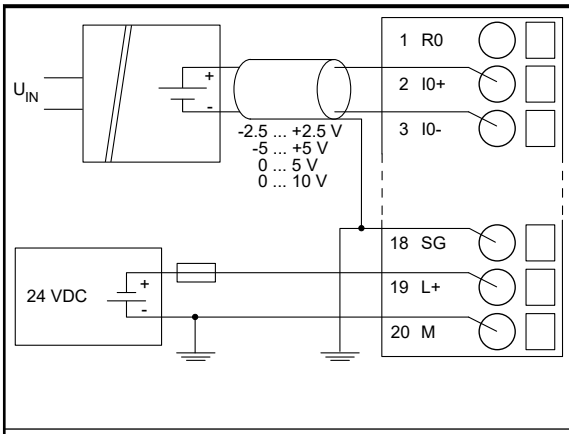
The module provides several diagnosis functions *Chapter 1.6.3.6.2.1.1.6 “Diagnosis” on page 2783*.

The following figure is an example of the internal construction of the analog input AI0. The analog inputs AI1...AI3 are designed in the same way.

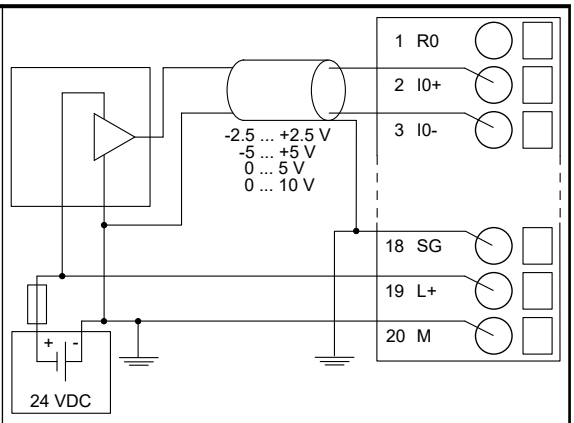


CAUTION!
Risk of damaging the analog input!
The 250 Ω input resistor can be damaged by overcurrent.
Make sure that the current through the resistor never exceeds 30 mA.

The following figures are an example of the connection of analog sensors (voltage) to the input I0 of the analog input module AI561. Proceed with the inputs I1 to I3 in the same way.

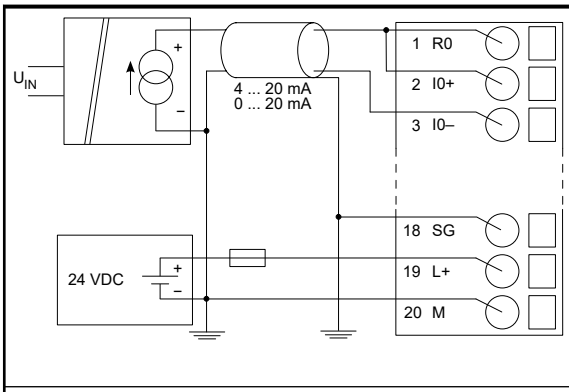


Connection of active-type analog sensors (voltage)

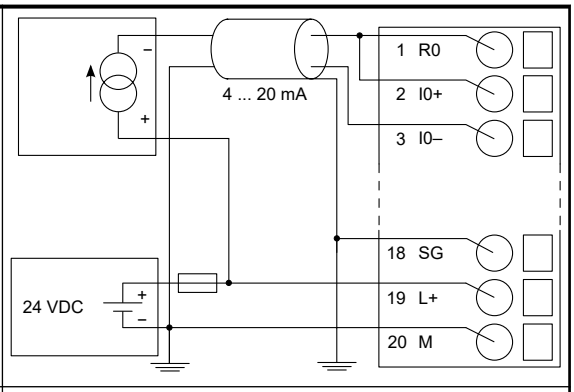


Connection of passive-type analog sensors (voltage)

The following figures are an example of the connection of analog sensors (current) to the input I0 of the analog input module AI561. Proceed with the inputs I1 to I3 in the same way.



Connection of active-type analog sensors (current)



Connection of passive-type analog sensors (current)

The meaning of the LEDs is described in the Displays section *Chapter 1.6.3.6.2.1.1.7 “State LEDs” on page 2784*.

I/O configuration

The analog input module AI561 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6500 [1])	WORD	0x1964	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Internal	6	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0 ... 7), LowByte is index (1...n)

GSD file:	Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	0x09 0x65, 0x19, 0x06, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Input channel (4x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00	0	65535

Table 481: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default)
1	0 V...10 V
3	0 mA...20 mA
4	4 mA...20 mA
6	0 V...5 V

Internal value	Operating modes for the analog inputs, individually configurable
7	-5 V...+5 V
20	-2,5 V...+2,5 V

Diagnosis

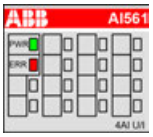
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<− Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	Interface	Device	Module	Channel	Error Identifier	Error message		Remedy
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	1	0...3	48	Analog value overflow at an analog input		Check input value or terminal
	11 / 12	ADR	1...0					
4	14	1...10	1	0...3	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...0					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI); COM1/COM2: 1...10 = expansion 1..10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Measuring ranges

Risk of invalid analog input values!

The analog input values may be invalid if the measuring range of the inputs is exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

Range	-2.5 ... +2.5 V	-5 ... +5 V	0 ... 5 V	0 ... 10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
							Decimal	Hex.
Overflow	>2.9397	>5.8795	>5.8795	>11.7589	>23.5178	>22.8142	32767	7FFF
Measured value too high	2.9397 : 2.5014	5.8795 : 5.0029	5.8795 : : : 5.0015	11.7589 : : : 10.0029	23.5178 : : : 20.0058	22.8142 : : : 20.0058	32511 : 27664 27658 27656	7EFF : 6C10 6C0A 6C08
Normal range	2.5000 : 0.0014	5.0000 : 0.0029	5.0000 : : : 0.0015	10.0000 : : : 0.0029	20.0000 : : : 0.0058	20.0000 : : : 4.0058	27648 : 16 10 8	6C00 : 0010 000A 0008

Range	-2.5 ... +2.5 V	-5 ... +5 V	0 ... 5 V	0 ... 10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
							Decimal	Hex.
Normal range or measured value too low	0.0000	0.0000	0.0000	0.0000	0	4	0	0000
	:	:				3.9942	-10	FFF6
	-0.0014	-0.0029				:	-16	FFF0
	:	:				:	-4864	ED00
	:	:				0	-6912	E500
	:	:					:	:
	-2.5000	-5.0000					-27648	9400
Measured value too low	-2.5014	-5.0029					-27664	93F0
	:	:					:	:
	-2.9398	-5.8795					-32512	8100
Underflow	<-2.9398	<-5.8795	<-0.0300	<-0.0600	<-0.1200	<-0.1200	-32768	8000

The represented resolution corresponds to 12 bits respectively 11 bits plus sign.

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 19 for L+ (+24 V DC) and terminal 20 for M (0 V)
	Rated value	24 V DC
	Current consumption via L+ terminal	0.1 A
	Inrush current (at power up)	0.05 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Protection fuse for L+	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		No
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.7 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value	
Number of channels per module		4 individually configurable voltage or current inputs	
Distribution of channels into groups		1 (4 channels per group)	
Resolution			
	Unipolar	Voltage: 0 V...+5 V; 0 V...+10 V: 12 bits Current 0 mA...20 mA; 4 mA...20 mA: 12 bits	
	Bipolar	Voltage -2.5 V...+2.5 V; -5 V...+5 V: 11 bits plus sign	
Connection of the signals I0- to I3-		Terminals 3, 6, 9, 12	
Connection of the signals I0+ to I3+		Terminals 2, 5, 8, 11	
Input type		Differential	
Galvanic isolation		No galvanic isolation between the inputs and the I/O bus	
Common mode input range		Signal voltage plus common mode voltage must be within ±12 V	
Indication of the input signals		No	
Channel input resistance		Voltage: > 1 MΩ Current: ca. 250 Ω	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale (voltage) ±0.5 % of full scale (current 0 mA...20 mA) ±0.7 % of full scale (current 4 mA...20 mA) at 25 °C	
	Max.	±2 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance	
Time constant of the input filter		Voltage: 300 µs Current: 300 µs	
Relationship between input signal and hex code		↳ *Chapter 1.6.3.6.2.1.1.8 “Measuring ranges” on page 2784*	
Analog to digital conversion time		Typ. 500 µs per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		8 bytes	
Overvoltage protection		Yes, up to 30 V DC only for voltage input	

Parameter		Value
Max. cable length (conductor cross section > 0,14 mm²)		
	Unshielded wire	10 m
	Shielded wire	100 m

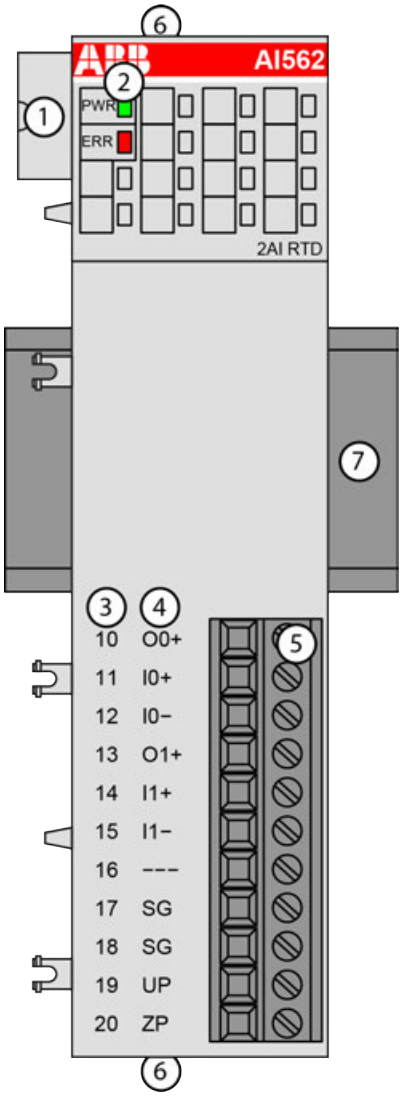
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1101	AI561, analog input module, 4 AI, U/I	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AI562 - Analog input module

- 2 configurable analog resistance temperature detector (RTD) inputs (I0 and I1) in 1 group
- Resolution: 15 bits plus sign



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are not galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

2 analog RTD-inputs, individually configurable for

- Not used (default)
- Pt100, -50 °C...+400 °C, 2-wire
- Pt100, -50 °C...+400 °C, 3-wire
- Pt1000, -50 °C...+400 °C, 2-wire
- Pt1000, -50 °C...+400 °C, 3-wire
- Ni1000, -50 °C...+150 °C, 2-wire
- Ni1000, -50 °C...+150 °C, 3-wire
- Ni100, -50 °C...+150 °C, 2-wire
- Ni100, -50 °C...+150 °C, 3-wire
- Analog input resistance 0 Ω...150 Ω
- Analog input resistance 0 Ω...300 Ω

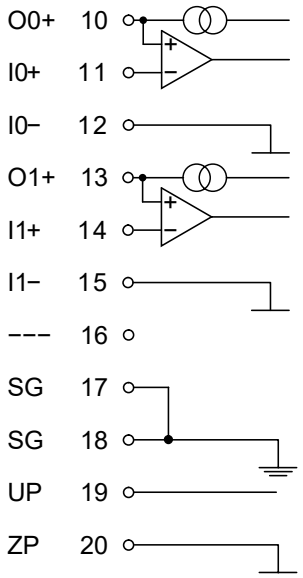
Parameter	Value
Resolution of the analog channels	
Temperature	0.1 °C
LED displays	2 LEDs for process voltage and error messages
Internal supply	Via I/O bus
External supply	Via the terminals UP (process voltage 24 V DC) and ZP (0 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↯ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the analog inputs:



The assignment of the terminals:

Terminal	Signal	Description
10	O0+	Current source of channel 0
11	I0+	Sense input of channel 0
12	I0-	Return input of channel 0
13	O1+	Current source of channel 1
14	I1+	Sense input of channel 1
15	I1-	Return input of channel 1
16	---	Reserved
17	SG	Shield grounding
18	SG	Shield grounding
19	UP	Process voltage UP (24 V DC)
20	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per AI562.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove or replace a module.

NOTICE!

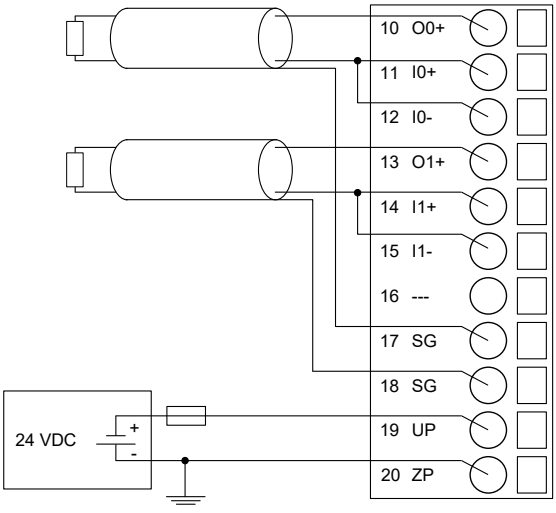
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

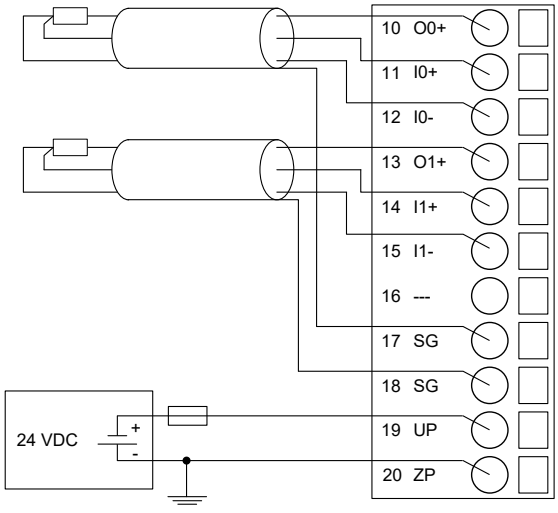
- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.3.6.2.1.2.6 “Diagnosis” on page 2793*.

The following figures show the connection of RTDs to the inputs of the analog input module AI562.



2-wires input



3-wires input

With 2-wires connection, the resistance of the connection wires influences the accuracy of the measured value. Use 3-wires connection to achieve the guaranteed measuring accuracy.

The meaning of the LEDs is described in the Displays section *Chapter 1.6.3.6.2.1.2.7 “State LEDs” on page 2794*.

I/O configuration

The analog input module AI562 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6505 [1])	WORD	0x1969	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Intern	4	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x07
Ext_User_Prm_Data_Const(0) =	0x6A, 0x19, 0x04, \ 0x01, 0x00, \ 0x00, 0x00;

Input channel (2x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [3])	0	65535

Table 482: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default) [3])
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C
16	2-wire Pt1000, -50 °C...+400 °C

Internal value	Operating modes for the analog inputs, individually configurable
17	3-wire Pt1000, -50 °C...+400 °C
18	2-wire Ni1000 -50 °C...+150 °C
19	3-wire Ni1000 -50 °C...+150 °C
22	2-wire Ni100, -50 °C...+150 °C
23	3-wire Ni100, -50 °C...+150 °C
32	Analog input resistor 0 Ω...150 Ω
33	Analog input resistor 0 Ω...300 Ω

Diagnosis

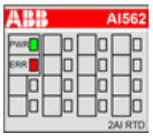
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...1	48	Analog value overflow at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...1	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Measuring ranges

Risk of invalid analog input values!

The analog input values may be invalid if the measuring range of the inputs is exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

Resistance temperature detectors

Range	Pt100 / Pt1000 -50 ... +400 °C	Ni1000 / Ni100 -50 ... +150 °C	Digital value	
			Decimal	Hex.
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1

Range	Pt100 / Pt1000 -50 ... +400 °C	Ni1000 / Ni100 -50 ... +150 °C	Digital value	
			Decimal	**Hex.**
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
Normal range	400.0 °C : : : : 0.1 °C	 150.0 °C : : 0.1 °C	4000 2000 1500 700 : 1	0FA0 07D0 05DC 02BC : 1
	0,0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500 -2000	FFFF : FE0C F830
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Resistances

Range	Resistance 0 ... 150 Ω	Resistance 0 ... 300 Ω	Digital value	
			Decimal	**Hex.**
Overflow	>176.383	>352.767	32767	7FFF
Measured value too high	176.383 150.005	352.767 300.011	32511 27649	7EFF 6C01
Normal range	150.000 : 0.005	300.000 : 0.011	27648 : 1	6C00 : 0001
	0	0	0	0000

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption	0.04 A
	Inrush current (at power-up)	0.05 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Protection fuse for UP	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the input group and the rest of the module
	Isolated groups	1 (2 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		1.1 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value
Number of channels per module		2 configurable RTD (resistance temperature detector) inputs
Distribution of channels into groups		1 (2 channels per group)
Resolution		
	RTD	0.1 °C / 0.1 °F
	Resistance	15 bits + sign
Connection of the signals O0+ and O1+		Terminals 10 and 13
Connection of the signals I0- and I1-		Terminals 11 and 14
Connection of the signals I0+ and I1+		Terminals 12 and 15
Input type		Module ground referenced RTD for 2-wire and 3-wire resistance temperature detectors

Parameter		Value	
Galvanic isolation		Against internal power supply and other modules	
Input ranges		Pt100, Pt1000, Ni100, Ni1000	
		150 Ω, 300 Ω	
Indication of the input signals		No	
Module update time		All channels: < 1 s	
Channel input resistance		> 100 kΩ	
Input filter attenuation		-3 dB at 3.6 kHz	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	Depending on RTD max. ±0.6 % of full scale (guaranteed for 3-wires connection only) at 25 °C
		Max.	±2 % of full scale (guaranteed for 3-wires connection only) at 0 °C...60 °C or EMC disturbances
Measuring range		*Chapter 1.6.3.6.2.1.2.8 “Measuring ranges” on page 2794*	
Analog to digital conversion time		Typ. 140 ms per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		4 bytes	
Power dissipation inside the sensor (max.)		1 mW	
Suppression of interference		On request	
Maximum input voltage		30 V DC (sense), 5 V DC (source)	
Basic error (resistance)		0.1 % of full-scale	
Repeatability		0.05 % of full-scale	
Overvoltage protection		Yes, up to 30 V DC	
Wire loop resistance		< 20 Ω	
Max. cable length (conductor cross section > 0.14 mm^2)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Ordering data

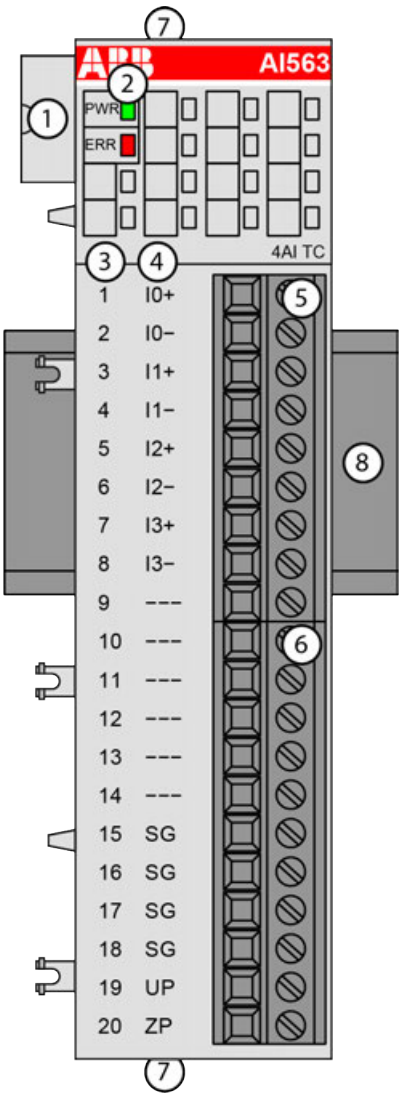
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1102	AI562, analog input module, 2 AI, RTD	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AI563 - Analog input module

- 4 configurable thermocouple (TC) / -80 mV...+80 mV inputs (I0 to I3) in 1 group
- Resolution: 15 bits plus sign



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number

4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

The other electronic circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

4 analog TC inputs, individually configurable for

- Not used (default)
- Voltage -80 mV ... + 80 mV
- Thermocouple J-type -210 °C...+1200 °C
- Thermocouple K-type -270 °C...+1372 °C
- Thermocouple R-type -50 °C...+1768 °C
- Thermocouple S-type -50 °C...+1768 °C
- Thermocouple T-type -270 °C...+400 °C
- Thermocouple E-type -270 °C...+1000 °C
- Thermocouple N-type -270 °C...+1300 °C

Parameter	Value
Resolution of the analog channels	
Temperature	0.1 °C
LED displays	2 LEDs for process voltage and error messages
Internal supply	Via I/O bus
External supply	Via the terminals UP (process voltage 24 V DC) and ZP (0 V DC)

Connections

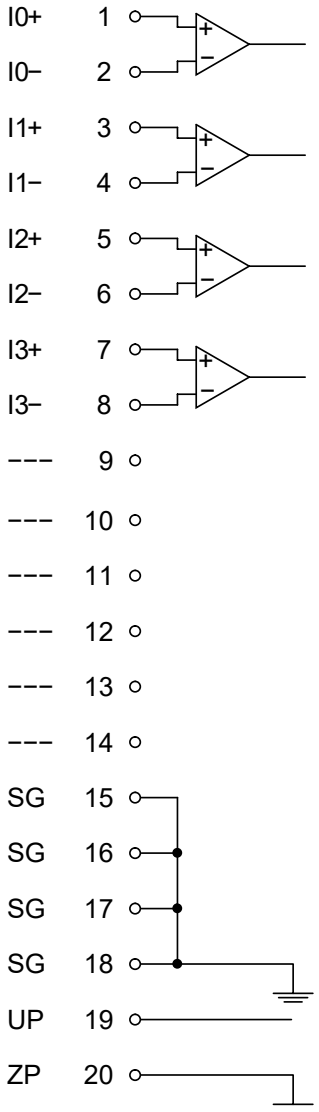
For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

After powering up the system, input channels, which are configured will have undefined values /diagnosis message for typically 45 seconds, if the wires of all configured channels are broken.

If the AI563 is connected to a PROFINET communication interface module, the firmware version of PROFINET communication interface module must be 1.2 or above.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the analog inputs:



The assignment of the terminals:

Terminal	Signal	Description
1	I0+	Positive pole of channel 0
2	I0-	Negative pole of channel 0

Terminal	Signal	Description
3	I1+	Positive pole of channel 1
4	I1-	Negative pole of channel 1
5	I2+	Positive pole of channel 2
6	I2-	Negative pole of channel 2
7	I3+	Positive pole of channel 3
8	I3-	Negative pole of channel 3
9	---	Reserved
10	---	Reserved
11	---	Reserved
12	---	Reserved
13	---	Reserved
14	---	Reserved
15	SG	Shield grounding
16	SG	Shield grounding
17	SG	Shield grounding
18	SG	Shield grounding
19	UP	Process voltage UP (24 V DC)
20	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module increases by 5 mA per AI563.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

NOTICE!

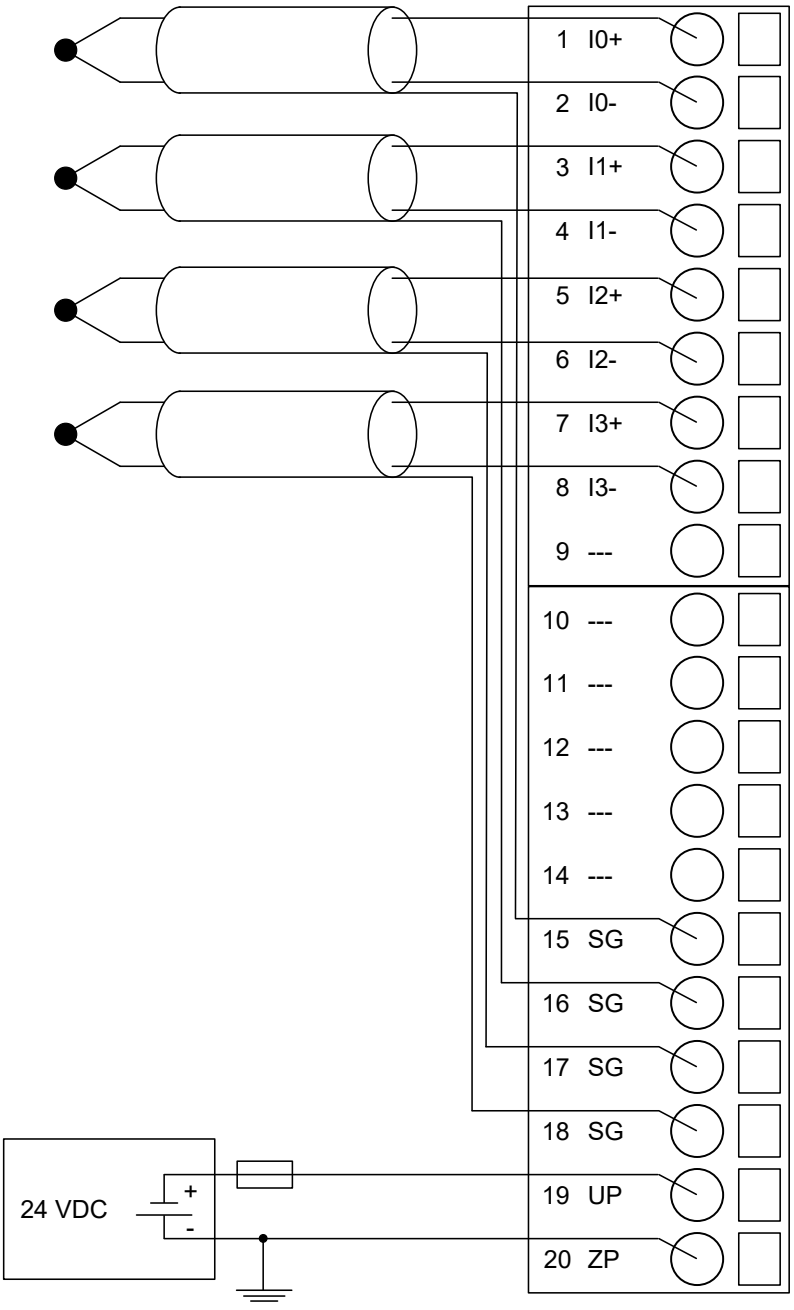
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.3.6.2.1.3.6 “Diagnosis” on page 2804*.

The following figure shows the connection of thermocouples to the inputs of the module:



The meaning of the LEDs is described in Displays *Chapter 1.6.3.6.2.1.3.7 “State LEDs” on page 2805* chapter.

I/O configuration

The analog input module AI563 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6510 [1])	WORD	0x196E	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Intern	6	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	
[1]) with CS31 and addresses less than 70, the value is increased by 1 [2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)							

GSD file:

Ext_User_Prm_Data_Len =	0x09
Ext_User_Prm_Data_Const(0) =	0x6F, 0x19, 0x06, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Input channel (4x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 483: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default)
21	Voltage -80 mV...+80 mV
24	Thermocouple J-type -210 °C...+1200 °C
25	Thermocouple K-type -270 °C...+1372 °C
26	Thermocouple R-type -50 °C...+1768 °C
27	Thermocouple S-type -50 °C...+1768 °C
28	Thermocouple T-type -270 °C...+400 °C
29	Thermocouple E-type -270 °C...+1000 °C
30	Thermocouple N-type -270 °C...+1300 °C

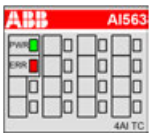
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...3	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...3	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Measuring ranges

AI563 needs typ. 6 to 8 seconds for initialization after applying the process supply voltage to clamp UP/ZP. During this time, the accuracy of the measurement values is not within specification. After that, valid measurement values are provided by the module. After that, valid measurement values are provided by the module.

After an interruption of the process supply voltage > 10 ms, a re-initialization is performed by AI563.

Risk of invalid analog input values!

The analog input values may be invalid if the measuring range of the inputs is exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

When a wire break occurs on a sensor wire, the temperature measurement value of the corresponding channel changes to Overflow (Hexadecimal 7FFF).

Range	**Type J** **-210 ... +1200 °C**	**Type K** **-270 ... +1372 °C**	**Type N** **-270 ... +1300 °C**	**Type T** **-270 ... +400 °C**	**Digital value**	
					Decimal	**Hex.**
Overflow	> 1200.0 °C	> 1372.0 °C	> 1300.0 °C	> 400.0 °C	32767	7FFF
Normal range					17680	4510
		1372.0 °C			13720	3598
		:	1300.0 °C		13000	32C8
	1200.0 °C	:	:		12000	2EE0
	:	:	:	400.0 °C	4000	0FA0
	:	:	:	:	:	:
	0.1 °C	0.1 °C	0.1 °C	0.1 °C	1	1
	0.0 °C	0.0 °C	0.0 °C		0	0000
	-0.1 °C	-0.1 °C	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:	:
	:	:	:	:	-500	FE0C
	-210.0 °C	:	:	:	-2100	F7CC
		-270.0 °C	-270.0 °C	-270.0 °C	-2700	F574
Underflow	< -210.0 °C	< -270.0 °C	< -270.0 °C	< -270.0 °C	-32768	8000

Range	**-80 mV ... +80 mV**	**Type E** **-270 ... +1000 °C**	**Types R, S** **-50 ... +1768 °C**	**Digital value**	
				Decimal	**Hex.**
Overflow	> +90 mV	> 1000.0 °C	> 1768.0 °C	32767	7FFF
Normal range	+80 mV			27648	6C00
			1768.0 °C	17680	4510
		1000.0 °C		10000	2710
				9000	2328
	:	:	:	:	:
	3 µV	0.1 °C	0.1 °C	1	1
	0 µV	0.0 °C	0.0 °C	0	0000
	-3 µV	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:
	:	:	-50.0 °C	-500	FE0C
	:	-270.0 °C		-2700	F574

Range	-80 mV ... +80 mV	Type E -270 ... +1000 °C	Types R, S -50 ... +1768 °C	Digital value	
				Decimal	Hex.
	-80 mV			-27648	9400
Underflow	< -90 mV	< -270.0 °C	< -50.0 °C	-32768	8000

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption	0.10 A
	Inrush current (at power-up)	0.07 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Not necessary
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the channels and the rest of the module
	Isolated groups	1 (4 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.6 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value	
Number of channels per module		4 configurable thermocouple (TC) inputs	
Distribution of channels into groups		1 (4 channels per group)	
Resolution			
	Temperature	0.1 °C	
	Voltage	15 bits plus sign	
Connection of the signals I0+ to I3+		Terminals 1, 3, 5 and 7	
Connection of the signals I0- to I3-		Terminals 2, 4, 6 and 8	
Input type		Floating thermocouple	
Galvanic isolation		Against internal power supply and other modules	
Common mode rejection		> 120 dB at 120 V AC	
Indication of the input signals		No	
Module update time		All channels: < 1.6 s	
Channel input resistance		On request	
Input filter attenuation		-3 dB at 15 kHz	
Cold junction error		±1.5 °C	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	0.1 % of full-scale (voltage) Depending on thermocouple, see table *Chapter 1.6.3.6.2.1.3.9.1.1 “Accuracy of thermocouple ranges at 25 °C (with cold junction compensation)” on page 2809* at 25 °C
		Max.	±2 % of full scale (T-Type: ±3 % for -240 °C...-270 °C) at 0 °C...60 °C
Relationship between input signal and hex code		*Chapter 1.6.3.6.2.1.3.8 “Measuring ranges” on page 2805*	
Analog to digital conversion time		400 ms per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		8 bytes	
Overvoltage protection		Yes, up to 30 V DC	
Repeatability		On request	
Wire loop resistance		< 100 Ω	
Max. cable length (conductor cross section > 0.14 mm²)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Accuracy of thermocouple ranges at 25 °C (with cold junction compensation)

Thermocouple Type	Range	Accuracy
E	-270 °C...-220 °C	±2 %
	-220 °C...+1000 °C	±0.6 %
J	-210 °C...+1200 °C	±0.6 %
K	-270 °C...-220 °C	±1.5 %
	-220 °C...+1372 °C	±0.6 %
N	-270 °C...-150 °C	±2 %
	-150 °C...+1300 °C	±0.6 %
R	-50 °C...+150 °C	±1.5 %
	+150 °C...+1768 °C	±0.6 %
S	-50 °C...+150 °C	±1.5 %
	+150 °C...+1768 °C	±0.6 %
T	-270 °C...-240 °C	±3 %
	-240 °C...-0 °C	±2 %
	0 °C...+400 °C	±0.6 %

These accuracy values are valid only for stable module temperatures.

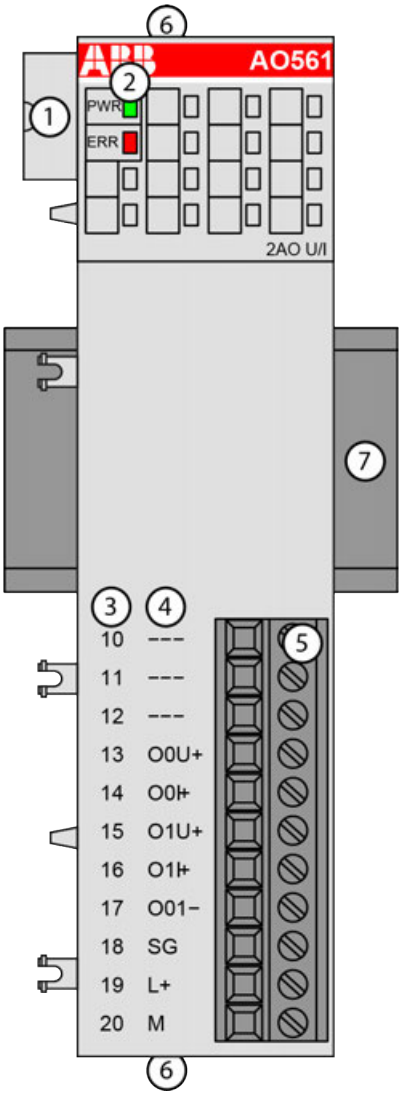
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1103	AI563, analog input module, 4 AI, thermocouple	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AO561 - Analog output module

- 2 configurable analog outputs (O0 and O1) in 1 group
- Resolution: 11 bits plus sign or 12 bit



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are not galvanically isolated from each other.

The other electronic circuitry of the module is not galvanically isolated from the outputs or from the I/O bus.

The I/O module must not be used as communication interface module at CI590-CS31-HA bus modules.

Functionality

2 analog outputs, individually configurable for

- Not used (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage bipolar (-10 V...+10 V)	11 bits plus sign
	Current (0 mA...20 mA; 4 mA...20 mA)	12 bits
LED displays		2 LEDs for process voltage and error messages
Internal supply		Via I/O bus
External supply		Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

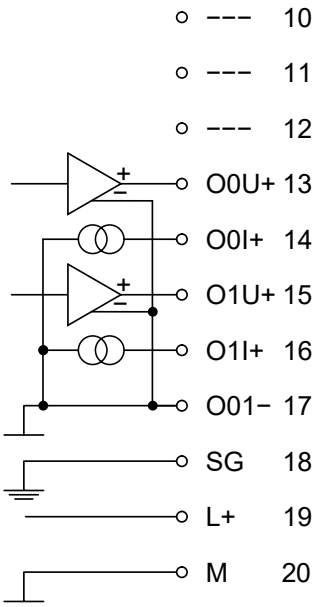
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↪ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

If the output is configured as not used, the voltage and current output signals are undefined and must not be connected.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the analog outputs:



The assignment of the terminals:

Terminal	Signal	Description
10	---	Reserved
11	---	Reserved
12	---	Reserved
13	O0U+	Voltage output of channel 0
14	O0I+	Current output of channel 0
15	O1U+	Voltage output of channel 1
16	O1I+	Current output of channel 1
17	O01-	Negative pole of channels O0 and O1
18	SG	Shield grounding
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module increases by 5 mA per AO561.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is electrically interconnected to the M/ZP terminal of the CPU/ communication interface module.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove or replace a module.

NOTICE!

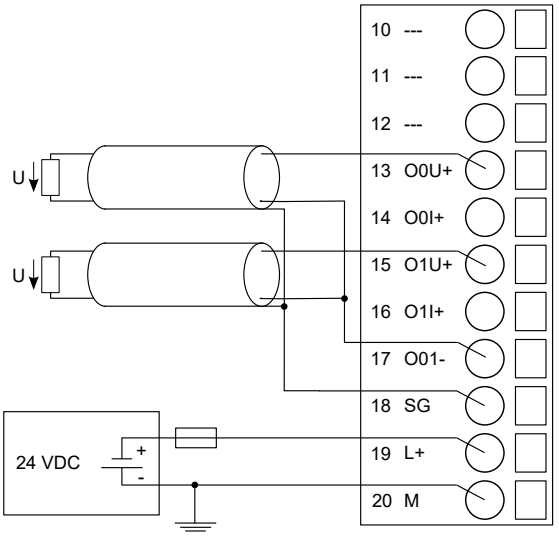
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

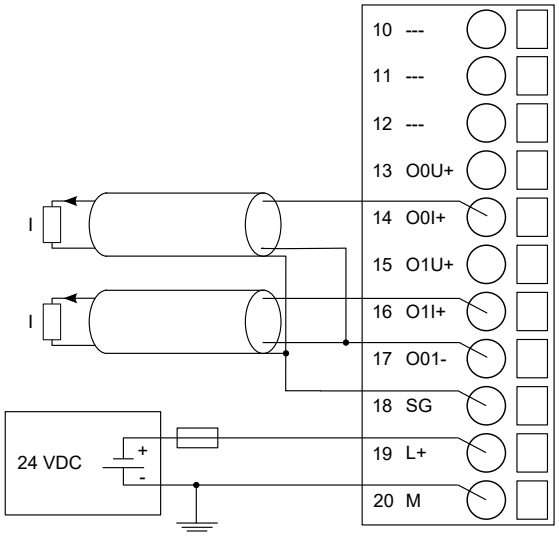
- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.3.6.2.1.4.6 “Diagnosis” on page 2815*.

The following figures show the connection of analog actuators to the analog output module AO561.



Connection of analog voltage actuators



Connection of analog current actuators

The output signal is undefined if the supply voltage at the L+ terminal is below 10 V. This can, for example, occur if the supply voltage has a slow ramp-up / ramp-down behavior and must be foreseen when planning the installation.

If the output is configured in current mode, the voltage output signal is undefined and must not be connected.

If the output is configured in voltage mode, the current output signal is undefined and must not be connected.

I/O configuration

The analog output module AO561 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6515 [1])	WORD	0x1973	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Intern	4	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	
[1]) with CS31 and addresses less than 70, the value is increased by 1 [2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)							

GSD file:

Ext_User_Prm_Data_Len =	0x07
Ext_User_Prm_Data_Const(0) =	0x74, 0x19, 0x04, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Output channel (2x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 484: Channel configuration [2])

Internal value	Operating modes for the analog outputs, individually configurable
0	Not used (default)
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

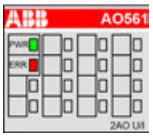
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	3	0...1	48	Analog value overflow at an analog output		Check output value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	3	0...1	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Output ranges

Range	-10 ... +10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Overflow	>11.7589	>23.5178	>22.8142	32767	7FFF
Value too high	11.7589	23.5178	22.8142	32511	7EFF
	:	:	:	:	:
	10.0058	:	:	27664	6C10
	:	:	20.0058	27658	6C0A
	:	20.0058	:	27656	6C08
Normal range	10.0000	20.0000	20.0000	27648	6C00
Normal range or value too low	:	:	:	:	:
	0.0058	:	:	16	0010
	:	:	4.0058	10	000A
	:	0.0058		8	0008
	0.0000	0	4	0	0000

Range	-10 ... +10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
	: -0.0058 : : : -10.0000		3.9942 : : 0	-10 -16 -4864 -6912 : -27648	FFF6 FFF0 ED00 E500 : 9400
Value too low	-10.0058 : -11.7589			-27664 : -32512	93F0 : 8100
Underflow	<-11.7589		<0.0000	-32768	8000

The represented resolution corresponds to 12 bit respectively 11 bit plus sign.

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 19 for L+ (+24 V DC) and terminal 20 for M (0 V)
	Rated value	24 V DC
	Current consumption	0.1 A + output load
	Inrush current (at power-up)	0.05 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Protection fuse for L+	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		No
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		3.1 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog outputs

Parameter		Value	
Number of channels per module		2 configurable voltage or current outputs	
Distribution of channels into groups		1 (2 channels per group)	
Connection of the signals O0U- and O1U+		Terminals 13 and 15	
Connection of the signals O0I+ and O1I+		Terminals 14 and 16	
Output type		Bipolar with voltage, unipolar with current	
Resolution		12 bits or 11 bits plus sign	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.5 % of full scale at 25 °C
		Max.	±2 % of full scale at 0 °C...+60 °C or EMC disturbances
Indication of the output signals		No	
Output Resistance (load) as current output		0 Ω...500 Ω	
Output load ability as voltage output		±2 mA max.	
Output data length		4 bytes	
Relationship between output signal and hex code		*Chapter 1.6.3.6.2.1.4.8 “Output ranges” on page 2816*	
Unused outputs		Must not be connected and must be configured as "unused"	
Overvoltage protection		Yes, up to 30 V DC	
Max. cable length (conductor cross section > 0.14 mm^2)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Ordering data

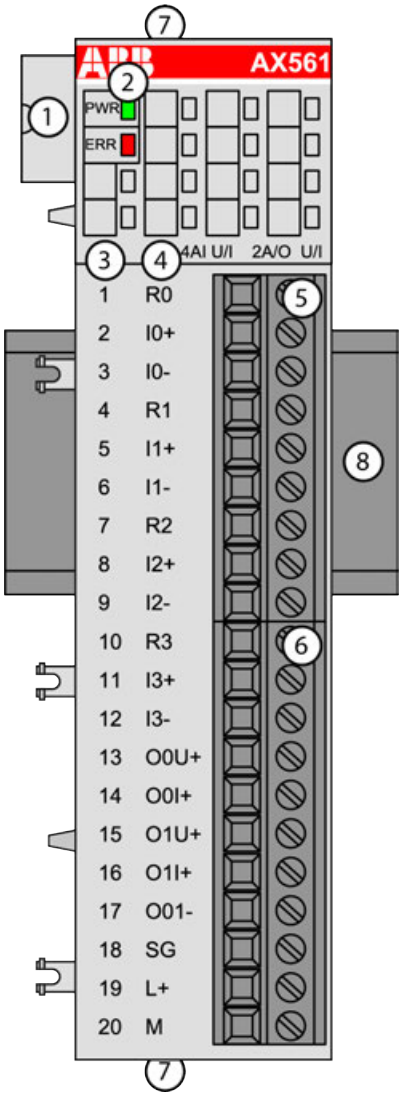
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1201	AO561, analog output module, 2 AO, U/I	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AX561 - Analog input/output module

- 4 configurable analog inputs (I0 to I3) in 1 group
- 2 configurable analog outputs (O0 and O1) in 1 group
- Resolution: 11 bits plus sign or 12 bits



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error

3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are not galvanically isolated from each other.

The outputs are not galvanically isolated from each other.

All other circuitry of the module is not galvanically isolated from the inputs/outputs or from the I/O bus.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

4 analog inputs, individually configurable for

- Not used (default)
- -2.5 V...+2.5 V
- -5 V...+ 5 V
- 0 V...+5 V
- 0 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

2 analog outputs, individually configurable for

- Not used (default)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage bipolar (-2.5 V...+2.5 V; -5 V...+5 V)	11 bits plus sign
	Voltage unipolar (0 V...5 V; 0 V...10 V)	12 bits
	Current (0 mA...20 mA; 4 mA...20 mA)	12 bits
LED displays		2 LEDs for process voltage and error messages

Parameter	Value
Internal supply	Via I/O bus
External supply	Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

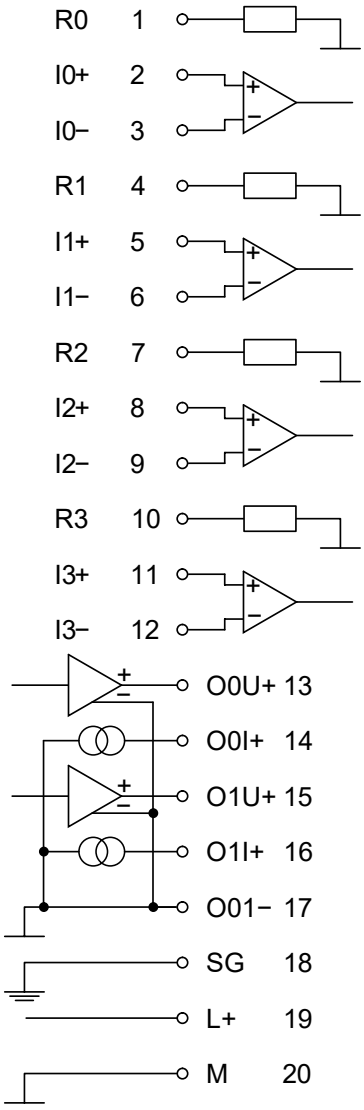
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.4.5 “AC500-eCo” on page 3352.

If the output is configured as not used, the voltage and current output signals are undefined and must not be connected.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the analog inputs and outputs:



The assignment of the terminals:

Terminal	Signal	Description
1	R0	Burden resistor for input signal 0 for current sensing
2	I0+	Positive pole of input signal 0
3	I0-	Negative pole of input signal 0
4	R1	Burden resistor for input signal 1 for current sensing
5	I1+	Positive pole of input signal 1
6	I1-	Negative pole of input signal 1
7	R2	Burden resistor for input signal 2 for current sensing
8	I2+	Positive pole of input signal 2
9	I2-	Negative pole of input signal 2
10	R3	Burden resistor for input signal 3 for current sensing
11	I3+	Positive pole of input signal 3
12	I3-	Negative pole of input signal 3
13	O0U+	Voltage output of channel 0
14	O0I+	Current output of channel 0

Terminal	Signal	Description
15	O1U+	Voltage output of channel 1
16	O1I+	Current output of channel 1
17	O01-	Negative pole of channels O0 and O1
18	SG	Shield grounding
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module increases by 5 mA per AX561.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is interconnected to the M/ZP terminal of the CPU/communication interface module.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

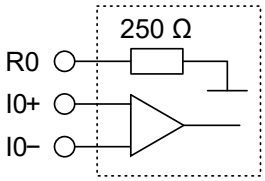
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.3.6.2.1.5.6 “Diagnosis” on page 2827*.

The following figure is an example of the internal construction of the analog input AI0. The analog inputs AI1...AI3 are designed in the same way.



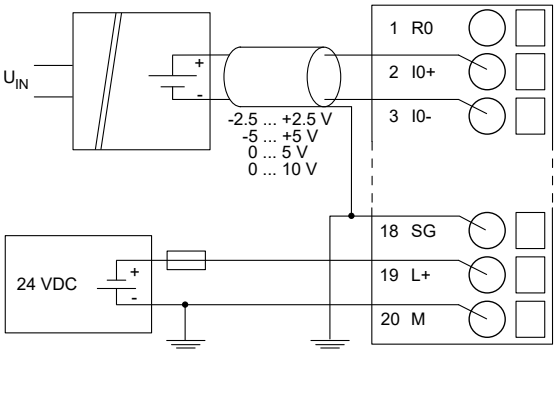
CAUTION!

Risk of damaging the analog input!

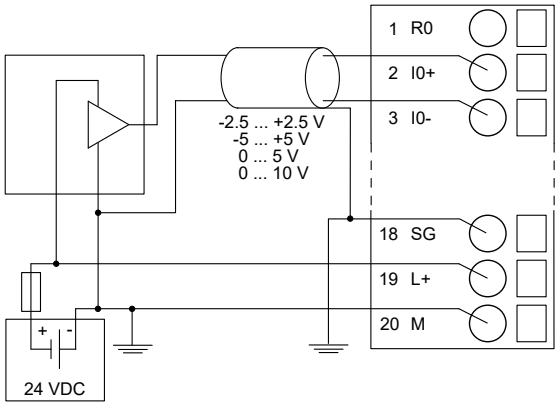
The 250 Ω input resistor can be damaged by overcurrent.

Make sure that the current through the resistor never exceeds 30 mA.

The following figures are an example of the connection of analog sensors (voltage) to the input I0 of the analog input/output module AX561. Proceed with the inputs I1 to I3 in the same way.

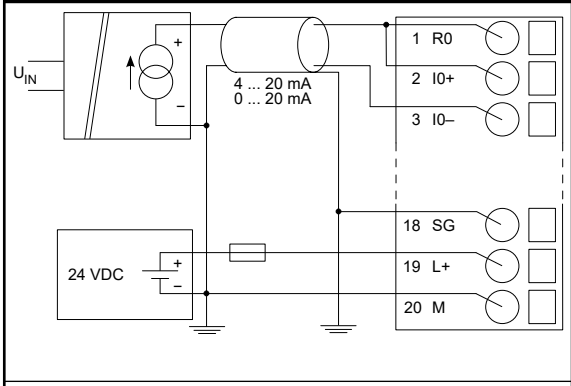


Connection of active-type analog sensors (voltage)

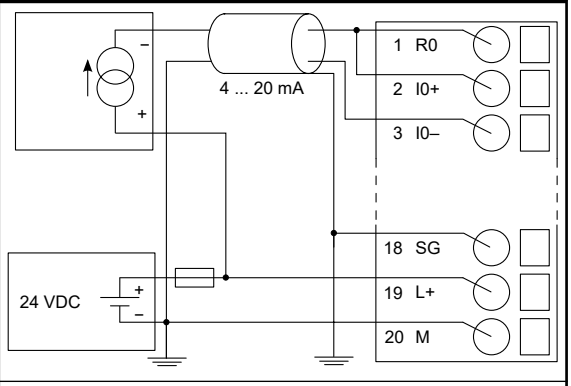


Connection of passive-type analog sensors (voltage)

The following figures are an example of the connection of analog sensors (current) to the input I0 of the analog input/output module AX561. Proceed with the inputs I1 to I3 in the same way.

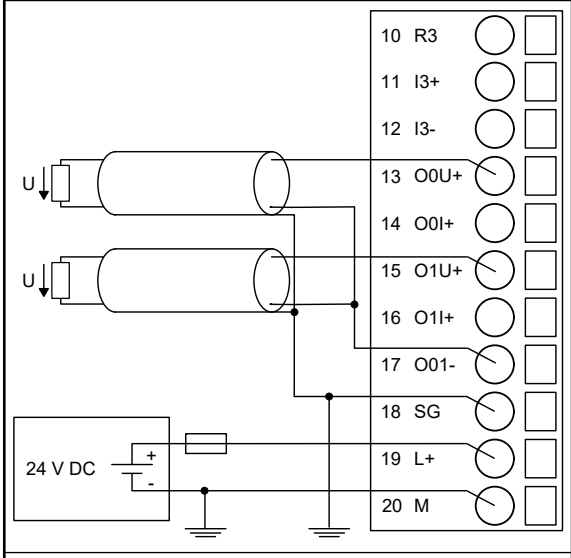


Connection of active-type analog sensors (current)

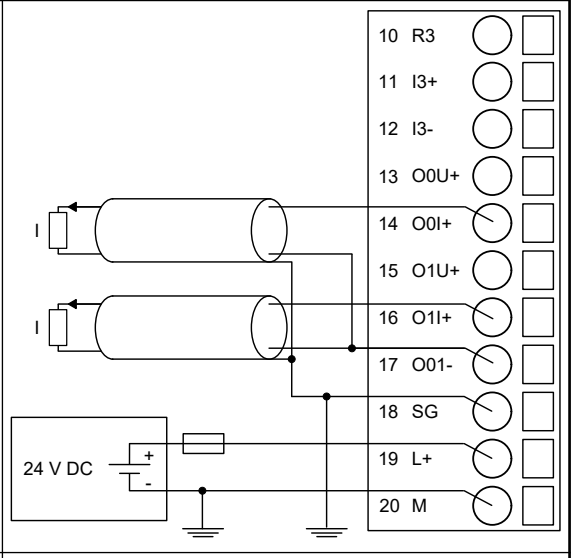


Connection of passive-type analog sensors (current)

The following figures are an example of the connection of analog actuators to the analog input/ output module AX561.



Connection of analog voltage actuators



Connection of analog current actuators

The output signal is undefined if the supply voltage at the L+ terminal is below 10 V. This can, for example, occur if the supply voltage has a slow ramp-up / ramp-down behavior and must be foreseen when planning the installation.

If the output is configured in current mode, the voltage output signal is undefined and must not be connected.

If the output is configured in voltage mode, the current output signal is undefined and must not be connected.

The meaning of the LEDs is described in the displays chapter *Chapter 1.6.3.6.2.1.5.7 “State LEDs” on page 2828*.

I/O configuration

The I/O module does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6520 [1])	WORD	0x1978	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Internal	8	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00			

[1]) With CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x0B
Ext_User_Prm_Data_Const(0) =	0x79, 0x19, 0x08, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00;

Input channel (4x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 485: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default)
1	0 V...+10 V
3	0 mA...20 mA
4	4 mA...20 mA
6	0 V...+5 V
7	-5 V...+5 V
20	-2.5 V...+2.5 V

Output channel (2x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see see table [2])	see see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 486: Channel configuration [2])

Internal value	Operating modes for the analog outputs, individually configurable
0	Not used (default)
128	-10 V...+ 10 V
129	0 mA...20 mA
130	4 mA...20 mA

Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<− Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	1	0...3	48	Analog value overflow at an analog input		Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...10					

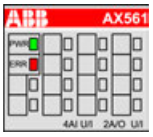
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	14	1...10	3	0...1	48	Analog value overflow at an analog output	Check output value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	3	0...1	7	Analog value underflow at an analog output	Check output value
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI, 3 = AO); COM1/ COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**	**LED flashes**
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more chan-nels of the module

Measuring ranges

CAUTION!

Risk of wrong analog input values!

The analog input values may be wrong if the measuring range of the inputs are exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

Range	-2.5 ... +2.5 V	-5 ... +5 V	0 ... 5 V	0 ... 10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
							Decimal	Hex.
Overflow	>2.9397	>5.8795	>5.8795	>11.7589	>23.5178	>22.8142	32767	7FFF
Measured value too high	2.9397 : 2.5014	5.8795 : 5.0029	5.8795 : : : 5.0015	11.7589 : : : 10.0029	23.5178 : : : 20.0058	22.8142 : : 20.0058	32511 : 27664 27658 27656	7EFF : 6C10 6C0A 6C08
Normal range Normal range or measured value too low	2.5000 : 0.0014	5.0000 : 0.0029	5.0000 : : : 0.0015	10.0000 : : : 0.0029	20.0000 : : : 0.0058	20.0000 : : 4.0058	27648 : 16 10 8	6C00 : 0010 000A 0008
	0.0000	0.0000	0.0000	0.0000	0	4	0	0000
	: -0.0014 : : : -2.5000	: -0.0029 : : : -5.0000				3.9942 : : 0	-10 -16 -4864 -6912 : -27648	FFF6 FFF0 ED00 E500 : 9400
Measured value too low	-2.5014 : -2.9398	-5.0029 : -5.8795					-27664 : -32512	93F0 : 8100
Underflow	<-2.9398	<-5.8795	<-0.0300	<-0.0600	<-0.1200	<-0.1200	-32768	8000

The represented resolution corresponds to 12 bits respectively 11 bits plus sign.

Output ranges

Range	-10 ... +10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Overflow	> 11.7589	> 23.5178	> 22.8142	32767	7FFF
Output value too high	11.7589 : 10.0058 : :	23.5178 : : : 20.0058	22.8142 : : 20.0058 :	32511 : 27664 27658 27656	7EFF : 6C10 6C0A 6C08
Normal range Normal range or output value too low	10.0000 : 0.0058 : :	20,0000 : : : 0.0058	20.0000 : : 4.0058	27648 : 16 10 8	6C00 : 0010 000A 0008
	0.0000	0	4	0	0000
	: -0.0058 : : : -10.0000		3.9942 : : 0	-10 -16 -4864 -6912 : -27648	FFF6 FFF0 ED00 E500 : 9400
Output value too low	-10.0058 : -11.7589			-27664 : -32512	93F0 : 8100
Underflow	< -11.7589		<0.0000	-32768	8000

The represented resolution corresponds to 12 bits respectively 11 bits plus sign.

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 19 for L+ (+24 V DC) and terminal 20 for M (0 V)
	Rated value	24 V DC
	Current consumption via L+ terminal	0.14 A + output load
	Inrush current (at power-up)	0.05 A
	Max. ripple	5 %
	Protection against reversed voltage	Yes

Parameter	Value
Protection fuse for L+	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 5 mA
Galvanic isolation	No
Surge-voltage (max.)	35 V DC for 0.5 s
Max. power dissipation within the module	4.9 W
Weight	Ca. 120 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value	
Number of channels per module		4 individually configurable voltage or current inputs	
Distribution of channels into groups		1 (4 channels per group)	
Resolution			
	Unipolar	Voltage: 0 V...+5 V; 0 V...+10 V: 12 bits Current 0 mA...20 mA; 4 mA...20 mA: 12 bits	
	Bipolar	Voltage -2.5 V...+2.5 V; -5 V...+5 V: 11 bits plus sign	
Connection of the signals I0- to I3-		Terminals 3, 6, 9, 12	
Connection of the signals I0+ to I3+		Terminals 2, 5, 8, 11	
Input type		Differential	
Galvanic isolation		No galvanic isolation between the inputs and the I/O bus	
Common mode input range		Signal voltage plus common mode voltage must be within ±12 V	
Indication of the input signals		No	
Channel input resistance		Voltage: >1 MΩ Current: ca. 250 Ω	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.5 % of full scale (voltage) ±0.5 % of full scale (current 0 mA...20 mA) ±0.7 % of full scale (current 4 mA...20 mA) at 25 °C

Parameter		Value	
		Max.	±2 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Time constant of the input filter		Voltage: 300 µs Current: 300 µs	
Relationship between input signal and hex code		↳ *Table on page 2829*	
Analog to digital conversion time		Typ. 500 µs per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		8 bytes	
Overvoltage protection		Yes, up to 30 V DC only for voltage input	
Max. cable length (conductor cross section > 0.14 mm²)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Technical data of the analog outputs

Parameter	Value	
Number of channels per module	2 configurable voltage or current outputs	
Distribution of channels into groups	1 (2 channels per group)	
Connection of the signals O0U- and O1U+	Terminals 13 and 15	
Connection of the signals O0I+ and O1I+	Terminals 14 and 16	
Output type	Bipolar with voltage, unipolar with current	
Resolution	12 bits or 11 bits plus sign	
Indication of the output signals	No	
Output resistance (load) as current output	0 Ω...500 Ω	
Output load ability as voltage output	2 mA max.	
Relationship between input signal and hex code	Table Output Ranges ↳ *Table on page 2830*	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale (voltage) ±0.5 % of full scale (current 0 mA...20 mA) ±0.7 % of full scale (current 4 mA...20 mA) at 25°C
	Max.	±2 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Unused outputs	Can be left open and should be configured as "unused"	
Output data length	4 bytes	
Overvoltage protection	Yes, up to 30 V DC	

Parameter		Value
Max. cable length (conductor cross section > 0.14 mm²)		
	Unshielded wire	10 m
	Shielded wire	100 m

Ordering data

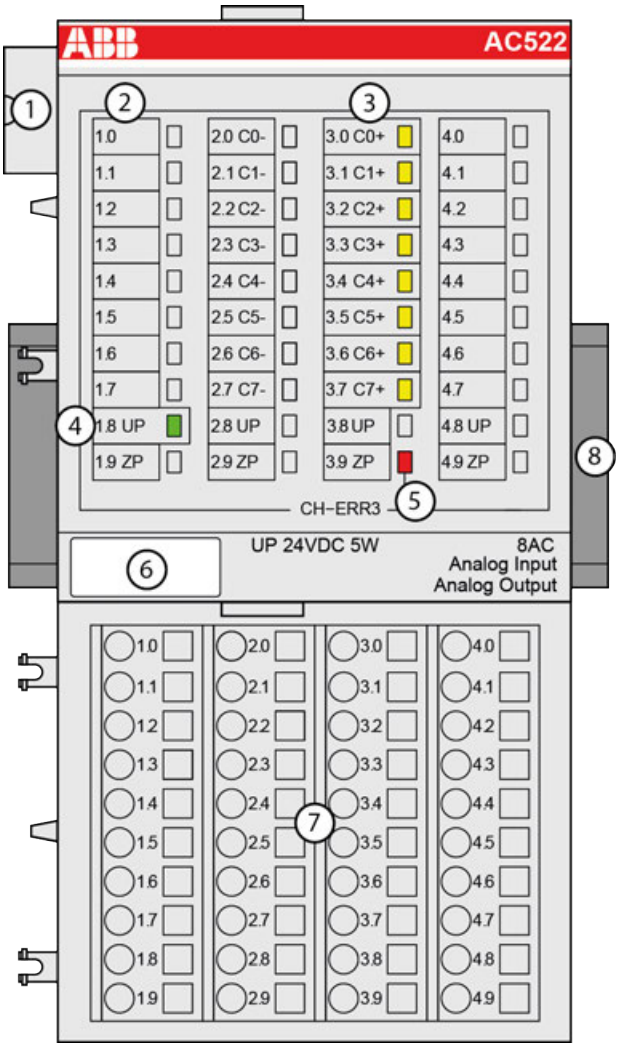
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1301	AX561, analog input/output module, 4 AI, 2 AO, U/I	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

S500

AC522 - Analog input/output module

- 8 configurable analog inputs/outputs in one group (2.0...2.7 and 3.0...3.7)
- Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the analog inputs/outputs (C0 - C7)
4 1 green LED to display the state of the process supply voltage UP
5 1 red LED to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The configuration is performed by software. The modules are supplied with a process voltage of 24 V DC.

The inputs and outputs are galvanically isolated from all other circuitry of the module.

Functionality

8 analog inputs (I0...I7), individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA
- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

4 analog outputs (O0...O3), individually configurable for

- Unused (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

4 analog outputs (O4...O7), individually configurable for

- Unused (default setting)
- -10 V...+10 V

Parameter		**Value**
Resolution of the analog channels		
	Voltage -10 V...+10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		10 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The modules are plugged on an I/O terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8 and 4.8 as well as 1.9, 2.9, 3.9 and 4.9 are electrically interconnected within the I/O terminal units and always have the same assignment, independent of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	Unused	Unused
2.0 to 2.7	C0- to C7-	Negative poles of the 8 analog inputs/outputs
3.0 to 3.7	C0+ to C7+	Positive poles of the analog inputs/outputs
4.0 to 4.7	Unused	Unused

The negative poles of the analog inputs are connected to each other to form an "Analog Ground" signal for the module.

The negative poles of the analog outputs are connected to each other to form an "Analog Ground" signal for the module.

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per I/O module. The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

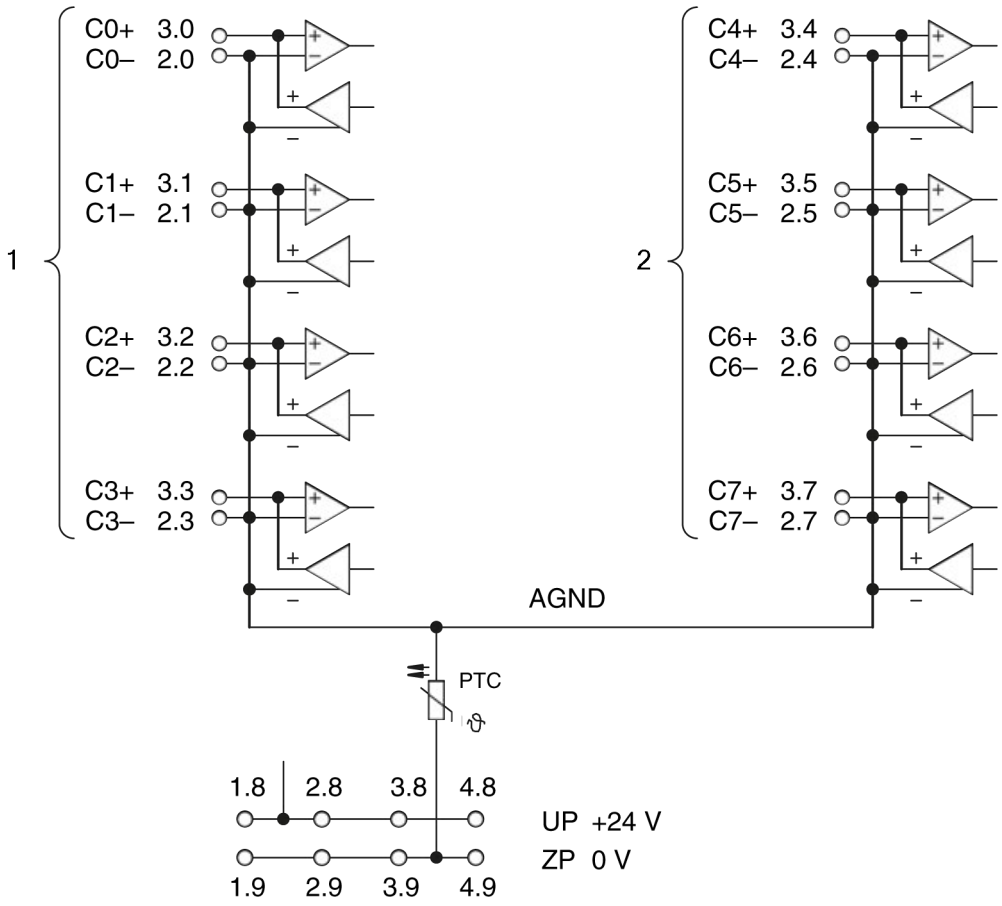
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the I/O module.



1 4 analog I/O channels
as inputs for 0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/Pt1000/Ni1000
digital signals
as outputs for -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA

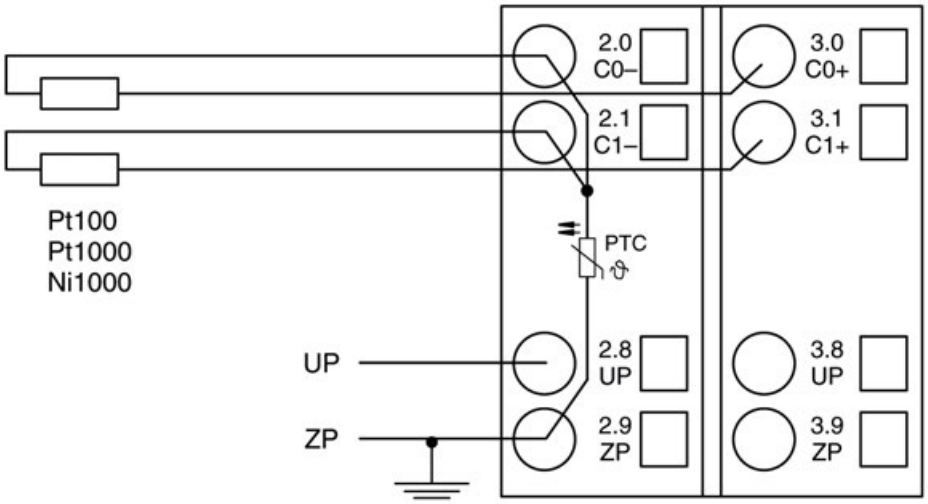
2 4 analog I/O channels
as inputs for 0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/Pt1000/Ni1000
digital signals
as outputs for -10 V...+10 V

The process voltage must be included in the grounding concept of the control system (e.g. grounding the negative pole).

By installing equipotential bonding conductors between the different parts of the system, it must be made ensured that the potential difference between ZP and AGND never exeeds 1 V.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the 8 analog channels.



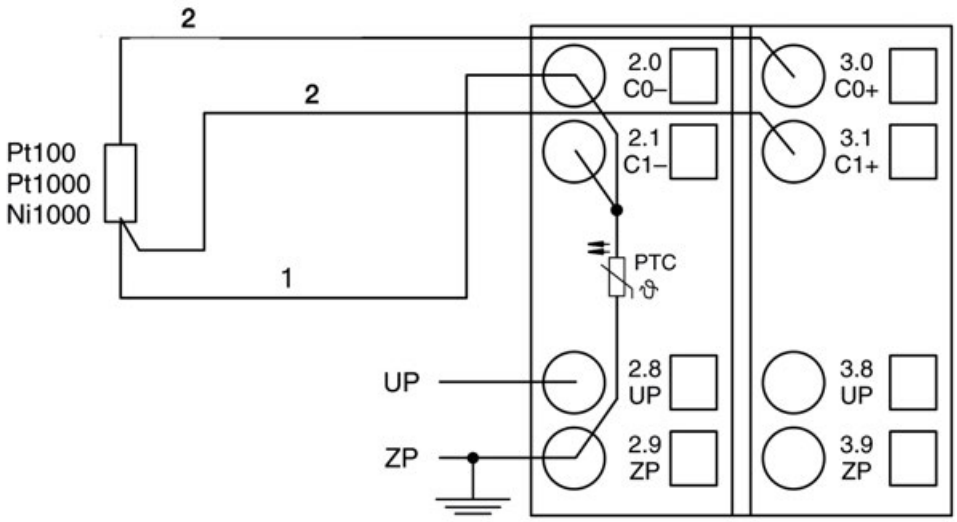
Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.



1 Return line
2 Twisted pair within the cable

> *If several measuring points are adjacent to each other, only one return line is necessary. This saves wiring costs.*

With the 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. C1).

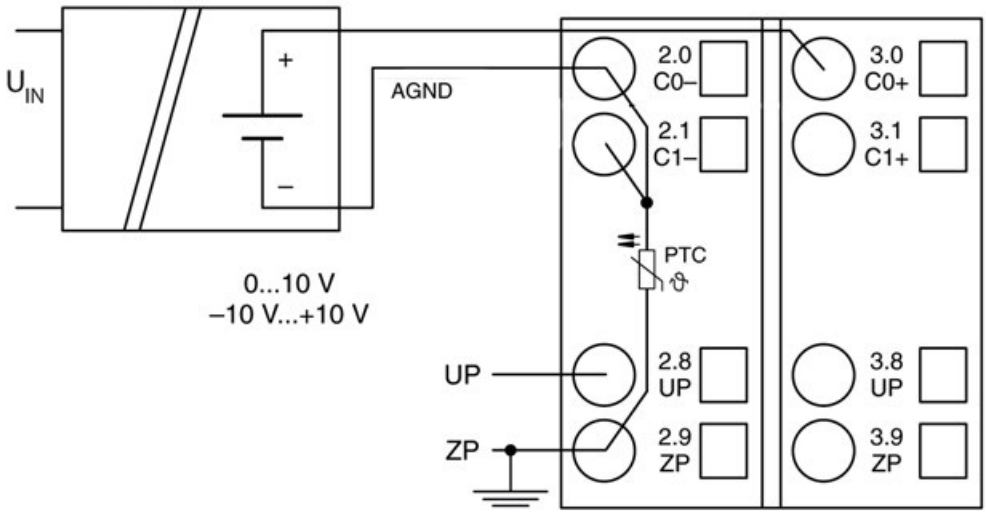
In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply



By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

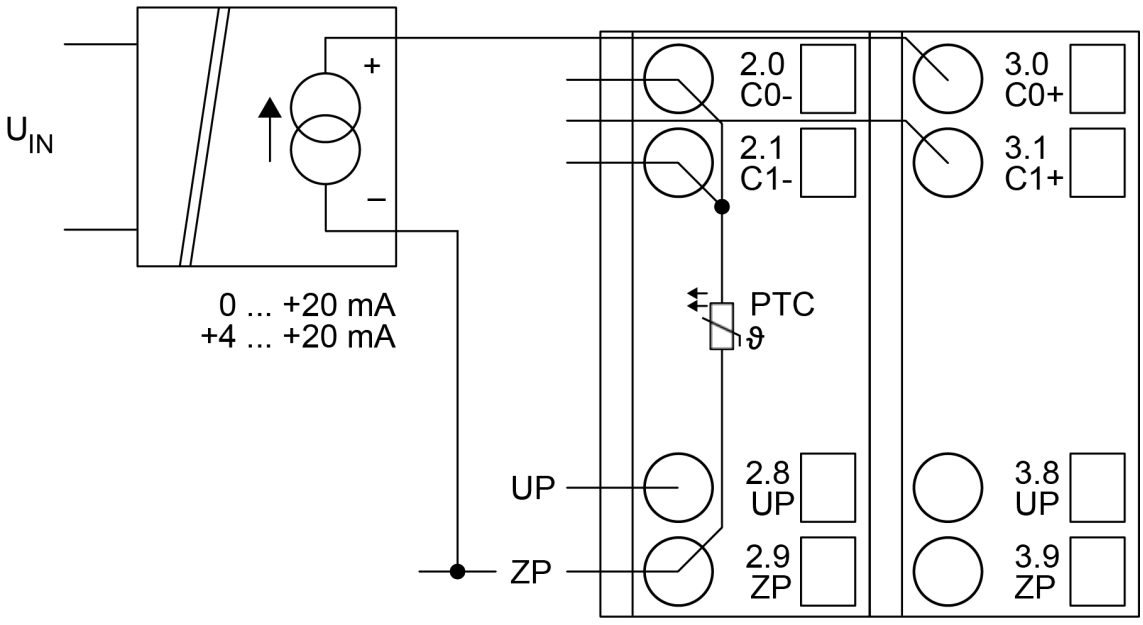
By connecting to AGND the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

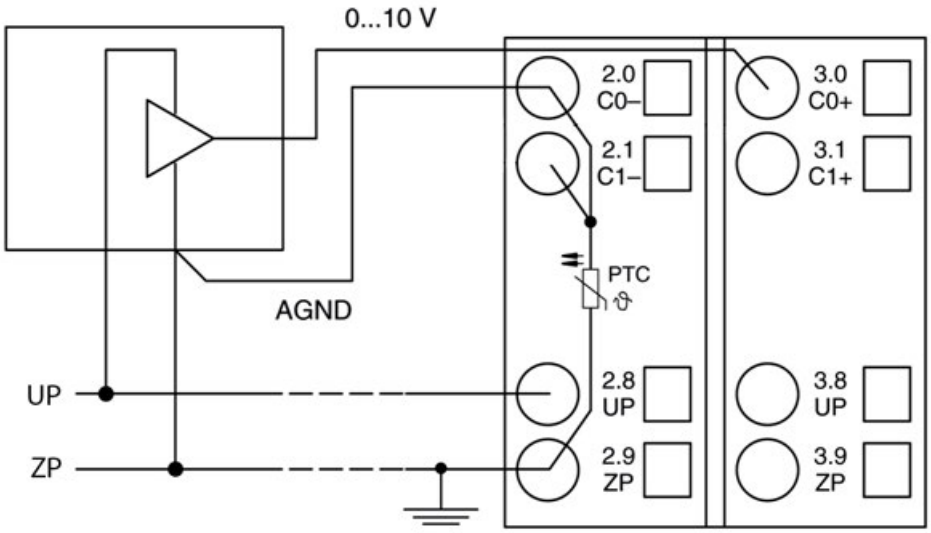


The following measuring ranges can be configured:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply



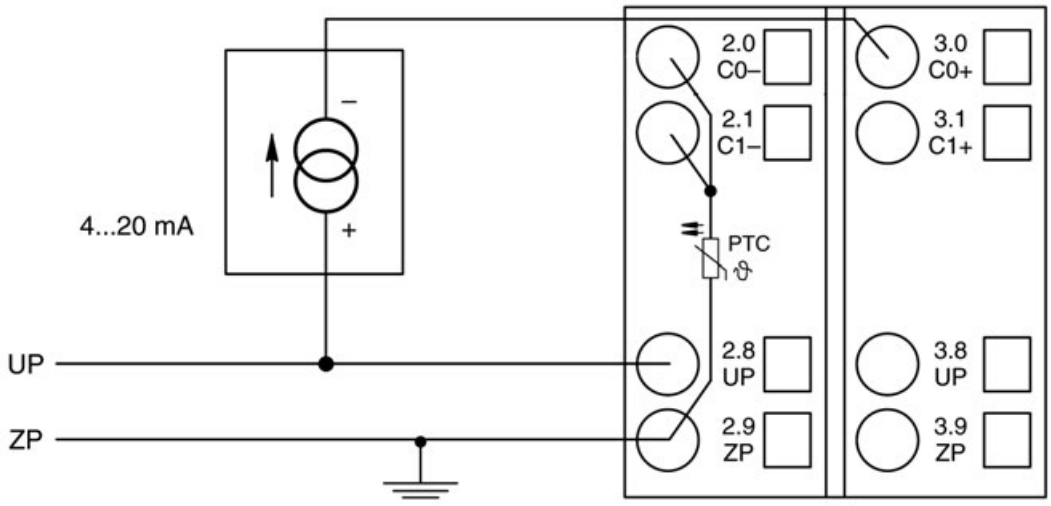
CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1V, not even in case of long lines (see figure Terminal Assignment).

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very small current flows through the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method should be applied.

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used
*) if the sensor can provide this signal range		

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)



Current	4 mA...20 mA	1 channel used

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second to an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10-volt Zener diode (in parallel to I+ and I-). But, in general, sensors with fast initialization or without current peaks higher than 25 mA are preferrable.

Unused input channels can be left open-circuited because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

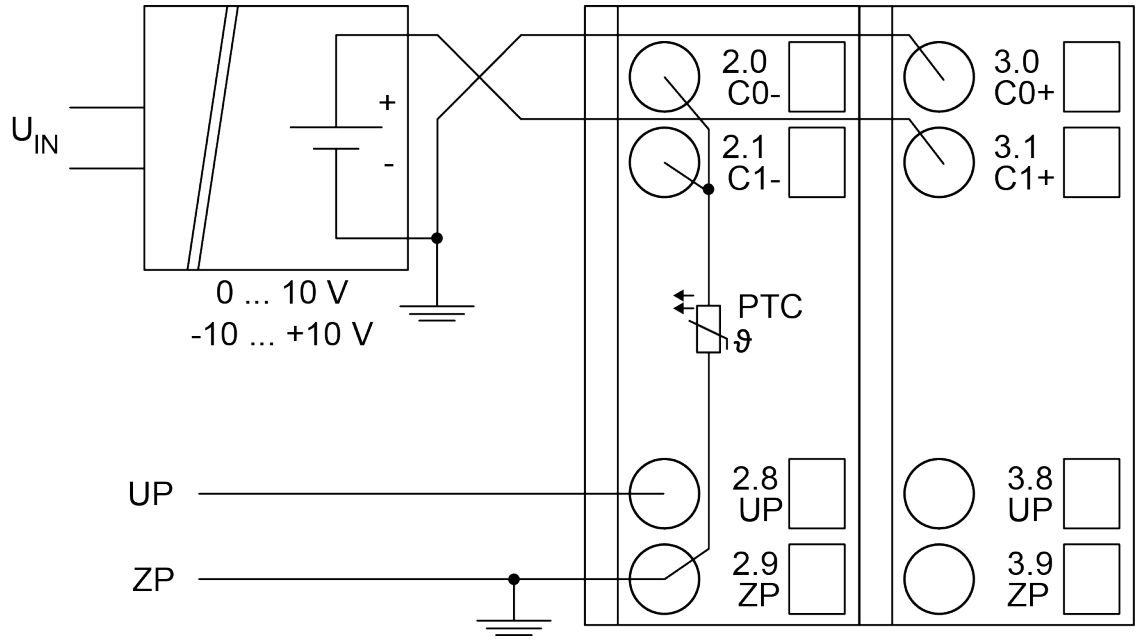
The use of differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise, problems may occur concerning the common-mode input voltages of the involved analog inputs.



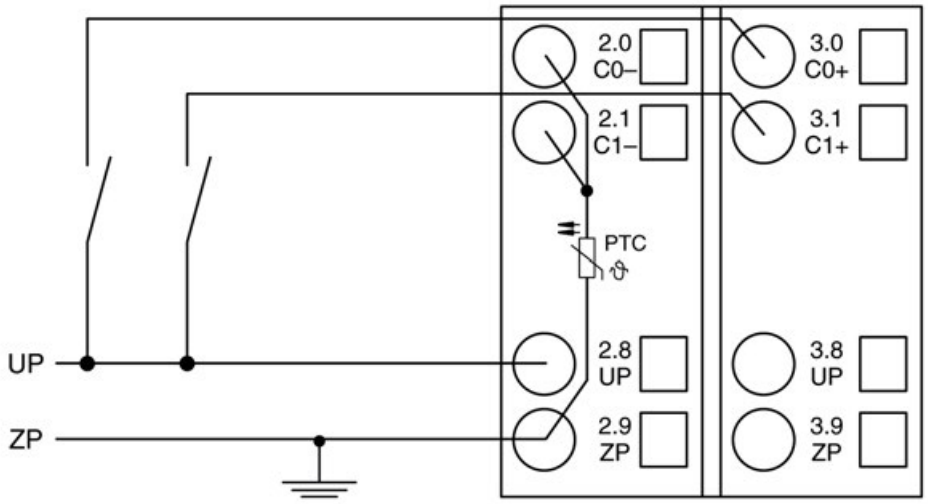
The negative pole of the sensor must be grounded next to the sensor.

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

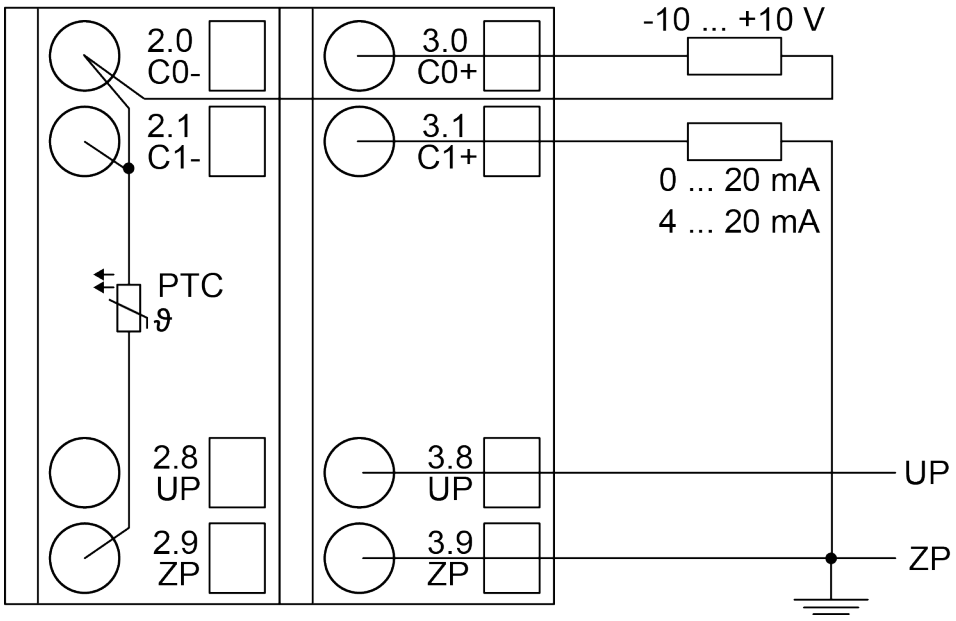
Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.



Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

Connection of analog output loads (Voltage, current)



Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

Unused analog outputs can be left open-circuited.

Internal data exchange

Analog inputs (words)	8
Analog outputs (words)	8

I/O configuration

The module does not store configuration data itself. The 8 configurable analog channels are defined as inputs or outputs by the configuration, i.e. each of the configurable channels can used as input or output (or re-readable output in case of voltage input/output).

When a channel is used as input, the corresponding output must be configured unused.

When a channel is used as output, the corresponding input must be configured unused.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1520 [1])	Word	1520 0x05f0	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			not for FBP
3	Parameter length in bytes	Internal	37	Byte	37-CPU 37-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
7	Channel configuration Input channel 0	see table Channel configuration		Byte	Default 0x00	0	19	0x0Y06
8	Channel monitoring Input channel 0	see table Channel monitoring		Byte	Default 0x00	0	3	0x0Y07
9 to 22	Channel configuration and channel monitoring of the input channels 1 to 7	see tables channel configuration and channel monitoring		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y08 to 0x0Y15
23	Channel configuration Output channel 0	see table Channel configuration		Byte	Default 0x00	0	130	0x0Y16
24	Channel monitoring Output channel 0	see table Channel monitoring		Byte	Default 0x00	0	3	0x0Y17
25	Substitute value Output channel 0	only valid for output channel 0	0...0xffff	Word	Default 0x0000	0	65535	0x0Y18
26 to 31	Channel configuration and channel monitoring of the output channels 1 to 3	see tables channel configuration and channel monitoring		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y19 to 0x0Y1E

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
32	Channel configuration Output channel 4	see table Channel configuration		Byte	Default 0x00	0	128	0x0Y1F
33	Channel monitoring Output channel 4	see table Channel monitoring		Byte	Default 0x00	0	3	0x0Y20
34 to 39	Channel configuration and channel monitoring of the output channels 5 to 7	see tables channel configuration and channel monitoring		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y21 to 0x0Y26

[1]) With CS31 and addresses less than 70 and FBP, the value is increased by 1

[2]) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	40
Ext_User_Prm_Data_Const(0) =	0x05, 0xf1, 0x25, \ 0x01, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Table 487: Input channel (8x)

No.	Name	Internal value, type	Default
1	Channel configuration see table [2])	Byte	0 0x00 see table [2])
2	Channel monitoring see table [3])	Byte	0 0x00 see table [3])

Table 488: Channel configuration [2])

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default)
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 489: Channel monitoring [3])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit
1	Open-circuit and short-circuit
2	Plausibility
3	No monitoring

Table 490: Output channel 0 (1 channel)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [4])	see table [4])	Byte	see table [4])
2	Channel monitoring	see table [5])	see table [5])	Byte	see table [5])
3	Substitute value see table [6])	0...65535	0... 0xffff	Word	0

Table 491: Output channels 1...7 (7x)

No.	Name	Internal value, type	Default
1	Channel configuration see table [4])	Byte	see table [4])
2	Channel monitoring see table [5])	Byte	see table [5])

Table 492: Channel configuration [4])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V
129	Analog output 0 mA...20 mA (not with the channels 4...7)
130	Analog output 4 mA...20 mA (not with the channels 4...7)

Table 493: Channel monitoring [5])

Internal value	Monitoring
0	Plausibility, open circuit (broken wire) and short circuit (default)
1	Open-circuit (broken wire) and short-circuit
2	Plausibility
3	No monitoring

Table 494: Substitute value [6])

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value	Last value	0
Substitute value	Off or last value	1...65535

Diagnosis

Table 495: Possible diagnosis of I/O channels

Output range	Condition	
	Output value in the PLC underflow	Output value in the PLC overflow
0..20 mA	Error identifier = 7	Error identifier = 4
4..20 mA		
-10..+10 V		

Input range	Condition			
	Short circuit	Wire break	Input value underflow	Input value overflow
0..20 mA	no diagnosis possible	no diagnosis possible	no diagnosis possible	Error identifier = 48
4..20 mA	Error identifier = 7	Error identifier = 7	Error identifier = 7	Error identifier = 48
-10..+10 V	no diagnosis possible	Error identifier = 48	Error identifier = 7	Error identifier = 48

Table 496: Content of diagnosis messages

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])				
3	14	1...10	31	31		26	Parameter error	Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31		11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31		45	Process voltage is switched off (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
				AX521	AX522			
4	14	1...10	1	0...3	0...7	48	Analog value over-flow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	7	Analog value under-flow at an analog input	Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	4	Analog value over-flow at an analog output	Check output value
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	7	Analog value under-flow at an analog output	Check output value
	11 / 12	ADR	1...10					

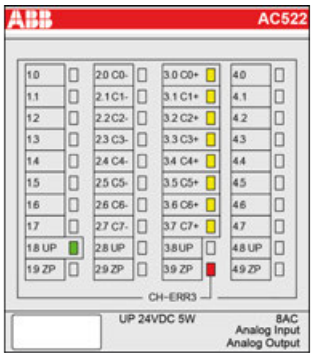
Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs/ outputs 00...07	Analog input/ output	Yellow	Input/output is OFF	Input/output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR3	Channel error, error messages combined into group 3	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group

Measuring ranges

Input ranges of voltage, current and digital input

The represented resolution corresponds to 16 bits.

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	OFF	0	0000

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

Input ranges resistance temperature detector

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V	23.5178 mA	22.8142 mA	32511	7EFF
	:	:	:	:	:
	10.0004 V	20.0007 mA	20.0006 mA	27649	6C01
Normal range	10.0000 V	20.0000 mA	20.0000 mA	27648	6C00
	:	:	:	:	:
	0.0004 V	0.0007 mA	4.0006 mA	1	0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V	0 mA	3.9994 mA	-1	FFFF
	:	:	0 mA	-6912	E500
	-10.0000 V	0 mA	0 mA	-27648	9400
Value too low	-10.0004 V	0 mA	0 mA	-27649	93FF
	:	:	:	:	:
	-11.7589 V	0 mA	0 mA	-32512	8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The System Data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

Only additional details are therefore documented below.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 VDC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 VDC power supply at the terminals UP/L+ and ZP/M of the CPU/bus module	Ca. 2 mA
	From UP at normal operation	0.10 A + output loads
Inrush current from UP (at power up)		0.040 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm²	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switch-gear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	8	
Distribution of channels into groups	1 group of 8 channels	
Connections of the channels C0- to C7-	Terminals 2.0 to 2.7	
Connections of the channels C0+ to C7+	Terminals 3.0 to 3.7	
Input type	Bipolar (not with current or Pt100/Pt1000/Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs Current: 100 µs	
Indication of the input signals	One LED per channel	
Conversion cycle	2 ms (for 8 inputs + 8 outputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between input signal and hex code	See table *Chapter 1.6.3.6.2.2.1.9.1 “Input ranges of voltage, current and digital input” on page 2853*	

Parameter	Value
Unused inputs	Must be configured as "unused".
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 8
Distribution of channels into groups		1 group of 8 channels
Connections of the channels C0+ to C7+		Terminals 3.0 to 3.7
Reference potential for the inputs		Terminals 1.9 to 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 VDC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		8, all channels for voltage, the first 4 channels also for current
Distribution of channels into groups		1 group of 8 channels
	Channels C0-...C7-	Terminals 2.0...2.7
	Channels C0+...C7+	Terminals 3.0...3.7
Output type		Bipolar with voltage, unipolar with current
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		Max. ±10 mA
Indication of the output signals		One LED per channel
Resolution		12 bits (+ sign)

Parameter	Value	
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between output signal and hex code	See table *Chapter 1.6.3.6.2.2.1.9.3 "Output ranges voltage and current" on page 2855*	
Unused outputs	Must be configured as "unused".	

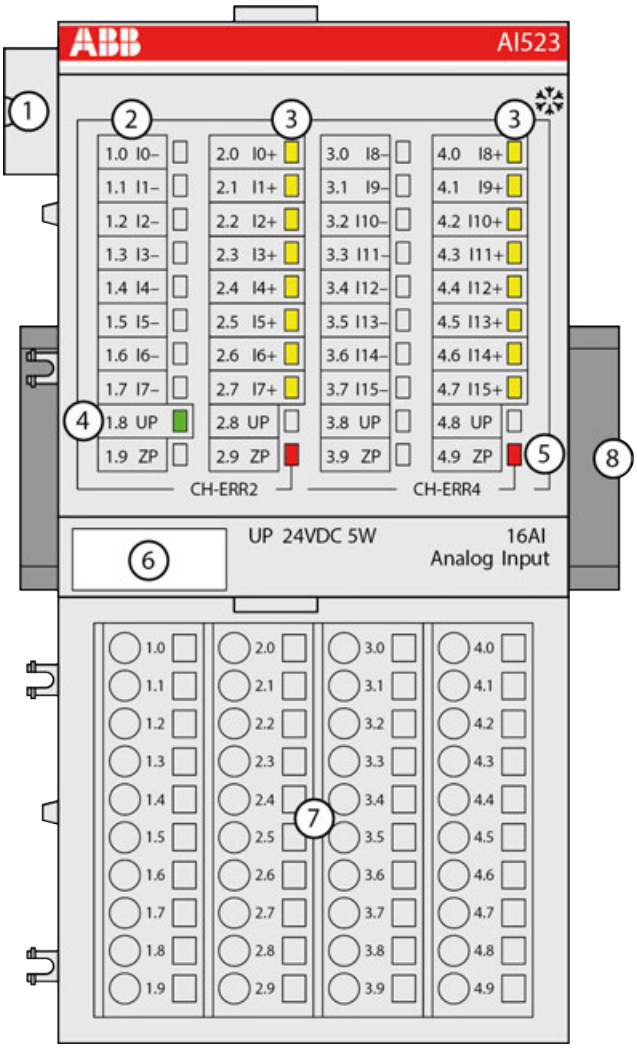
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 500 R0001	AC522, analog input/output module, 8 AC, U/I/RTD, 12 bits + sign, 2-wires	Active
1SAP 450 500 R0001	AC522-XC, analog input/output module, 8 AC, U/I/RTD, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AI523 - Analog input module

- 16 configurable analog inputs (I0 to I15) in 2 groups (1.0...2.7 and 3.0...4.7)
  Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states at the analog inputs (I0 - I15)
4 1 green LED to display the state of the process supply voltage UP
5 2 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

16 analog inputs, individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V... +10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		19 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

The modules are plugged on an I/O terminal unit ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal units and have always the same assignment, independent of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0- to I7-	Negative poles of the first 8 analog inputs
2.0 to 2.7	I0+ to I7+	Positive poles of the first 8 analog inputs
3.0 to 3.7	I8- to I15-	Negative poles of the following 8 analog inputs
4.0 to 4.7	I8+ to I15+	Positive poles of the following 8 analog inputs

CAUTION!
The negative poles of the analog inputs are galvanically connected to each other. They form an "Analog Ground" signal for the module. The negative poles of the analog outputs are also galvanically connected to each other to form an "Analog Ground" signal.

CAUTION!
There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

CAUTION!
Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per AI523.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the module:

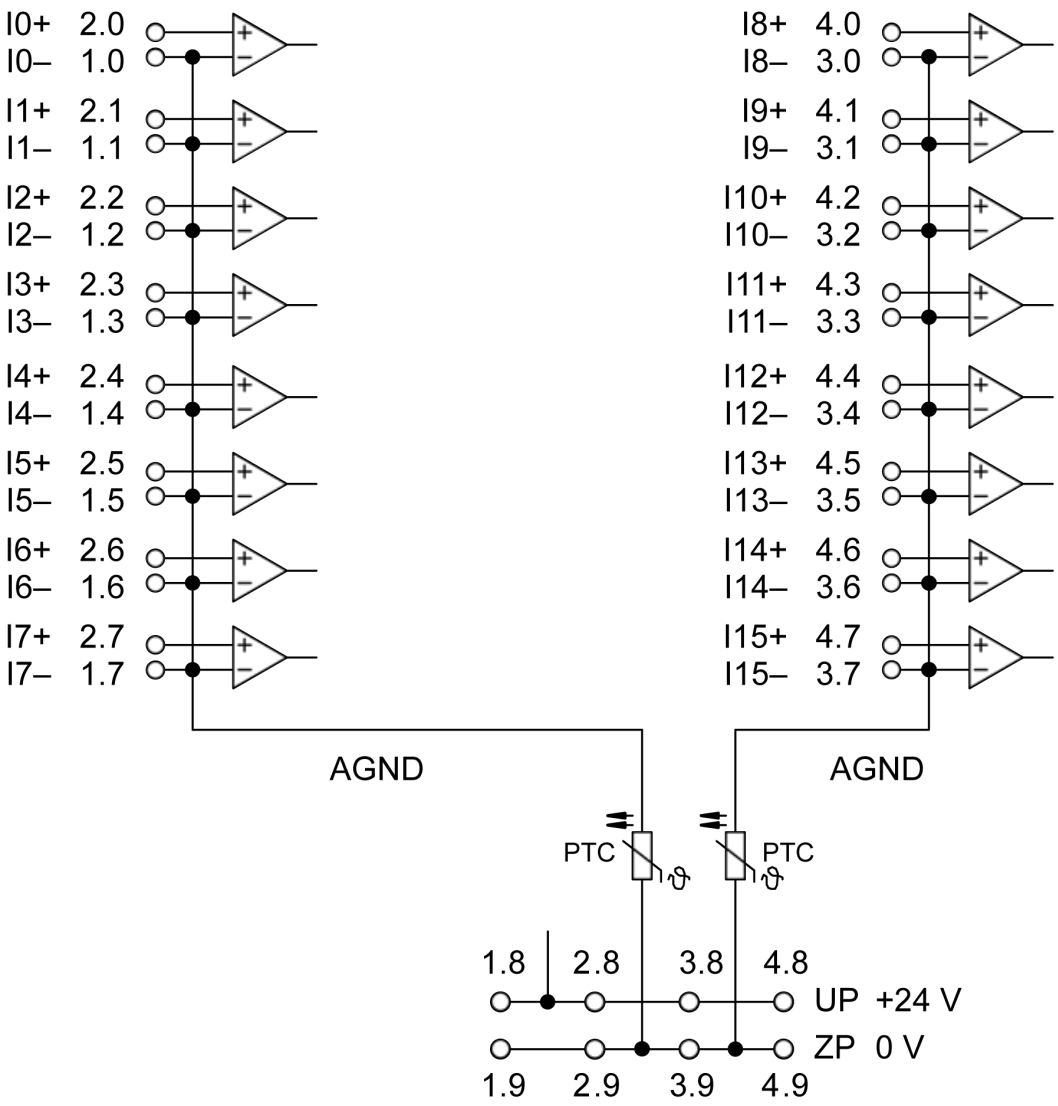


Fig. 122: 16 analog inputs in two groups, individually configurable *Chapter 1.6.3.6.2.2.2.2 “Functionality” on page 2859*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The modules provide several diagnosis functions *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI523 provides a constant current source which is multiplexed over the 8 analog channels.

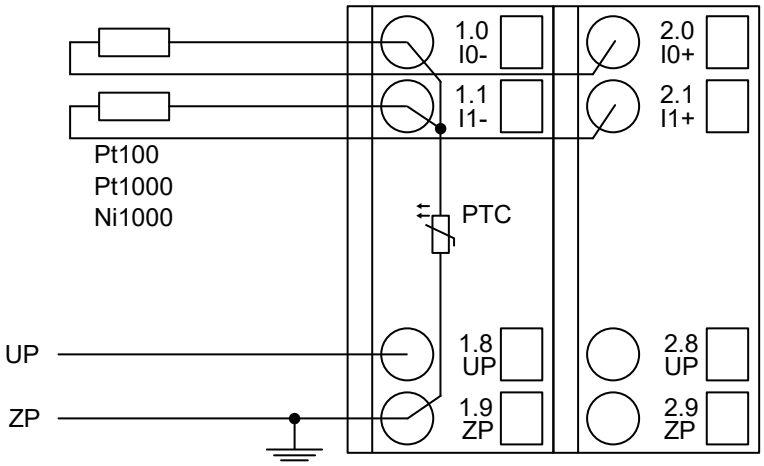


Fig. 123: Connection example

The following measuring ranges can be configured *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871*.

Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The function of the LEDs is described under Displays *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

The module AI523 performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI523 provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.

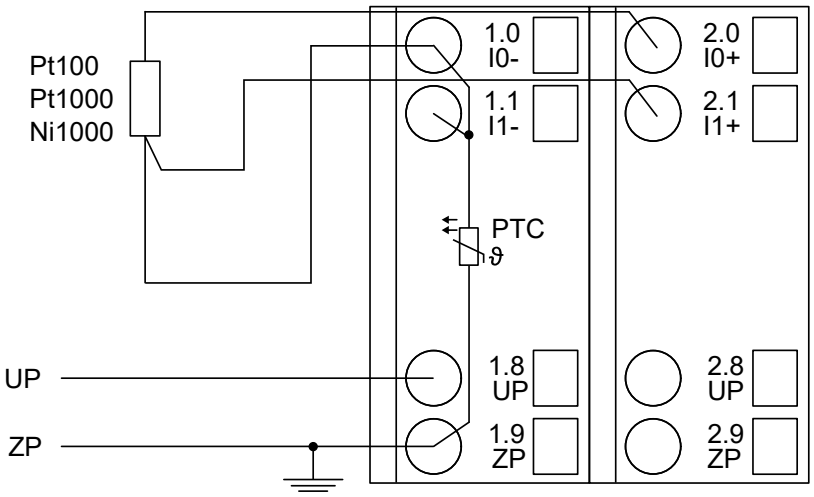


Fig. 124: Connection example

> *If several measuring points are adjacent to each other, the return line is necessary only once. This saves wiring costs.*

With 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871*

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The function of the LEDs is described under Displays *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

The module AI523 performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

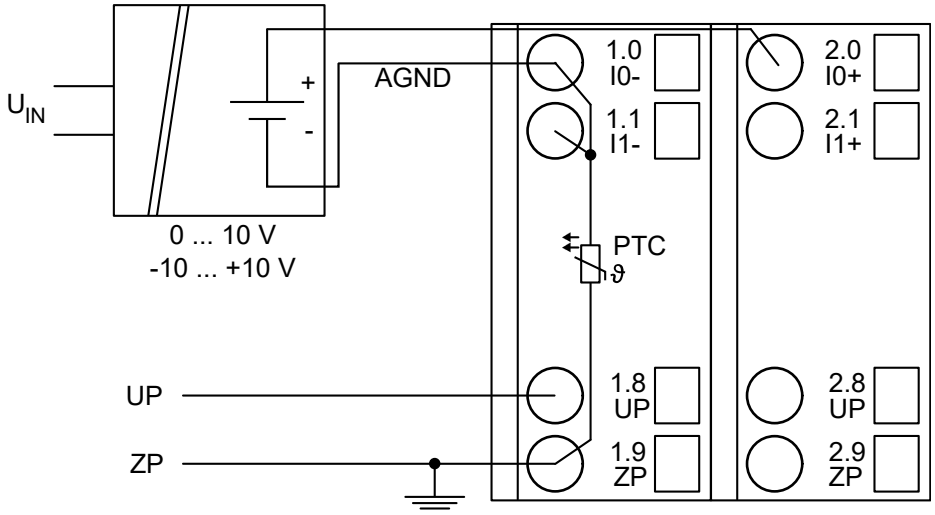


Fig. 125: Connection example

By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871* ↳ *Chapter 1.6.3.6.2.2.2.9 “Measuring ranges” on page 2876*

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Displays ↳ *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

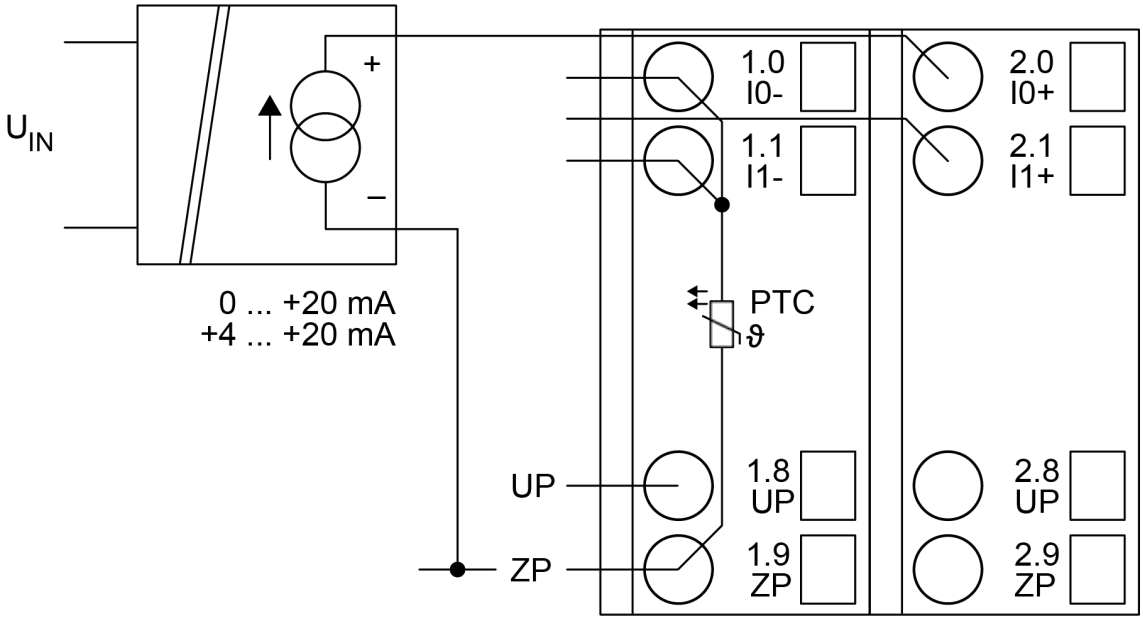


Fig. 126: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871* ↳ *Chapter 1.6.3.6.2.2.2.9 “Measuring ranges” on page 2876*

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Displays ↳ *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

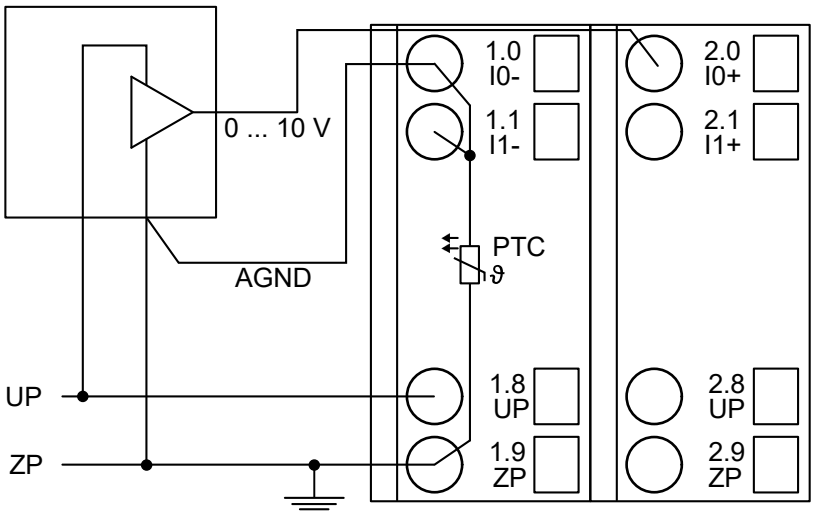


Fig. 127: Connection example

CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1 V, not even in case of long lines .

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very low current flows over the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method has to be preferred.

The following measuring ranges can be configured *Chapter 1.6.3.6.2.2.2.9 “Measuring ranges” on page 2876*

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used
*) if the sensor can provide this signal range		

The function of the LEDs is described under Displays *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

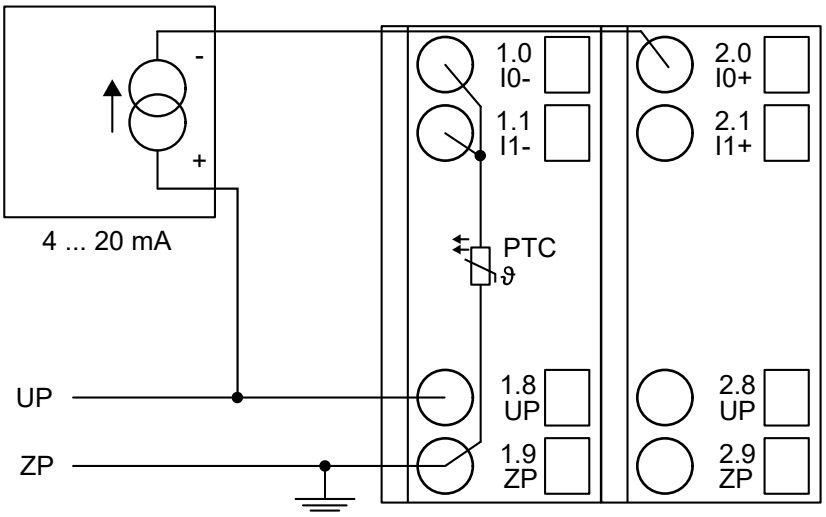


Fig. 128: Connection example

The following measuring ranges can be configured *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871* *Chapter 1.6.3.6.2.2.2.9 “Measuring ranges” on page 2876*

Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Displays *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second into an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10 volt Zener diode (in parallel to I+ and I-). But, in general, it is a better solution to use sensors with fast initialization or without current peaks higher than 25 mA.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise problems can occur concerning the common-mode input voltages of the involved analog inputs.

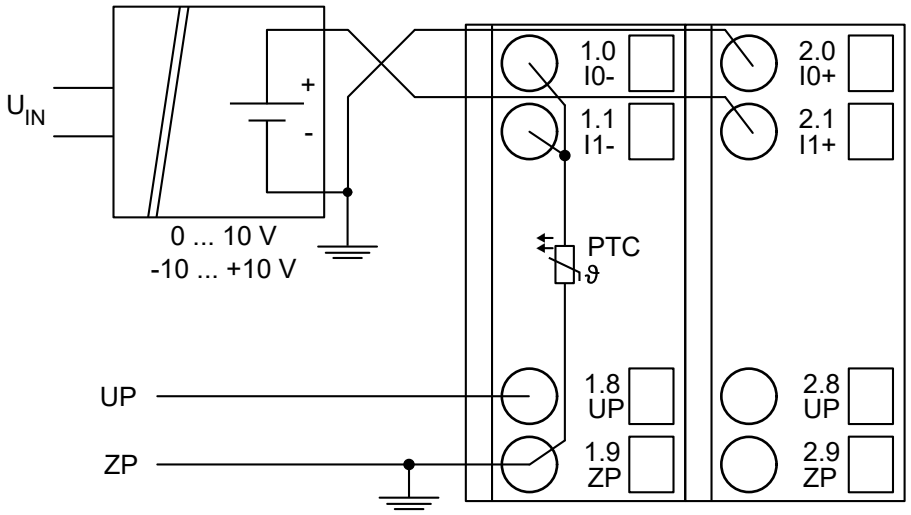


Fig. 129: Connection example

The negative pole of the sensor must be grounded next to the sensor.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871* ↳ *Chapter 1.6.3.6.2.2.2.9 “Measuring ranges” on page 2876*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

The function of the LEDs is described under Displays ↳ *Chapter 1.6.3.6.2.2.2.7 “Diagnosis” on page 2874*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

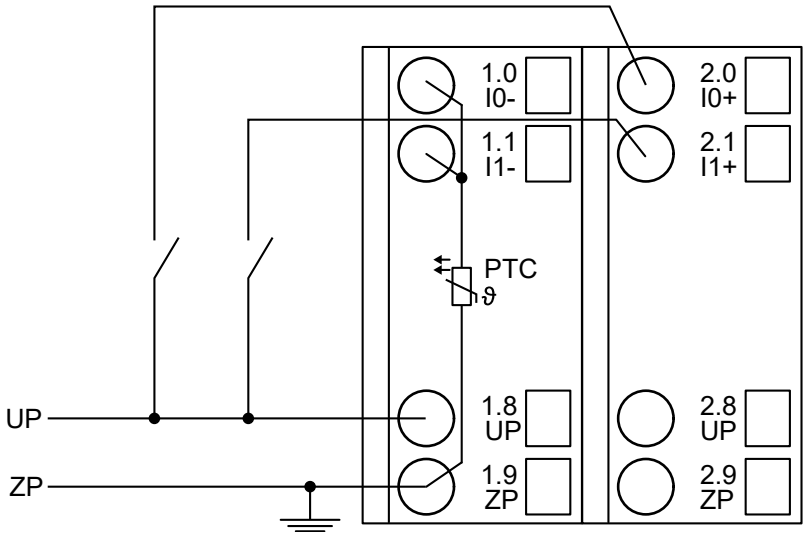


Fig. 130: Connection example

The following operating mode can be configured ↳ *Chapter 1.6.3.6.2.2.2.6 “Parameterization” on page 2871* ↳ *Chapter 1.6.3.6.2.2.2.9 “Measuring ranges” on page 2876*

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

The function of the LEDs is described under Displays.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	16
Counter output data (words)	0

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

That means replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1515 [1)]	Word	1515 0x05eb	0	65535	0x0Y01
2	Ignore module [2)]	No Yes	0 1	Byte	No 0x00			not for FBP
3	Parameter length in bytes	Internal	34	Byte	34-CPU 34-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Channel configuration Input channel 0	See *Table 497 “Channel configuration [2)]” on page 2873*		Byte	Default 0x00	0	19	0x0Y05

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
7	Channel monitoring Input channel 0	See *Table 498 “Channel monitoring [4])” on page 2873*		Byte	Default 0x00	0	3	0x0Y06
8 to 35	Channel configuration and channel monitoring of the input channels 1 to 14	See *Table 497 “Channel configuration [2])” on page 2873* and *Table 498 “Channel monitoring [4])” on page 2873*		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y07 to 0x0Y22
36	Channel configuration Input channel 15	See *Table 497 “Channel configuration [2])” on page 2873*		Byte	Default 0x00	0	19	0x0Y23
37	Channel monitoring Input channel 15	See *Table 498 “Channel monitoring [4])” on page 2873*		Byte	Default 0x00	0	3	0x0Y24
[1]) With CS31 and addresses less than 70 and FBP, the value is increased by 1 [2]) Not with FBP								

GSD file:

Ext_User_Prm_Data_Len =	37
Ext_User_Prm_Data_Const(0) =	0x05, 0xec, 0x22, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Input channel (16 x with AI523)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [2])	see table [2])	Byte	0 0x00 see [3])
2	Channel monitoring	see table [4])	see table [4])	Byte	0 0x00 see [5])

Table 497: Channel configuration [2])

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default) [3])
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 498: Channel monitoring [4])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit [5])
1	Open-circuit and short circuit
2	Plausibility
3	No monitoring

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	

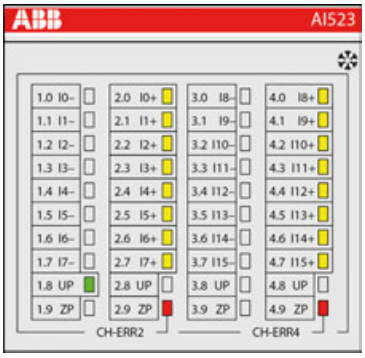
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON -> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...15	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...15	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				
4	14	1...10	1	0...15	47	Short circuit at an analog input	Check ter-minal
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1..10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1..10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI); COM1/COM2: 1..10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7 and I8...I15	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Measuring ranges

Input ranges of voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	OFF	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	< -1.7593	<-11.7589	<0.0000	<1.1858		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

The resolution corresponds to 16 bits.

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Normal	:	400.0 °C	:	4000	0FA0
range	:	:	150.0 °C	1500	05DC
	70.0 °C	:	:	700	02BC
	:	:	:	:	:
	0.1 °C	0.1 °C	0.1 °C	1	0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:
	-50.0 °C	-50.0 °C	-50.0 °C	-500	FE0C
Measured value too low	-50.1 °C	-50.1 °C	-50.1 °C	-501	FE0B
	:	:	:	:	:
	-60.0 °C	-60.0 °C	-60.0 °C	-600	FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + output loads
Inrush current from UP (at power up)		0.050 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm^2	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	16	
Distribution of channels into groups	2 groups of 8 channels each	
Connections of the channels I0- to I7- Connections of the channels I0+ to I7+	Terminals 1.0 to 1.7 Terminals 2.0 to 2.7	
Connections of the channels I8- to I15- Connections of the channels I8+ to I15+	Terminals 3.0 to 3.7 Terminals 4.0 to 4.7	
Input type	Bipolar (not with current or Pt100/ Pt1000/ Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0/4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs Current: 100 µs	
Indication of the input signals	1 LED per channel	
Conversion cycle	2 ms (for 16 inputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits	
	Range -10 V...+10 V: 12 bits + sign	
	Range 0 mA...20 mA: 12 bits	
	Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C

Parameter	Value	
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between input signal and hex code	↳ *Chapter 1.6.3.6.2.2.2.9.1 “Input ranges of voltage, current and digital input” on page 2876* ↳ *Chapter 1.6.3.6.2.2.2.9.2 “Input ranges resistance temperature detector” on page 2876*	
Unused voltage inputs	Are configured as "unused"	
Unused current inputs	Have a low resistance, can be left open-circuited	
Overvoltage protection	Yes	

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 16
Distribution of channels into groups		2 groups of 8 channels each
Connections of the channels I0+ to I7+		Terminals 2.0 to 2.7
Connections of the channels I8+ to I15+		Terminals 4.0 to 4.7
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

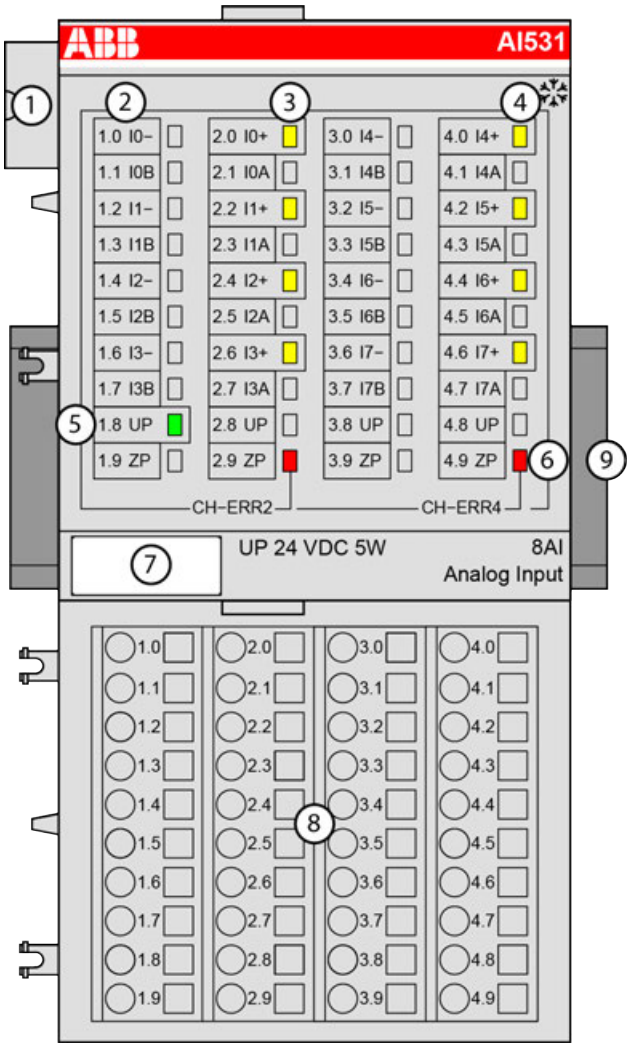
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 300 R0001	AI523, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires	Active
1SAP 450 300 R0001	AI523-XC, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

AI531 - Analog input module

- 8 configurable analog inputs (I0 to I7) in 2 groups (1.0...1.7 and 2.0...2.7 as well as 3.0...3.7 and 4.0...4.7)
  Resolution 15 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal names
3 4 yellow LEDs to display the states at the inputs I0 to I3
4 4 yellow LEDs to display the states at the inputs I4 to I7
5 1 green LED to display the process supply voltage UP
6 2 red LEDs to display errors (CH-ERR2 and CH-ERR4)
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

8 analog inputs, individually configurable for

- Unused (default setting)
- 0 V...5 V, 0 V...10 V
- -50 mV...+50 mV, -500 mV...+500 mV
- -1 V...+1 V, -5 V...+5 V, -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA
- -20 mA...20 mA
- Pt100, -50 °C...+70 °C or 400 °C (2-, 3- and 4-wire)
- Pt100, -200 °C...+850 °C (2-, 3- and 4-wire)
- Pt1000, -50 °C...+400 °C (2-, 3- and 4-wire)
- Ni1000, -50 °C...+150 °C (2-, 3- and 4-wire)
- Cu50 (1.426): -50 °C...+200 °C (2-, 3- and 4-wire)
- Cu50 (1.428): -200 °C...+200 °C (2-, 3- and 4-wire)
- 0 Ω...50 kΩ
- Thermocouples of types J, K, T, N, S
- Resistance measuring bridge
- Digital signals (digital input)

Parameter		Value
Resolution of the analog channels		
	Voltage and current, bipolar	15 bits plus sign
	Voltage and current, unipolar	15 bits
	Temperature	0.1 °C (0,01°C at Pt100 -50 °C...+70 °C)
LED displays		11 LEDs for signals and error messages
Internal power supply		through the I/O bus interface (I/O bus)
External power supply		via terminals (process voltage UP = 24 V DC)
Required terminal unit		TU515 or TU516 ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The modules are plugged on an I/O terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8, 4.8, 1.9, 2.9, 3.9 and 4.9 are electrically interconnected within the I/O terminal units and always have the same assignment, independent of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V

The assignment of the other terminals:

Terminals	Signal	Description
2.0, 2.2, 2.4, 2.6	I0+ to I3+	Positive poles of the first 4 analog inputs
1.0, 1.2, 1.4, 1.6	I0- to I3-	Negative poles of the first 4 analog inputs
2.1, 2.3, 2.5, 2.7	I0A to I3A	Connections A (supply) of the first 4 analog inputs
1.1, 1.3, 1.5, 1.7	I0B to I3B	Connections B (analog ground) of the first 4 analog inputs
4.0, 4.2, 4.4, 4.6	I4+ to I7+	Positive poles of the following 4 analog inputs
3.0, 3.2, 3.4, 3.6	I4- to I7-	Negative poles of the following 4 analog inputs
4.1, 4.3, 4.5, 4.7	I4A to I7A	Connections A (supply) of the following 4 analog inputs
3.1, 3.3, 3.5, 3.7	I4B to I7B	Connections B (analog ground) of the following 4 analog inputs

CAUTION!
Analog sensors must be galvanically isolated against the ground. In order to avoid inaccuracy with the measuring results, the analog sensors should also be isolated against the power supply.

The "IxB" clamps (x=0..7) of the analog inputs are galvanically connected to each other. They form an "Analog Ground Signal" (AGND) for the module.

The negative poles of the analog inputs Ix- may accept a potential difference up to ±20 V DC with regard to the common reference potential IxB (AGND, ZP). Observing this maximum voltage difference, analog current inputs of one module can be switched in series to each other and also with current inputs of other modules.

For the open-circuit detection (cut wire), each positive analog input channel Ix+ is pulled up to "plus" by a high-resistance resistor and each negative analog input channel Ix- is pulled down to "minus" by a resistor. If cut wire occurs, a maximum voltage (overflow or underflow) will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per AI531.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

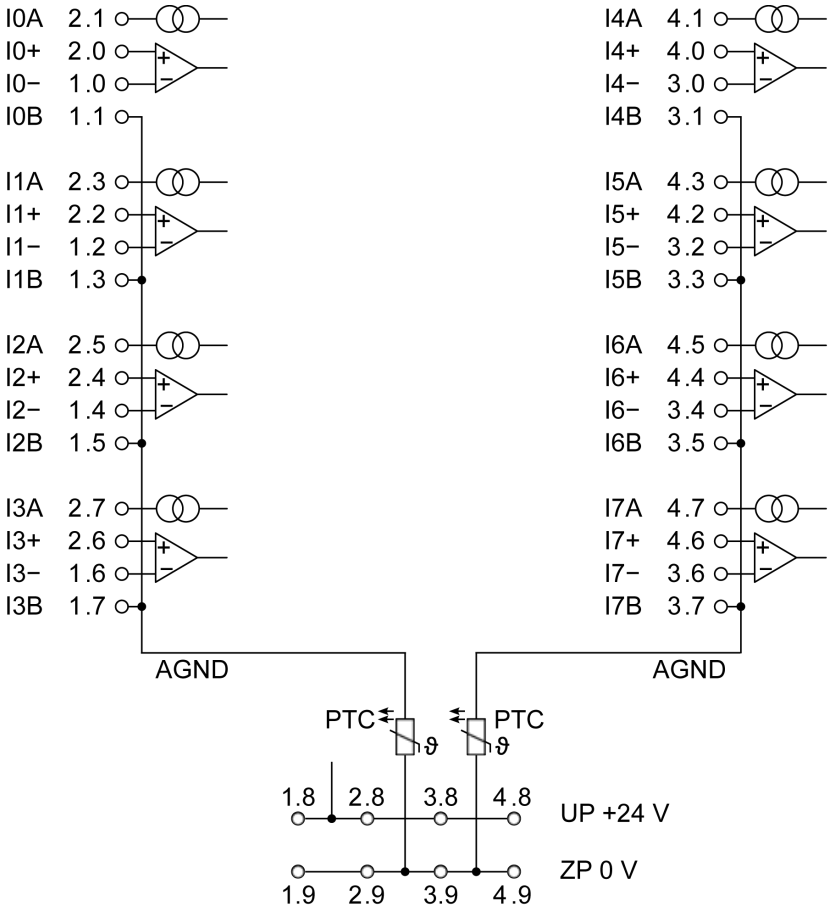


Fig. 131: 8 analog inputs in two groups, individually configurable ↳ *Chapter 1.6.3.6.2.2.3.2 “Functionality” on page 2881*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The module provides several diagnosis functions ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

Standard ranges

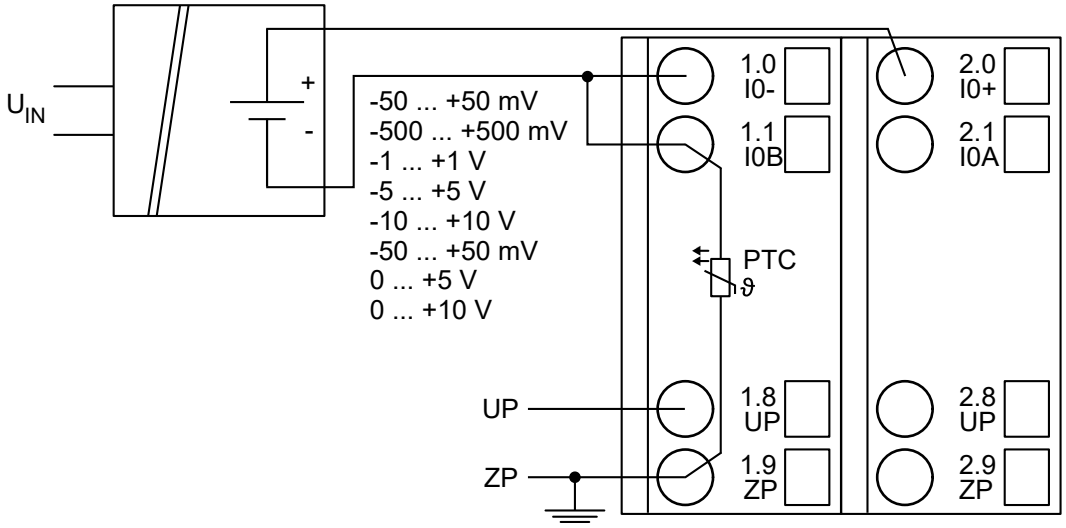


Fig. 132: Connection example

The measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Voltage	-50 mV...+50 mV	1 channel used
Voltage	-500 mV...+500 mV	1 channel used
Voltage	-1 V...+1 V	1 channel used
Voltage	-5 V...+5 V	1 channel used
Voltage	-10 V...+10 V	1 channel used
Voltage	0 V...+5 V	1 channel used
Voltage	0 V...+10 V	1 channel used

Common mode range (+/-20 V)

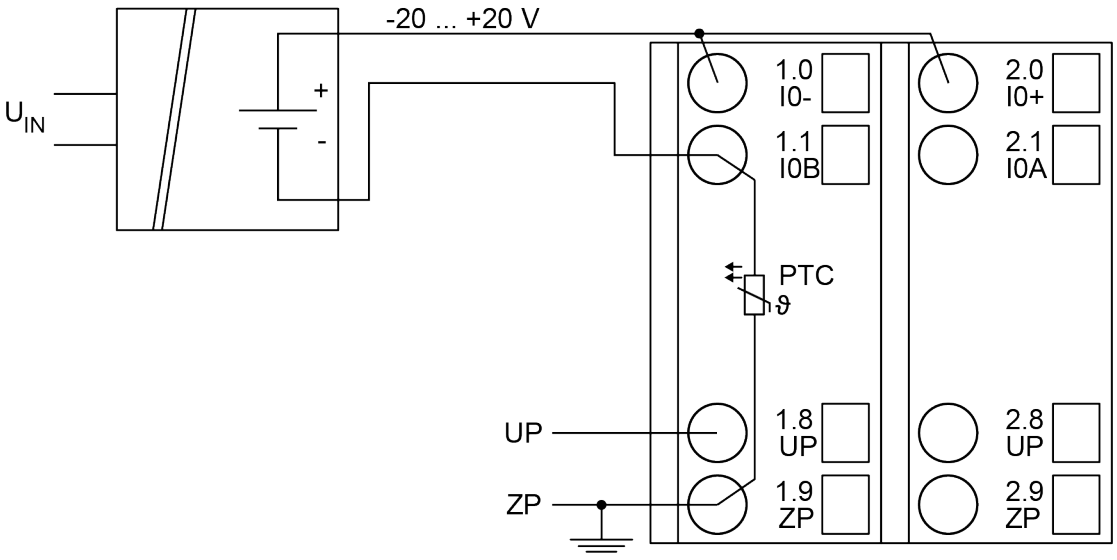


Fig. 133: Connection example

The measuring range can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Voltage	Common mode voltage	1 channel used

The function of the LEDs is described under Diagnosis and displays / displays *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

Standard ranges

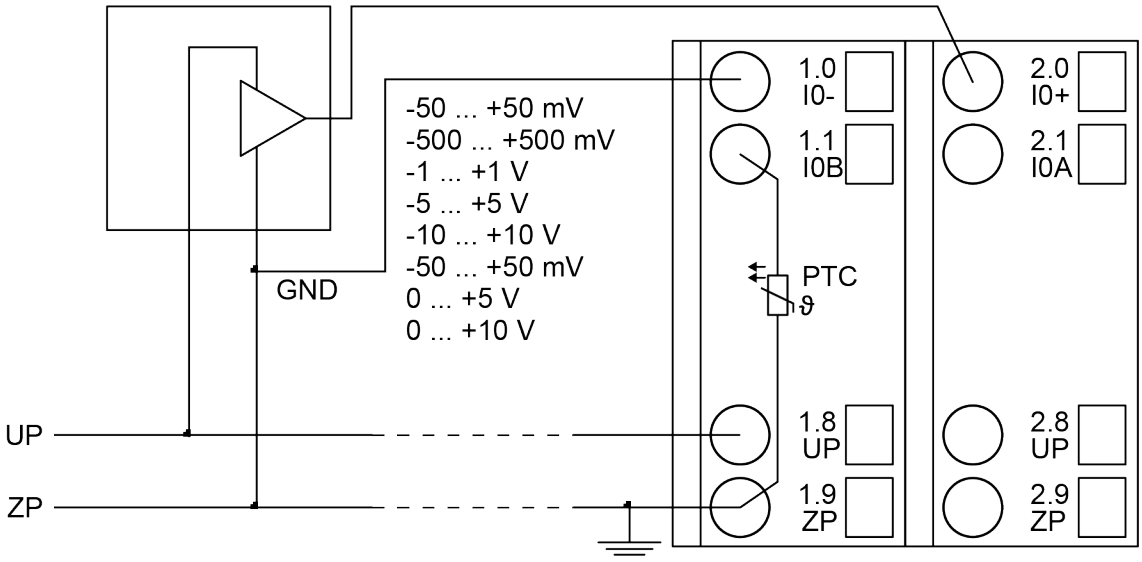


Fig. 134: Connection example

CAUTION!
If GND is not directly connected to ZP at the sensor, the supply current flows via the GND line to ZP. Measuring errors can only occur caused by voltage differences higher than ±20 V DC between GND and ZP.

The measuring ranges can be configured *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899* :

Voltage	-50 mV...+50 mV	1 channel used
Voltage	-500 mV...+500 mV	1 channel used
Voltage	-1 V...+1 V	1 channel used
Voltage	-5 V...+5 V	1 channel used
Voltage	-10 V...+10 V	1 channel used
Voltage	0 V...+5 V	1 channel used
Voltage	0 V...+10 V	1 channel used

Common mode range (+/-20 V)

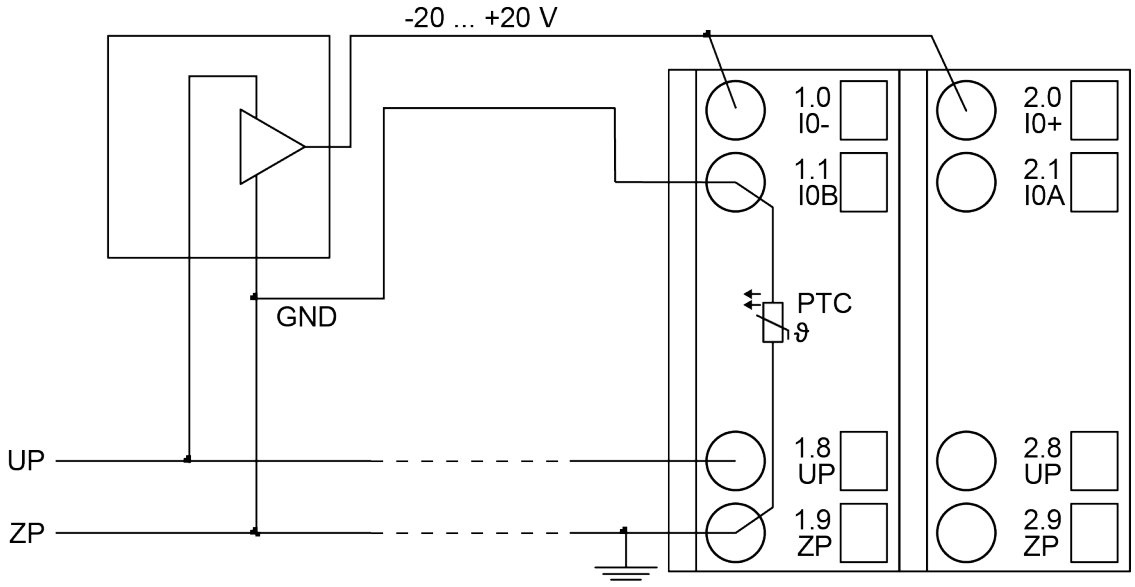


Fig. 135: Connection example

CAUTION!
If GND is not directly connected to ZP at the sensor, the supply current flows via the GND line to ZP. Measuring errors can only occur caused by voltage differences higher than ±20 V DC between GND and ZP.

The measuring range can be configured *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Voltage	Common mode voltage	1 channel used

The function of the LEDs is described under Diagnosis and displays / displays *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

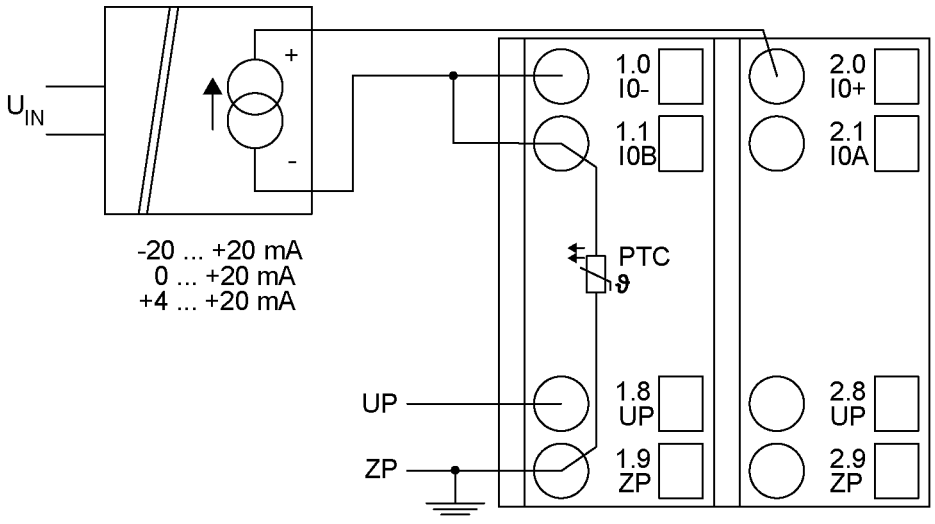


Fig. 136: Connection example

Figure:

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Current	-20 mA...20 mA	1 channel used
Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

Unused input channels can be left open, because they are of low resistance.

Connection of active-type analog sensors (Current) with galvanically isolated power supply and series-connection of an additional input

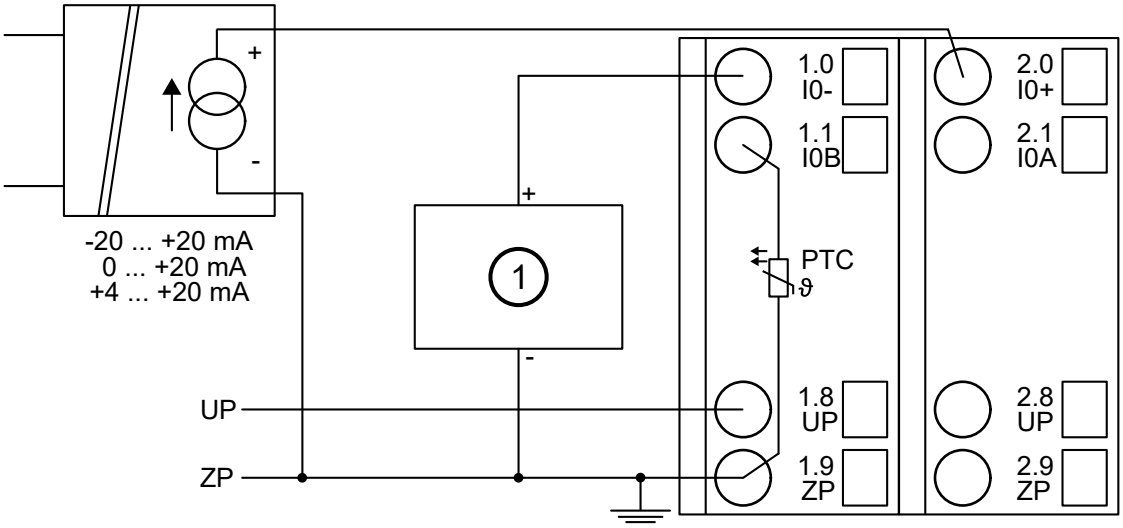


Fig. 137: Connection example

1 Analog input of the second device

> *If series-connection of an additional input is used, the input resistance of the module (ca. 330 Ω) must be added to the input resistance of the second device. Make sure that the maximum permitted load resistance of the analog sensor is not exceeded (see the data sheet of the analog sensor).*

> *The input of the module is not related to ZP. If the input of the second device is related to ZP, the order of sequence in the series-connection must be observed by all means (from the sensor to the module and then to the input of the second device).*

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Current	-20 mA...20 mA	1 channel used
Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

For a description of the functions of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

Unused input channels can be left open, because they are of low resistance.

Connection of passive-type analog sensors (Current)

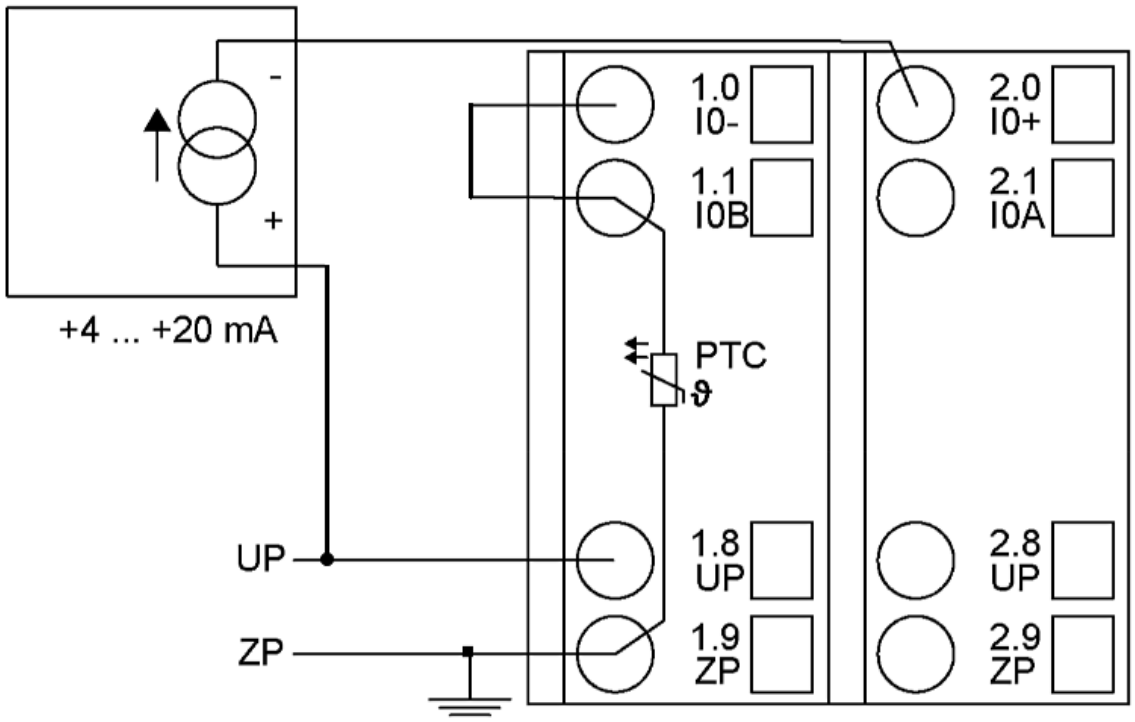


Fig. 138: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Current	-20 mA... 20 mA *)	1 channel used
Current	0 mA... 20 mA *)	1 channel used
Current	4 mA... 20 mA	1 channel used
*) This setting is not applicable with passive-type analog sensors (current).		

The function of the LEDs is described under Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

Unused input channels can be left open, because they are of low resistance.

Connection of passive-type analog sensors (Current) and series-connection of an additional analog sensor

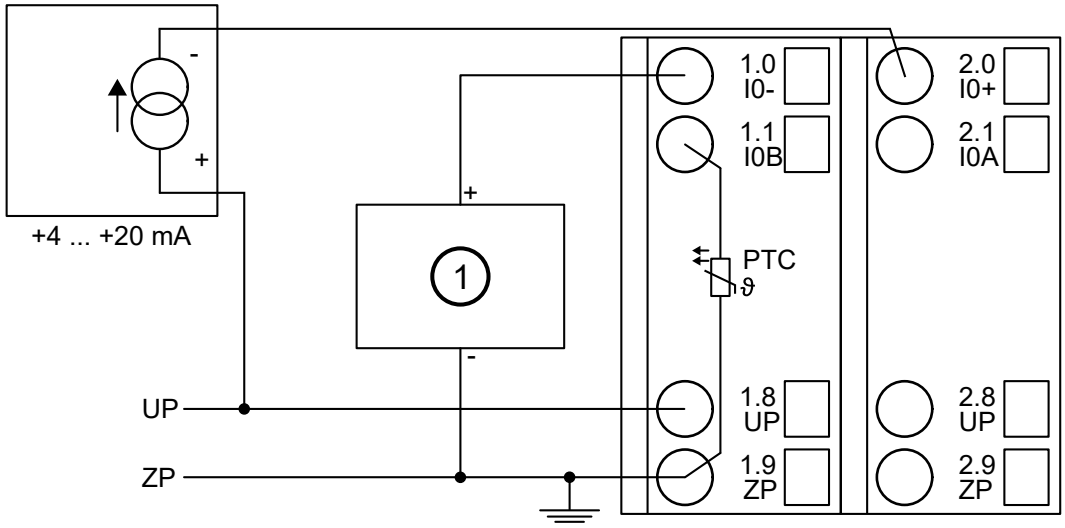


Fig. 139: Connection example

1 Analog input of the second device

> *If series-connection of an additional input is used, the input resistance of the module (ca. 330 Ω) must be added to the input resistance of the second device. Make sure that the maximum permitted load resistance of the analog sensor is not exceeded (see the data sheet of the analog sensor).*

> *The input of the module is not related to ZP. If the input of the second device is related to ZP, the order of sequence in the series-connection must be observed by all means (from the sensor to the module and then to the input of the second device).*

The following measuring ranges can be configured ↬ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Current	-20 mA...20 mA *)	1 channel used
Current	0 mA...20 mA *)	1 channel used
Current	4 mA...20 mA	1 channel used
*) This setting is not applicable with passive-type analog sensors (current).		

The function of the LEDs is described under Diagnosis and displays / displays ↬ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

Unused input channels can be left open, because they are of low resistance.

Connection of digital signal sources at analog inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

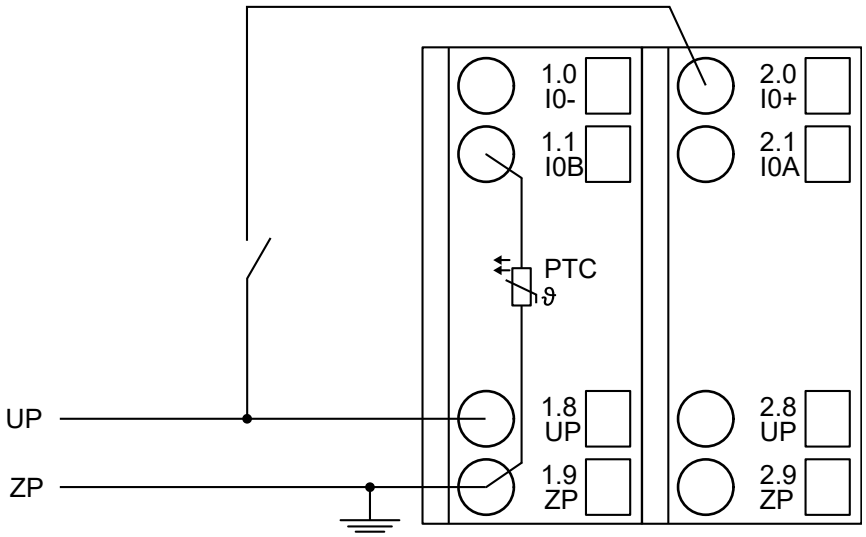


Fig. 140: Connection example

The following operating mode can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899* :

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000, Cu50) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

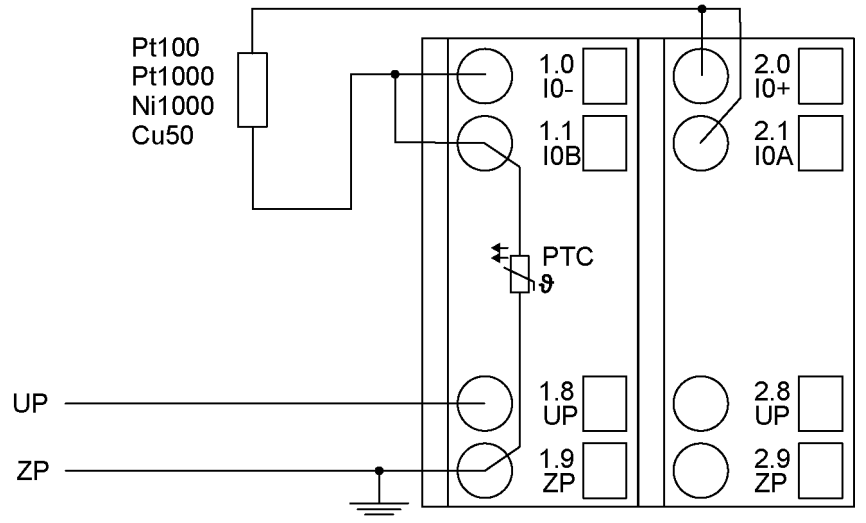


Fig. 141: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Pt100	-50 °C...+70 °C / +400 °C; -200 °C...+850 °C	1 channel used
Pt1000	-50 °C...+400 °C	1 channel used
Ni1000	-50 °C...+150 °C	1 channel used
Cu50	-50 °C...+200 °C (1.426); -200 °C...+200 °C (1.428)	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

The module linearizes the resistance thermometer characteristics.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000, Cu50) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

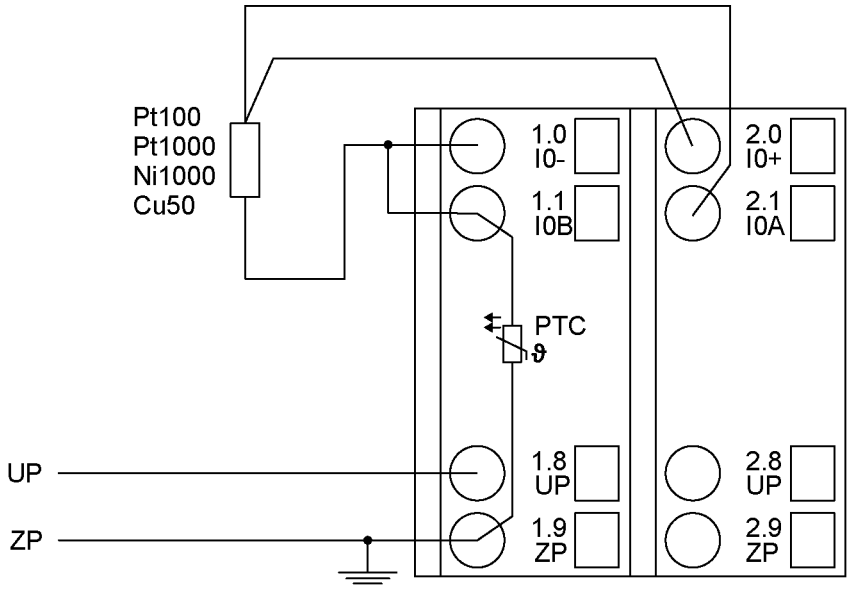


Fig. 142: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*:

Pt100	-50 °C...+70 °C / +400 °C; -200 °C ... +850 °C	1 channel used
Pt1000	-50 °C...+400 °C	1 channel used
Ni1000	-50 °C...+150 °C	1 channel used
Cu50	-50 °C...+200 °C (1.426); -200 °C...+200 °C (1.428)	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

The module linearizes the resistance thermometer characteristics. In order to keep measuring errors as small as possible, it is necessary by all means to have all the involved conductors in the same cable. All the conductors must have the same cross section.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 4-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000, Cu50) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

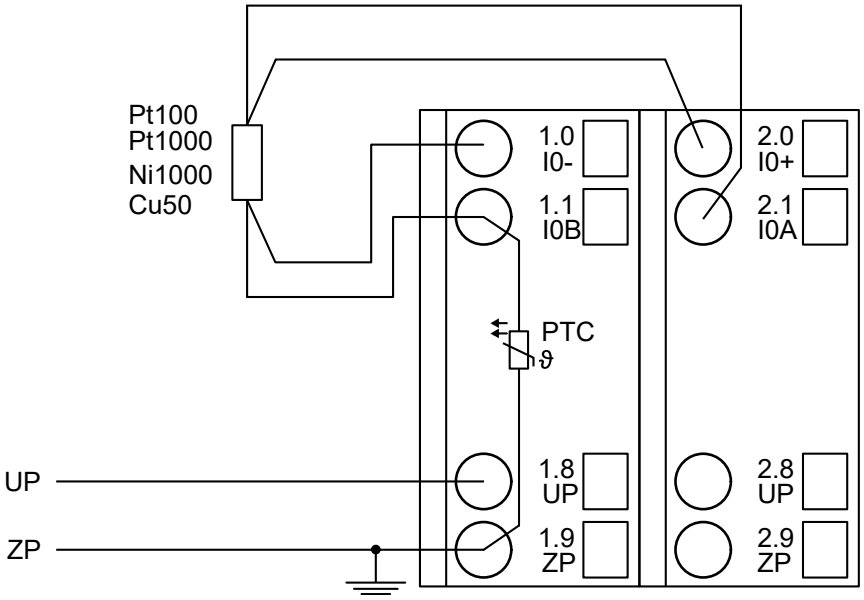


Fig. 143: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 "Parameterization" on page 2899*:

Pt100	-50 °C...+70 °C / +400 °C; -200 °C...+850 °C	1 channel used
Pt1000	-50 °C...+400 °C	1 channel used
Ni1000	-50 °C...+150 °C	1 channel used
Cu50	-50 °C...+200 °C (1.426); -200 °C...+200 °C (1.428)	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 "Diagnosis" on page 2902*.

The module linearizes the resistance thermometer characteristics. In order to keep measuring errors as small as possible, it is necessary by all means, to have all the involved conductors in the same cable.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistors in 2-wire configuration

For evaluating resistors, a constant current must flow through them to build the necessary voltage drop. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

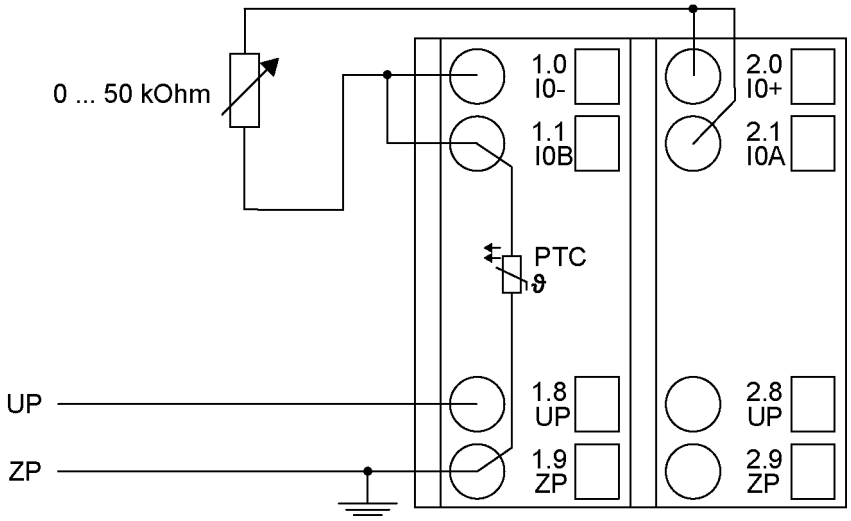


Fig. 144: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899* :

Resistor	50 kΩ	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of a resistance measuring bridge with internal supply

When resistance measuring bridges are connected, the short-circuit-proof voltage output (internal supply) at pin I0A (or I2A, I4A, I6A) must be used. This supply voltage is activated as soon as "Voltage Measurement" is configured for the relevant channel.

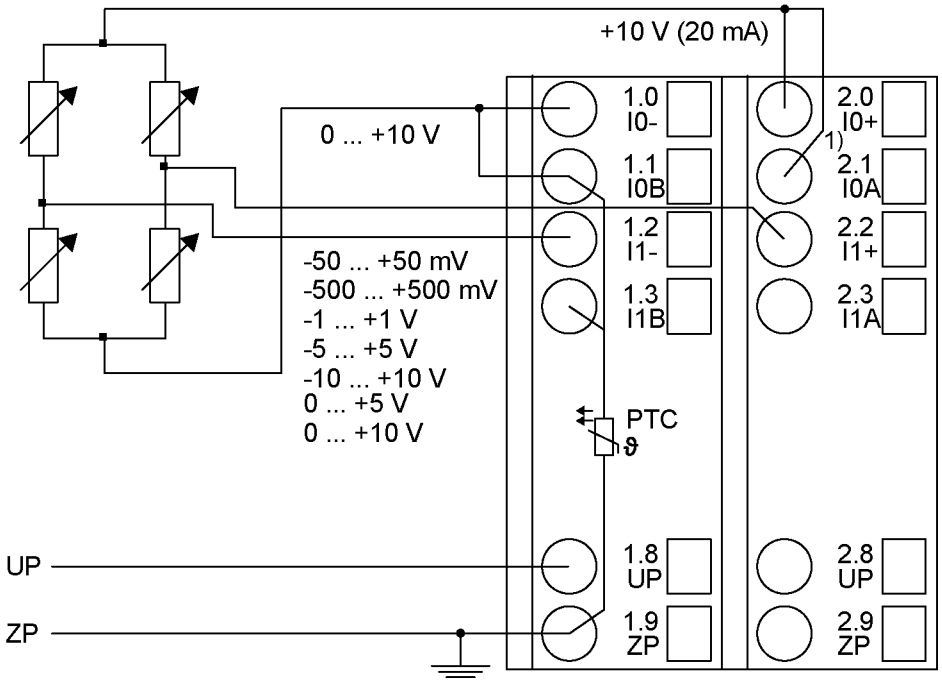


Fig. 145: Connection example

1 Internal supply

All voltage measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899*.

The calculation of the resistor deviation must be performed via the bridge voltage by the PLC user program.

Connection of a resistance measuring bridge with external supply

With the connection of a resistance measuring bridge with external supply, the supply voltage is provided separately.

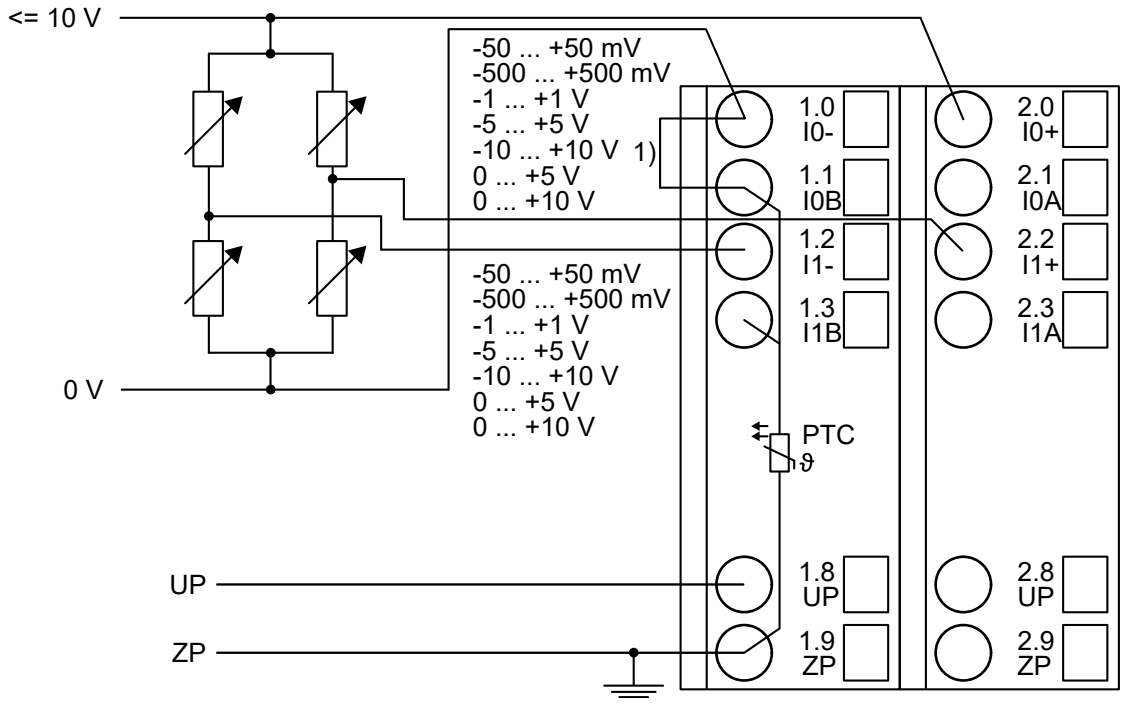


Fig. 146: Connection example

1 Bridge to IxB necessary with galvanically isolated supply

All voltage measuring ranges can be configured ↬ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899* .

The calculation of the resistor deviation must be performed via the bridge voltage by the PLC user program.

Connection of thermocouples

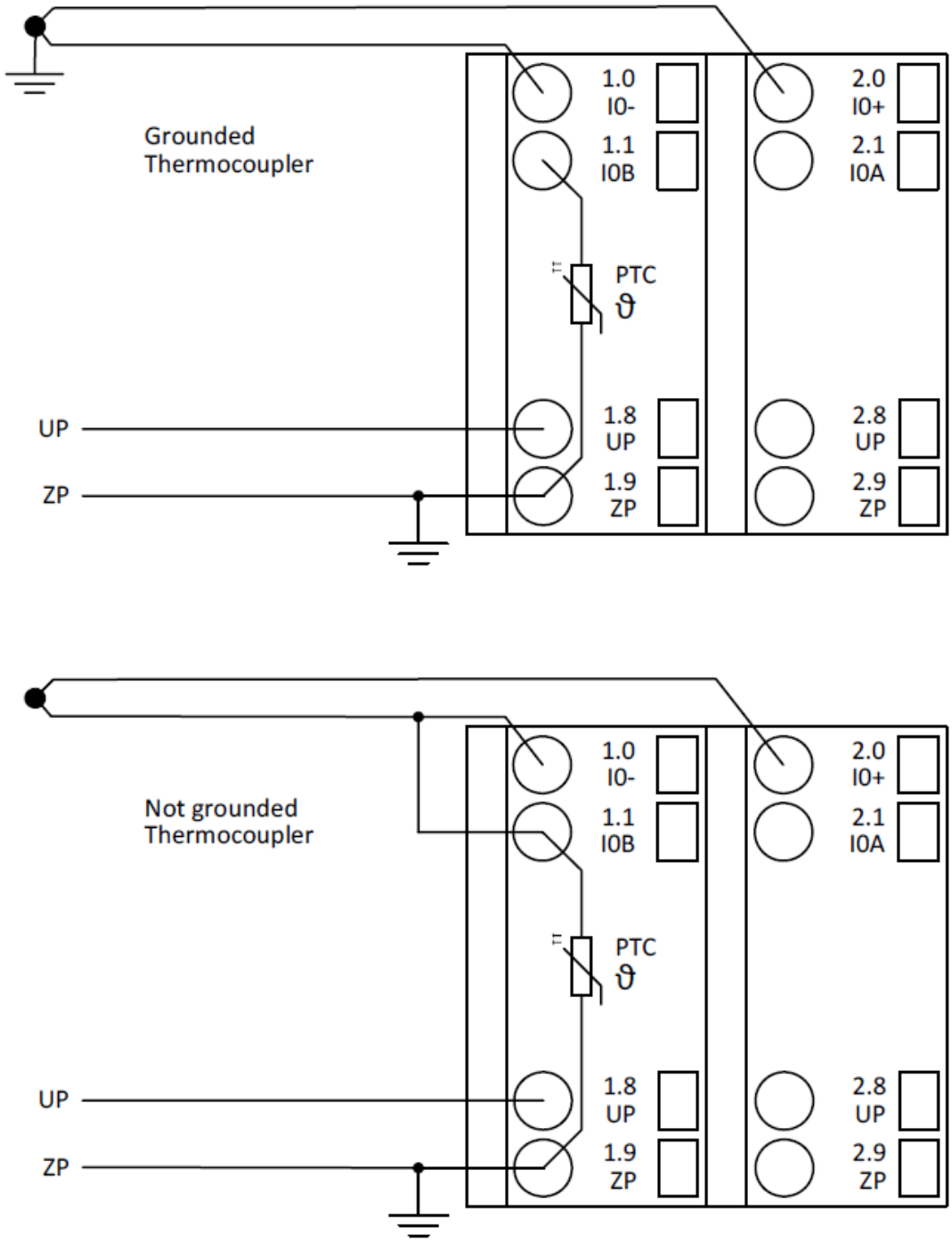


Fig. 147: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.3.6 "Parameterization" on page 2899* :

J type	-210 °C...1200 °C	Fe-CuNi	1 channel used
K type	-270 °C...1372 °C	Ni-CrNi	1 channel used
N type	-270 °C...1300 °C	NiCrSi-NiSi	1 channel used
S type	-50 °C...1768 °C	Pt10Rh-Pt	1 channel used
T type	-270 °C...400 °C	Cu-CuNi	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.3.6.2.2.3.7 “Diagnosis” on page 2902*.

The module linearizes the thermocouple characteristics. It supports the following possibilities of temperature compensation and handling with cold junctions:

Internal compensation

An internal temperature sensor which is located next to the terminal unit is used to detect the temperature of the cold junction. So the compensating cables must be connected directly to the terminal unit, where the cold junction is located.

The setting "Internal compensation (default)" for the parameter "Compensation channel" should be selected.

To get more precise temperature measurements, the use of an external compensation method is recommended.

External compensation with temperature input

The temperature for the cold junction can be determinated externally.

A measured or known temperature value (e.g. ambient temperature in the cabinet) is transferred to the module via the output data word to all required channels. The possible temperature range is from -25 °C to +60 °C and is monitored by the AI531.

The setting "External with temperature value" for the parameter "Compensation channel" should be selected.

External compensation with compensation box

A compensation box balances the temperature difference between the cold junction and the reference temperature by generating a bridge voltage. The reference temperature is transferred via the output data word.

The compensation box must fit to the type of thermocouple and is located at the end of the compensating cables, where the cold junction is located. The cabling to the AI531 can be carried out with normal cables. The operating manual of the compensation box also has to be considered.

The setting "External with temperature value" for the parameter "Compensation channel" should be selected.

External compensation with flanking channel

A flanking channel of the same input group can be used for compensation, e. g. for channel 3, the channels 0, 1 and 2 can be selected as reference channels. The type of sensor for the reference channel can be selected in the parameters for the flanking channel. For example, a RTD sensor which is located next to the thermocouple terminal can be used as reference point for other channels.

The setting "Channel x" for the parameter "Compensation channel" should be selected. Refer to Channel configuration ↳ *Chapter 1.6.3.6.2.2.3.6 “Parameterization” on page 2899* for possible settings.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Analog inputs (words)	8
Analog outputs (words)	1

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

This means that replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1535 [1])	Word	1535 0x05ff	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length in bytes	Internal	36	Byte	36	0	255	0x0Y02
Check supply	Off On	0 1	Byte	On 0x01			0x0Y03
Analog data format	Default	0	Byte	Default 0x00			0x0Y04

[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1

[2]) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	39
Ext_User_Prm_Data_Const(0) =	0x05, 0xff, 0x24, \ 0x01, 0x00, 0x00, 0x00 \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Input channel (8x)

No.	Name	Value	Internal value	Internal value, Type	Default	EDS Slot Index
1	Channel configuration	see *Table 499 "Channel configuration" on page 2900*	see *Table 499 "Channel configuration" on page 2900*	Byte	0 0x00	0x0Y07
2	Channel monitoring	see *Table 500 "Channel monitoring" on page 2902*	see *Table 500 "Channel monitoring" on page 2902*	Byte	0 0x03	
3	Line frequency suppression	see *Further information on page 2902*	see *Further information on page 2902*	Byte	0 0x00	
4	Compensation channel	see *Further information on page 2902*	see *Further information on page 2902*	Byte	0 0x00	

Table 499: Channel configuration

Internal value	Operating modes for the analog inputs, individually configurable
0	Unused (default)
2	Digital input
34	Analog input -50 mV...+50 mV
35	Analog input -500 mV...+500 mV
36	Analog input -1 V...+1 V
7	Analog input -5 V...+5 V
5	Analog input -10 V...+10 V
6	Analog input 0 V...+5 V

Internal value	Operating modes for the analog inputs, individually configurable
1	Analog input 0 V...+10 V
37	Analog input -20 mA...+20 mA
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
14	Analog input Pt100 (2-wire), -50 °C...+70 °C
15	Analog input Pt100 (3-wire), -50 °C...+70 °C
48	Analog input Pt100 (4-wire), -50 °C...+70 °C
57	Analog input Pt100 (2-wire), -50 °C...+70 °C (resolution: 0,01 K)
58	Analog input Pt100 (3-wire), -50 °C...+70 °C (resolution: 0,01 K)
59	Analog input Pt100 (4-wire), -50 °C...+70 °C (resolution: 0,01 K)
8	Analog input Pt100 (2-wire), -50 °C...+400 °C
9	Analog input Pt100 (3-wire), -50 °C...+400 °C
49	Analog input Pt100 (4-wire), -50 °C...+400 °C
45	Analog input Pt100 (2-wire), -200 °C...+850 °C
46	Analog input Pt100 (3-wire), -200 °C...+850 °C
47	Analog input Pt100 (4-wire), -200 °C...+850 °C
16	Analog input Pt1000 (2-wire), -50 °C...+400 °C
17	Analog input Pt1000 (3-wire), -50 °C...+400 °C
50	Analog input Pt1000 (4-wire), -50 °C...+400 °C
18	Analog input Ni1000 (2-wire), -50 °C...+150 °C
19	Analog input Ni1000 (3-wire), -50 °C...+150 °C
51	Analog input Ni1000 (4-wire), -50 °C...+150 °C
39	Analog input Cu50 1.426 (2-wire) -50 °C...+200 °C
40	Analog input Cu50 1.426 (3-wire) -50 °C...+200 °C
41	Analog input Cu50 1.426 (4-wire) -50 °C...+200 °C
42	Analog input Cu50 1.428 (2-wire) -200 °C...+200 °C
43	Analog input Cu50 1.428 (3-wire) -200 °C...+200 °C
44	Analog input Cu50 1.428 (4-wire) -200 °C...+200 °C
24	Analog input J-type thermocouple -210 °C...+1200 °C
25	Analog input K-type thermocouple -270 °C...+1372 °C
30	Analog input N-type thermocouple -270 °C...+1300 °C
27	Analog input S-type thermocouple -50 °C...+1768 °C
28	Analog input T-type thermocouple -270 °C...+400 °C
38	Analog input resistor 50 kΩ
52	Temperature-internal reference point
53	Common mode voltage

Table 500: Channel monitoring

Internal value	Monitoring
0	Plausibility, open-circuit (cut wire) and short circuit (default)
3	No monitoring

Table 501: Line frequency suppression

Internal value	Line frequency suppression
0	50 Hz
1	60 Hz
2	No line frequency suppression

Table 502: Compensation channel

Internal value	Compensation channel
0	Internal compensation (default)
1	Channel 0 (possible with channels 1, 2, 3)
2	Channel 1 (possible with channels 0, 2, 3)
3	Channel 2 (possible with channels 0, 1, 3)
4	Channel 3 (possible with channels 0, 1, 2)
5	Channel 4 (possible with channels 5, 6, 7)
6	Channel 5 (possible with channels 4, 6, 7)
7	Channel 6 (possible with channels 4, 5, 7)
8	Channel 7 (possible with channels 4, 5, 6)
9	External with temperature value

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module, e.g. internal analog voltage is not correct	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched OFF (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...7	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...7	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				
4	14	1...10	1	0...7	47	Short circuit at an analog input	Check ter-minal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...7	1	Possibly wrong meas-ured value caused by inadmissible temper-ature of the compensa-tion channel	Check the tempera-ture com-pensation channel
	11 / 12	ADR	1...10				

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
4	14	1...10	1	0...7	2	Invalid measured value of the channel caused by overly high voltage difference		Check voltage difference; install equalizing conductors if necessary
	11 / 12	ADR	1...10					
4	14	1...10	1	0...7	11	Output voltage 10 V faulty		Check output load
	11 / 12	ADR	1...10					

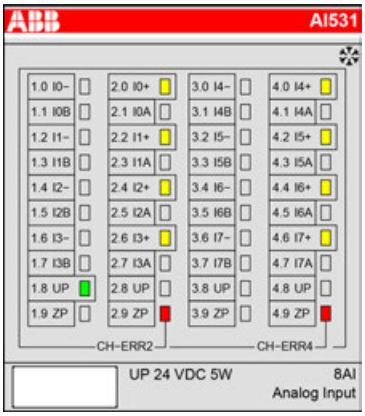
Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

States of the LEDs (see also section Diagnosis LEDs in the S500 system data):

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I3 and I4...I7	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, messages in groups (analog inputs combined into the groups 2 and 4)	Red	No error, or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs CH-ERR2 and CH-ERR4 light up together					

Measuring ranges

Voltage input ranges

Bipolar voltage input range, measuring bridge

The represented resolution corresponds to 16 bits.

Range	-50 ... +50 mV	-500 ... +500 mV	-1 ... +1 V	-5 ... +5 V	-10 ... +10 V	Common Mode Voltage	Digital value	
							Decimal	Hex.
Over-flow	> 58.7945	> 587.9449	> 1.17589	> 5.8794	> 11.7589	> 20.0000	32767	7FFF
Meas-ured value too high	58.7945 : 50.0018	587.9449 : 500.0181	1.17589 : 1.00004	5.8794 : 5.0002	11.7589 : 10.0004		32511 : 27649	7EFF : 6C01
Normal range Normal range or Meas-ured value too low	50.0000 : 0.0018	500.0000 : 0.0181	1.00000 : 0.00004	5.0000 : 0.0002	10.0000 : 0.0004	20.0000 : 0.0008	27648 : 1	6C00 : 0001
	0.0000	0.0000	0.0000	0.00000	0.0000	0.0000	0	0000
	-0.0018 : -50.0000	-0.0181 : -500.0000	-0.00004 : -1.00000	-0.0002 : -5.0000	-0.004 : -10.0000	-0.0008 : -20.0000	-1 : -27648	FFFF : 9400

Range	-50 ... +50 mV	-500 ... +500 mV	-1 ... +1 V	-5 ... +5 V	-10 ... +10 V	Common Mode Voltage	Digital value	
							Decimal	Hex.
Measured value too low	-50.0018 : -58.7945	-500.0181 : -587.9449	-1.00004 : -1.17589	-5.0002 : -5.8794	-10.0004 : -11.7589		-27649 : -32512	93FF : 8100
Underflow	< -58.7945	< -587.9449	< -1.17589	< -5.8794	< -11.7589	< -20.0000	-32768	8000

Unipolar voltage input range, measuring bridge, digital input

Range		0 ... +5 V	0 ... +10 V	Digital input	Digital value	
					Decimal	Hex.
Measured value too high		5.8794 : 5.0002	11.7589 : 10.0004		32511 : 27649	7EFF : 6C01
Normal range		5.0000 : 0.0002	10.0000 : 0.0004	ON	27648 : 1	6C00 : 0001
		0.0000	0.0000	OFF	0	0000
Measured value too low		-0.0002 : -0.8794	-0.0004 : -1.1759		-1 : -4864	FFFF : ED00
Underflow		< -0.8794	< -1.1759		-32768	8000

Current input ranges

Range	-20 ... +20 mA	0 ... +20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Overflow	> 23.5178	> 23.5178	> 22.8142	32767	7FFF
Measured value too high	23.5178 : 20.0007	23.5178 : 20.0007	22.8142 : 20.0006	32511 : 27649	7EFF : 6C01
Normal range	20.0000 : 0.0007	20.0000 : 0.0007	20.0000 : 4.0006	27648 : 1	6C00 : 0001
	0.0000	0.0000	4.0000	0	0000
	-0.0007 : -20.0000			-1 : -27648	FFFF : 9400

Range	-20 ... +20 mA	0 ... +20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Measured value too low		-0.0007 : -3.5178	3.9994 : 1.1852	-1 : -4864	FFFF : ED00
	-20.0007 : -23.5178			-27649 : -32512	93FF : 8100
Underflow	< -23.5178	< -3.5178	< 1.1852	-32768	8000

Resistance thermometer input ranges

The represented resolution corresponds to 16 bits.

Range	Pt100 -50 ... +70 °C [1])	Pt100 / Pt1000 -50 ... +400 °C	Pt100 -200 ... +850 °C	Ni1000 -50 ... +150 °C	Cu50 -200 ... +200 °C	Digital value	
						Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 850 °C	> 160.0 °C	> 200 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C				4500 : 4001	1194 : 0FA1
				160.0 °C : 150.1 °C		1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C					800 : 701	0320 : 02BD
Normal range	: : : : 70.0 °C : 0.1 °C	: 400.0 °C : : : : 0.1 °C	850.0 °C : : : : : 0.1 °C	: : : 150.0 °C : : 0.1 °C	: : 200.0 °C : : : 0.1 °C	8500 4000 2000 1500 700 : 1	2134 0FA0 07D0 05DC 02BC : 1
	0.0 °C	0.0 °C	0.0 °C	0.0 °C		0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : : -200 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C [2]) -200.0 °C [2])	-1 : -500 -2000	FFFF : FE0C F830

Range	Pt100 -50 ... +70 °C [1])	Pt100 / Pt1000 -50 ... +400 °C	Pt100 -200 ... +850 °C	Ni1000 -50 ... +150 °C	Cu50 -200 ... +200 °C	Digital value	
						Decimal	Hex.
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C		-50.1 °C : -60.0 °C		-501 : -600	FE0B : FDA8
Under-flow	< -60.0 °C	< -60.0 °C	< -200 °C	< -60.0 °C	< -200 °C [2])	-32768	8000

[1]) also possible with resolution 0.01 K

[2]) if Cu50 with 1.426, -50 °C is valid; if Cu50 with 1.428, -200.0 °C is valid

Resistor input range

The represented resolution corresponds to 16 bits.

Range	Resistor [Ω]	Digital value	
		Decimal	Hex.
Overflow	> 55000	32767	7FFF
Measured value too high	55000 : 50001	30413 : 27649	76CD : 6C01
Normal range	50000 : 2 1 0	27648 : 1 1 0	6C00 : 0001 0001 0000

Thermocouple input ranges

The represented resolution corresponds to 16 bits.

Range	Typ J -210 ... +1200 °C	Typ K -270 ... +1372 °C	Typ N -270 ... +1300 °C	Typ S -50 ... +1768 °C	Typ T -270 ... +400 °C	Digital value	
						Decimal	Hex.
Overflow	> 1200.0 °C	> 1372.0 °C	> 1300.0 °C	> 1768.0 °C	> 400.0 °C	32767	7FFF
Normal range				1768.0 °C		17680	4510
		1372.0 °C		:		13720	3598
		:	1300.0 °C	:		13000	32C8
	1200.0 °C	:	:	:		12000	2EE0
	:	:	:	:	400.0 °C	4000	0FA0
	:	:	:	:	:	:	:

Range	Typ J -210 ... +1200 °C	Typ K -270 ... +1372 °C	Typ N -270 ... +1300 °C	Typ S -50 ... +1768 °C	Typ T -270 ... +400 °C	Digital value	
						Decimal	Hex.
	0.1 °C	0.1 °C	0.1 °C	0.1 °C	0.1 °C	1	1
	0.0 °C	0.0 °C	0.0 °C	0.0 °C		0	0000
	-0.1 °C	-0.1 °C	-0.1 °C	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:	:	:
	:	:	:	-50.0 °C	:	-500	FE0C
	-210.0 °C	:	:	:	:	-2100	F7CC
		-270.0 °C	-270.0 °C		-270.0 °C	-2700	F574
Under-flow	< -210.0 °C	< -270.0 °C	< -270.0 °C	< -50.0 °C	< -270.0 °C	-32768	8000

Temperature-internal reference point ranges

Range	Value	Digital value	
		Decimal	Hex.
Overflow	> +85 °C	32767	7FFF
Normal range	+85 °C	850	0352
	0 °C	0	0000
	-40 °C	-400	FE70
Underflow	< -40 °C	-32768	8000

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		

Parameter		Value
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	Current consumption from UP in normal operation	130 mA
	Inrush current from UP (at power up)	0.056 A^2s
Max. length of analog cables, conductor cross section > 0.14 mm^2		100 m
Weight		130 g
Mounting position		Horizontal or vertical with derating (max. temperature 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value
Number of channels per module	8
Distribution of channels into groups	2 groups of 4 channels each
Connections of the channels I0 to I3	Terminals 1.0 to 1.7 and terminals 2.0 to 2.7
Connections of the channels I4 to I7	Terminals 3.0 to 3.7 and terminals 4.0 to 4.7
Input type	Bipolar (not with current or Pt100/ Pt1000/ Ni1000/ Cu50/ resistor)
Galvanic isolation	Against internal supply and other modules
Common mode input range	±20 V DC plus signal voltage
Configurability	Digital input, -50 mV...+50 mV, -500mV...+500 mV, -1 V...+1 V, -5 V...+5 V, -10 V...+10 V, 0 V...+5 V, 0 V...+10 V, -20 mA...+20 mA, 0 mA...20 mA, 4 mA...20 mA, Pt100, Pt1000, Ni1000, Cu50, resistor, thermocouple types J, K, N, S, T (each input can be configured individually)
Channel input resistance	Voltage: > 100 kΩ, current: ca. 330 Ω
Time constant of the input filter	Line-frequency suppression 50 Hz, 60 Hz, none
Indication of the input signals	1 yellow LED per channel, the brightness depends on the value of the analog signal

Parameter		Value		
Conversion time		1 ms (none), 100 ms (50 Hz / 60 Hz) per channel		
Resolution		Range	unipolar	15 bits
			bipolar	15 bits + sign
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.1 % (voltage) ±0.3 % (current, resistor) at 25 °C	
		Max.	±0.7 % (voltage) ±0.9 % (current, resistor) ±0.5 % (thermocouple type J, N, S, T; thermocouple type K > -220 °C) 1.0 K (resistance temperature detectors) at 0 °C...60 °C or EMC disturbance	
Maximum permanent allowed overload (no damage)				
	Current input	When the input current exceeds the overflow value of the measurement range, the input impedance is switched to high impedance for protection. The maximum allowed overload is then 30 V. The digital value corresponds to the overflow value. Periodically, the input impedance is switched to the normal value and the input current is measured. If the input current is within the measurement range, the input impedance remains at the normal level and the digital value corresponds to the measured current.		
	Voltage input	30 V		
Relationship between input signal and hex code		*Table 500 “Channel monitoring” on page 2902*		
Unused voltage inputs		Are configured as "unused"		
Unused current inputs		Have a low resistance, can be left open-circuited		
Overvoltage protection		Yes		

Technical data of the analog inputs if used as digital inputs

Parameter	Value
Number of channels per module	Max. 8
Distribution of channels into groups	2 groups of 4 channels each
Connections of the channels I0+ to I3+	Terminals 2.0, 2.2, 2.4, 2.6
Connections of the channels I4+ to I7+	Terminals 4.0, 4.2, 4.4, 4.6
Reference potential for the inputs	Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)

Parameter		Value
Input delay		Typ. 2 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	Typ. 1 mA
	Input voltage +15 V	Typ. 3.1 mA
	Input voltage +30 V	< 7 mA
Input resistance		Ca. 4.8 kΩ

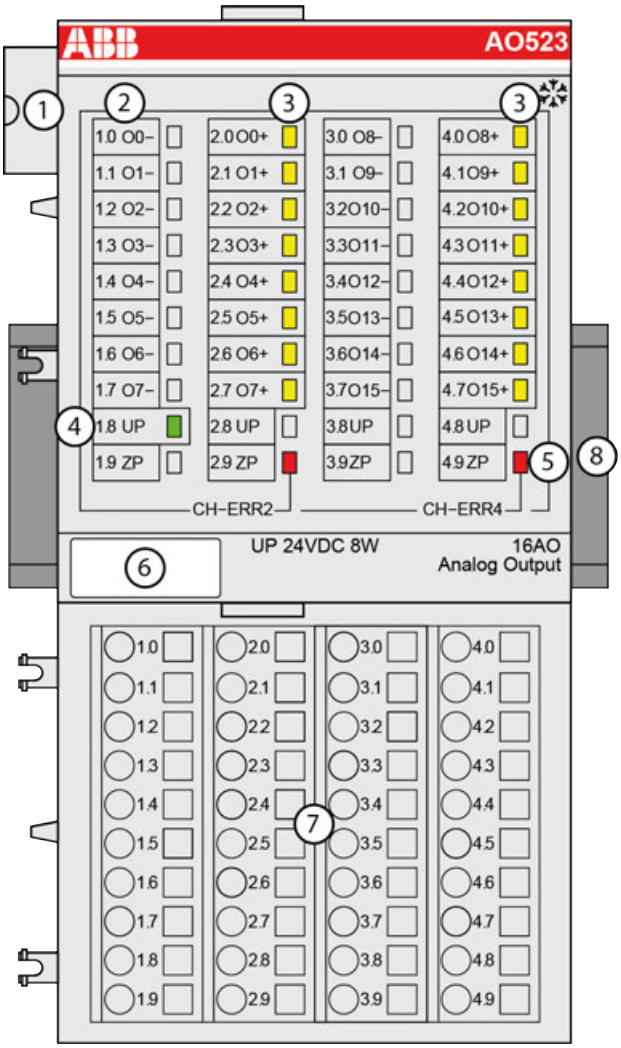
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 600 R0001	AI531, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires	Active
1SAP 450 600 R0001	AI531-XC, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AO523 - Analog output module

- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states at the analog outputs (O0 - O15)
4 1 green LED to display the state of the process supply voltage UP
5 2 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

- 16 analog outputs in two groups:
  - 8 channels configurable for voltage or currrent output (O0...O3 / O8...O11)
  - 8 channels for voltage output (O4...O7 / O12...O15)

  Resolution 12 bits plus sign

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V...+10 V	12 bits plus sign
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
LED displays		19 LEDs for signals and error messages
Internal power supply		Through the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.

The modules are plugged on an I/O terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal units and have always the same assignment, independent of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	O0- to O7-	Negative poles of the first 8 analog outputs
2.0 to 2.7	O0+ to O7+	Positive poles of the first 8 analog outputs
3.0 to 3.7	O8- to O15-	Negative poles of the following 8 analog outputs
4.0 to 4.7	O8+ to O15+	Positive poles of the following 8 analog outputs

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per AO523.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the module:

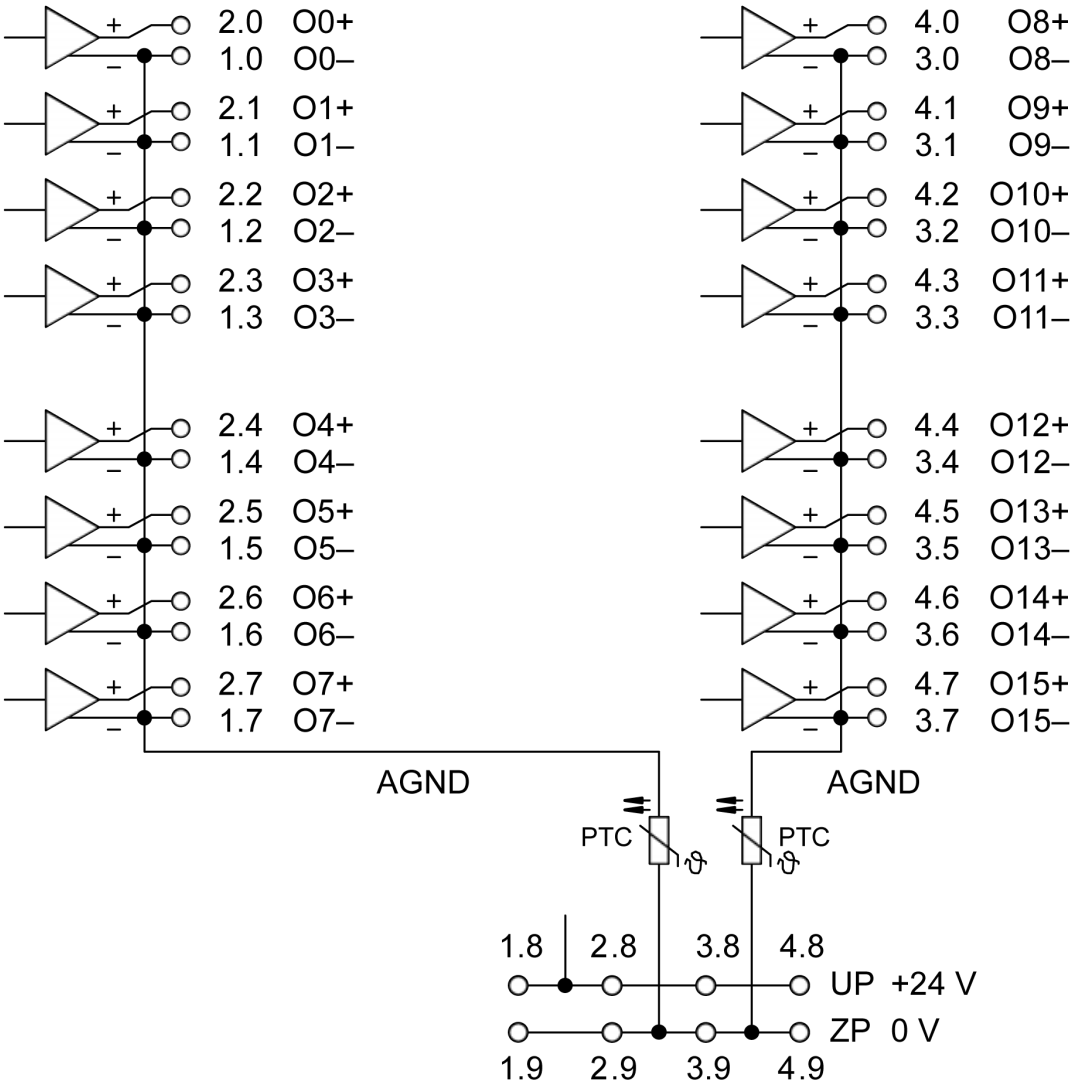


Fig. 148: 16 analog outputs in two groups *Chapter 1.6.3.6.2.2.4.2 “Functionality” on page 2913*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The modules provide several diagnosis functions *Chapter 1.6.3.6.2.2.4.7 “Diagnosis” on page 2922*.

Connection of analog output loads (Voltage, current)

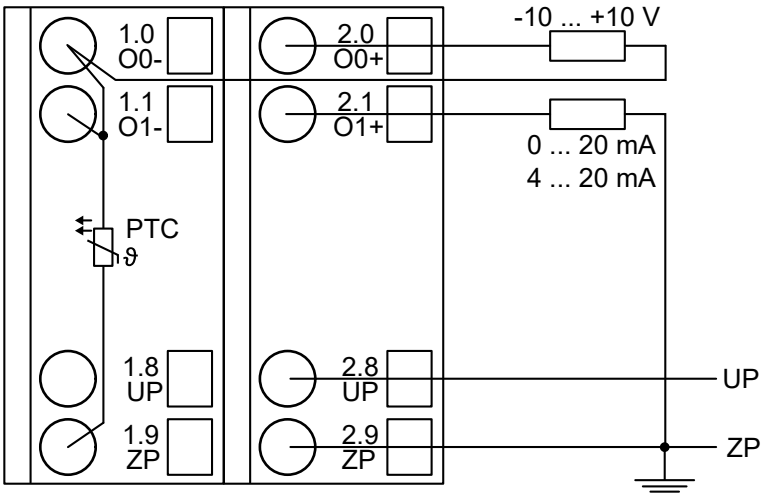


Fig. 149: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.2.2.4.6 “Parameterization” on page 2918*:

Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 and 8...11 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

The function of the LEDs is described under Displays.

Unused analog outputs can be left open-circuited.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	0
Counter output data (words)	16

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

That means replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1510 [1])	Word	1510 0x05e6	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
3	Parameter length in bytes	Internal	39	Byte	39-CPU 39-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
7	Channel configuration Output channel 0	See *Table 503 "Channel configuration [3])" on page 2921*		Byte	Default 0x00	0	130	0x0Y06
8	Channel monitoring Output channel 0	See *Table 504 "Channel monitoring [4])" on page 2921*		Byte	Default 0x00	0	3	0x0Y07

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
9	Substitute value Output channel 0	Output channel 0!	0...0xffff	Word	Default 0x0000	0	65535	0x0Y08
10 to 15	Channel configuration and channel monitoring of the output channels 1 to 3	See *Table 503 “Channel configuration 3)” on page 2921* and *Table 504 “Channel monitoring 4)” on page 2921*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y09 to 0x0Y0E
16 to 23	Channel configuration and channel monitoring of the output channels 4 to 7	See *Table 503 “Channel configuration 3)” on page 2921* and *Table 504 “Channel monitoring 4)” on page 2921*		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y0F to 0x0Y16
24	Channel configuration Output channel 8	See *Table 503 “Channel configuration 3)” on page 2921*		Byte	Default 0x00	0	130	0x0Y17
25	Channel monitoring Output channel 8	See *Table 504 “Channel monitoring 4)” on page 2921*		Byte	Default 0x00	0	3	0x0Y18
26	Substitute value Output channel 8	Output channel 8!	0...0xffff	Word	Default 0x0000	0	65535	0x0Y19

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
27 to 32	Channel configuration and channel monitoring of the output channels 9 to 11	See ↳ *Table 503 “Channel configuration 3)” on page 2921* and ↳ *Table 504 “Channel monitoring 4)” on page 2921*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y1A to 0x0Y1F
33 to 40	Channel configuration and channel monitoring of the output channels 12 to 15	See ↳ *Table 503 “Channel configuration 3)” on page 2921* and ↳ *Table 504 “Channel monitoring 4)” on page 2921*		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y20 to 0x0Y27
[1]) With CS31 and addresses less than 70 and FBP, the value is increased by 1 [2]) Not with FBP								

GSD file:

Ext_User_Prm_Data_Len =	42
Ext_User_Prm_Data_Const(0) =	0x05, 0xe7, 0x27, \ 0x01, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Output channels 0 and 8 (2 channels, AO523)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see below ↳ *Table 503 “ Channel configuration [3])” on page 2921*	see below ↳ *Table 503 “ Channel configuration [3])” on page 2921*	Byte	see below ↳ *Table 503 “ Channel configuration [3])” on page 2921*
2	Channel monitoring	see below ↳ *Table 504 “ Channel monitoring [4])” on page 2921*	see below ↳ *Table 504 “ Channel monitoring [4])” on page 2921* *8)	Byte	see below ↳ *Table 504 “ Channel monitoring [4])” on page 2921*
3	Substitute value ↳ *Table 505 “ Substitute value” on page 2922*	0...65535	0... 0xffff	Word	0

Output channels 1...7 and 9...15 (14 channels, AO523)

No.	Name	Internal value, type
1	Channel configuration see table [3])	Byte
2	Channel monitoring see table [4])	Byte

Table 503: Channel configuration [3])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V
129	Analog output 0 mA...20 mA (not with the channels 4...7 and 12...15)
130	Analog output 4 mA...20 mA (not with the channels 4...7 and 12...15)

Table 504: Channel monitoring [4])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit (default)
1	Open-circuit (broken wire) and short circuit
2	Plausibility
3	No monitoring

Table 505: Substitute value

Intended behavior of channel 0 when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	OFF	0
Last value	Last value	0
Substitute value	OFF or Last value	1...65535

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON -> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	3	0...15	48	Analog value overflow at an analog output		Check output value

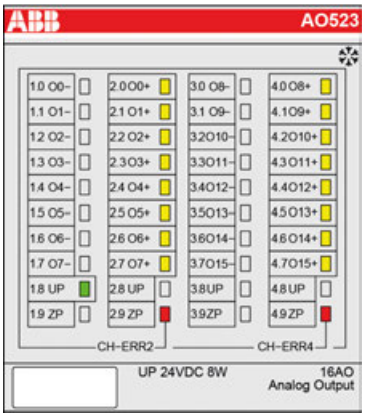
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
	11 / 12	ADR	1...10					
4	14	1...10	3	0...15	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Outputs O0...O7 and O8...O15	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Output ranges

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	> 11.7589 V	> 23.5178 mA	> 22.8142 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	Current consumption from UP at normal operation	0.15 A + output loads
	Inrush current from UP (at power up)	0.040 A^2s
Max. length of analog cables, conductor cross section > 0.14 mm^2		100 m
Weight		300 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog outputs

Parameter		Value	
Number of channels per module		16, of which channnels O0...O3 and O8...O11 for voltage and current, and channels O4...7 and O12...15 only for voltage	
Distribution of channels into groups		2 groups of 8 channels each	
	Channels O0-...O7-	Terminals 1.0...1.7	
	Channels O0+...O7+	Terminals 2.0...2.7	
	Channels O8-...O15-	Terminals 3.0...3.7	
	Channels O8+...O15+	Terminals 4.0...4.7	
Output type		Bipolar with voltage, unipolar with current	
Galvanic isolation		Against internal supply and other modules	
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3 and 8...11	
Output resistance (load), as current output		0 Ω...500 Ω	
Output loadability, as voltage output		Max. ±10 mA	
Indication of the output signals		One LED per channel	
Resolution		12 bits (+ sign)	
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C	
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance	
Relationship between output signal and hex code		*Chapter 1.6.3.6.2.2.4.9 “Output ranges” on page 2924*	
Unused outputs		Can be left open-circuited	

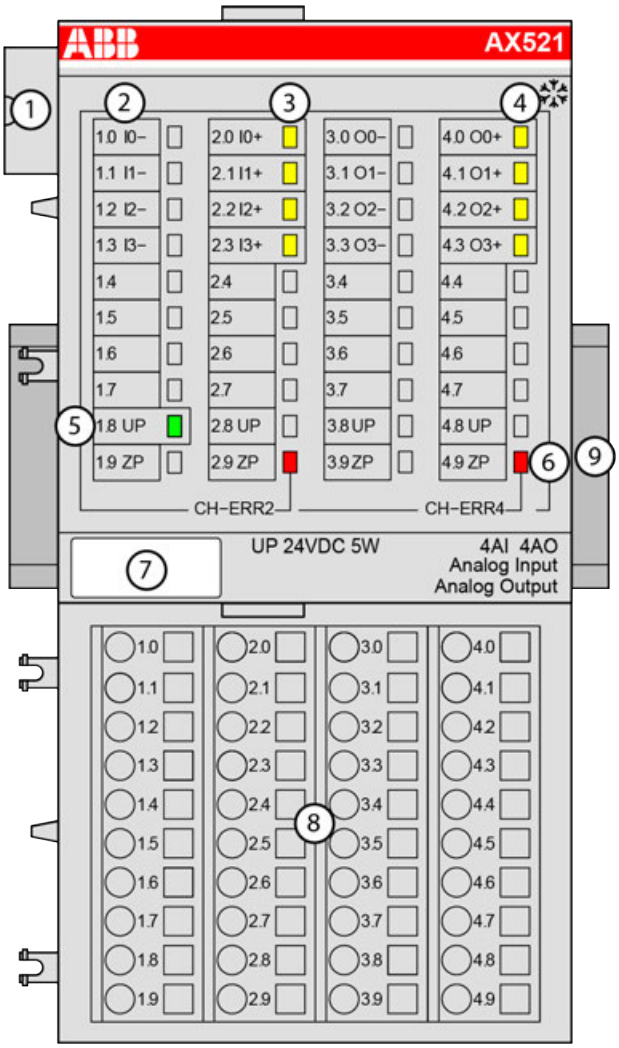
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 200 R0001	AO523, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires	Active
1SAP 450 200 R0001	AO523-XC, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AX521 - Analog input/output module

- 4 configurable analog inputs (I0 to I3) in 1 group (1.0...2.3)
  Resolution 12 bits plus sign
- 4 configurable analog outputs (O0 to O3) in 1 group (3.0...4.3)
  Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 4 yellow LEDs to display the signal states at the analog inputs (I0 - I3)
4 4 yellow LEDs to display the signal states at the analog outputs (O0 - O3)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

AX521

4 analog inputs (I0...I3), individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA
- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

4 analog outputs (O0...O3), individually configurable for

- Unused (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V... +10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		11 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The modules are plugged on an I/O terminal unit *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8 and 4.8 as well as 1.9, 2.9, 3.9 and 4.9 are electrically interconnected within the I/O terminal units and have always the same assignment, irrespective of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.3	I0- to I3-	Negative poles of the 4 analog inputs
2.0 to 2.3	I0+ to I3+	Positive poles of the 4 analog inputs
3.0 to 3.3	O0- to O3-	Negative poles of the 4 analog outputs
4.0 to 4.3	O0+ to O3+	Positive poles of the 4 analog outputs

The negative poles of the analog inputs are connected to each other to form an "Analog Ground" signal for the module.

The negative poles of the analog outputs are connected to each other to form an "Analog Ground" signal for the module.

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per I/O module.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the I/O module.

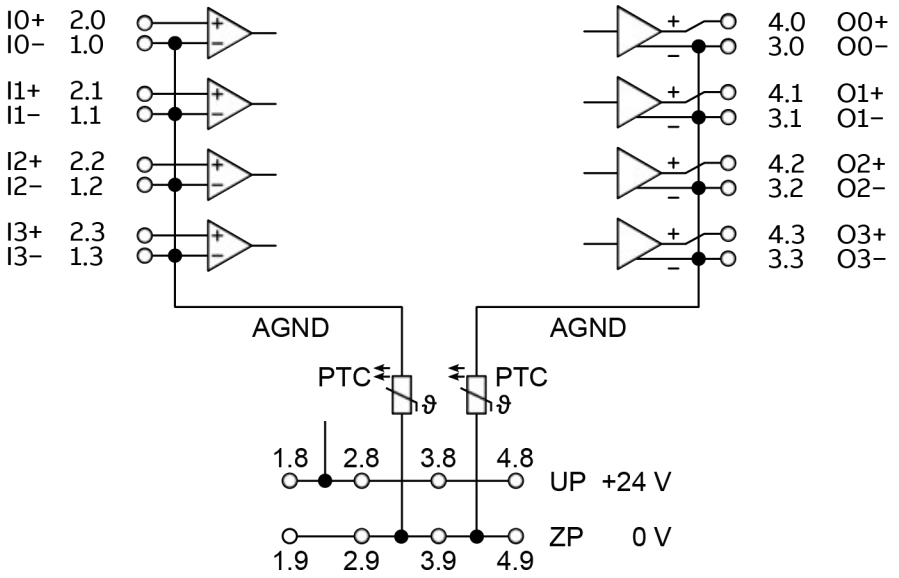


Fig. 150: 4 analog inputs and 4 analog outputs, individually configurable ↬ *Chapter 1.6.3.6.2.2.5.2 “Functionality” on page 2928*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the 8 analog channels.

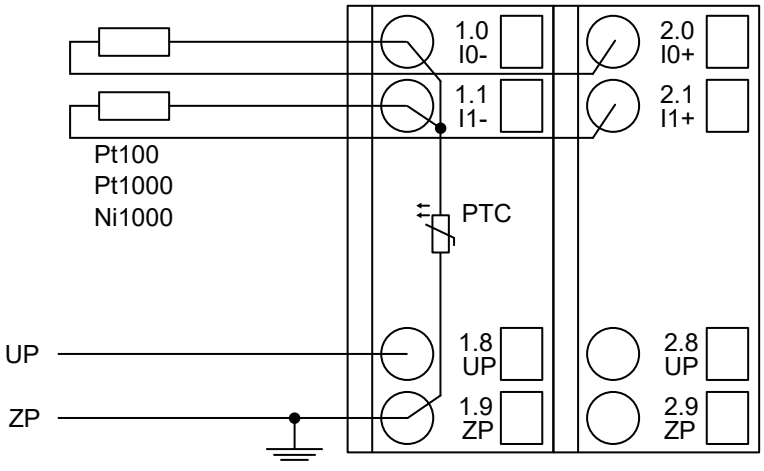


Fig. 151: Connection example

Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.

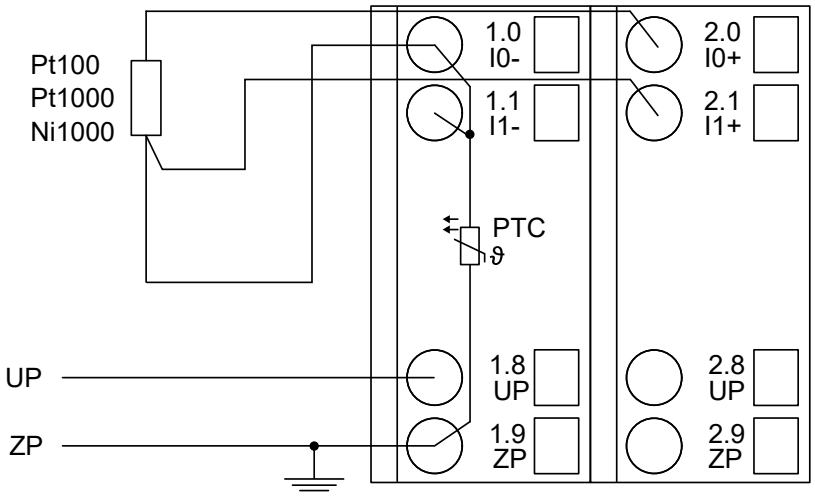


Fig. 152: Connection example

If several measuring points are adjacent to each other, only one return line is necessary. This saves wiring costs.

With the 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

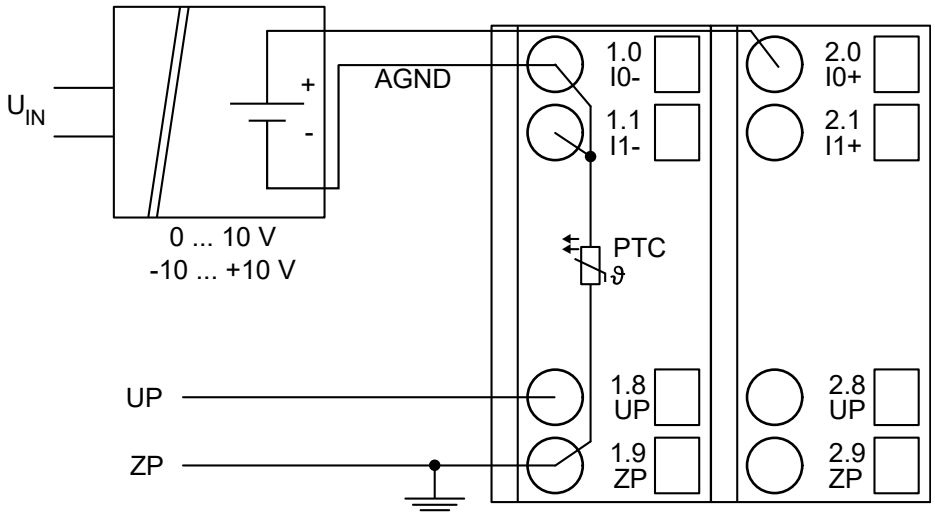


Fig. 153: Connection example

By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured for AX521 ↳ *Chapter 1.6.3.6.2.2.5.6 "Parameterization" on page 2938* and for AX522 ↳ *Chapter 1.6.3.6.2.2.6.6 "Parameterization" on page 2963*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

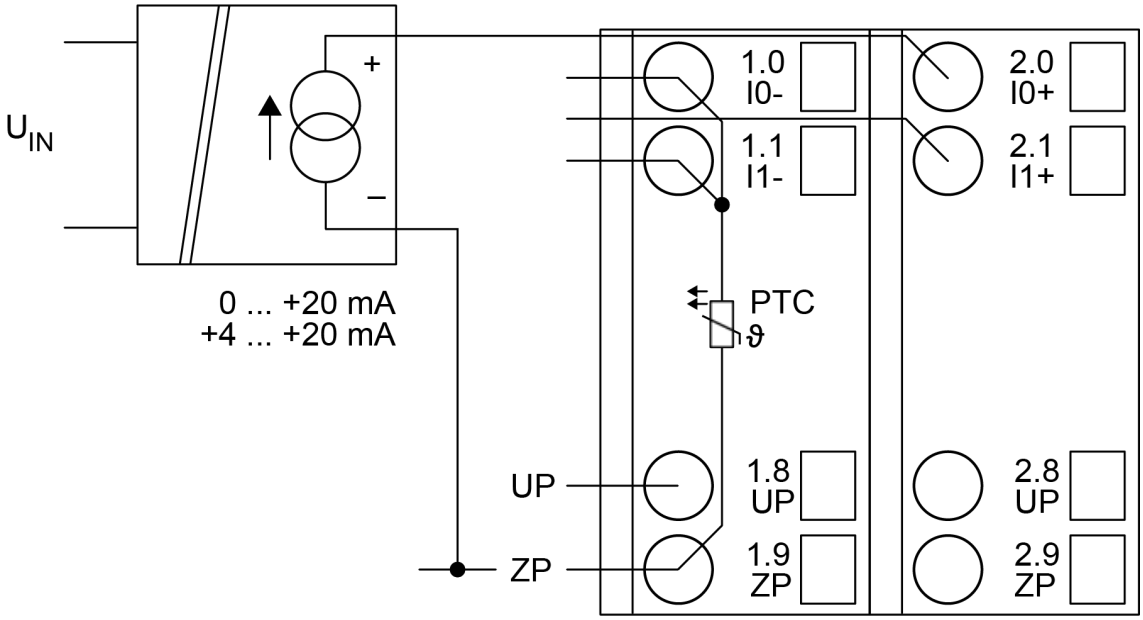


Fig. 154: Connection example

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

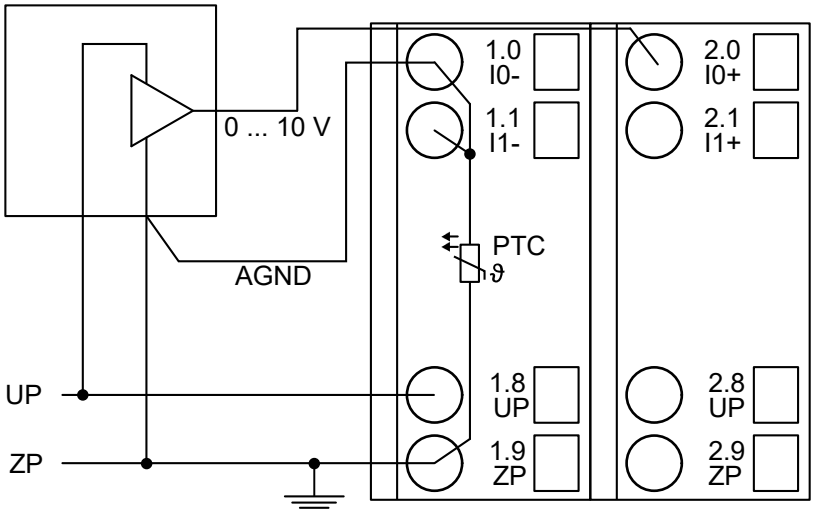


Fig. 155: Connection example

CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1V, not even in case of long lines (see figure Terminal Assignment).

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very small current flows through the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method should be applied.

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used

*) if the sensor can provide this signal range

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

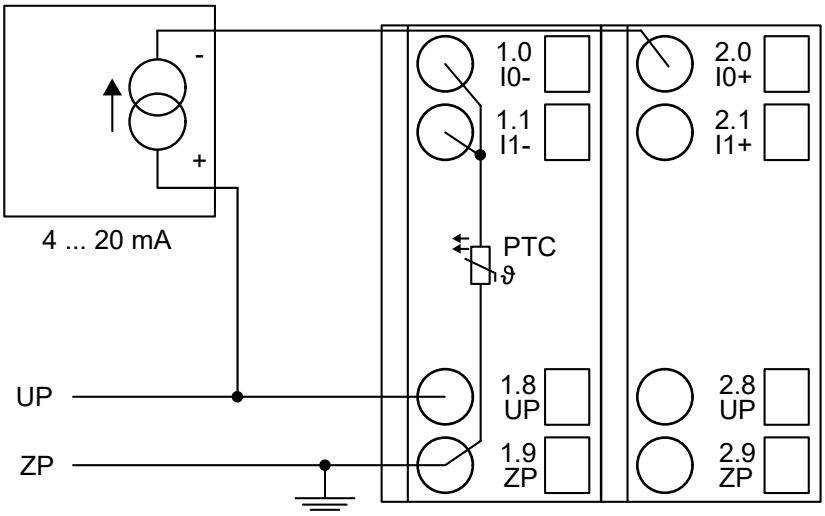


Fig. 156: Connection example

Current	4 mA...20 mA	1 channel used

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second to an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10-volt Zener diode (in parallel to I+ and I-). But, in general, sensors with fast initialization or without current peaks higher than 25 mA are preferrable.

Unused input channels can be left open-circuited because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The use of differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise, problems may occur concerning the common-mode input voltages of the involved analog inputs.

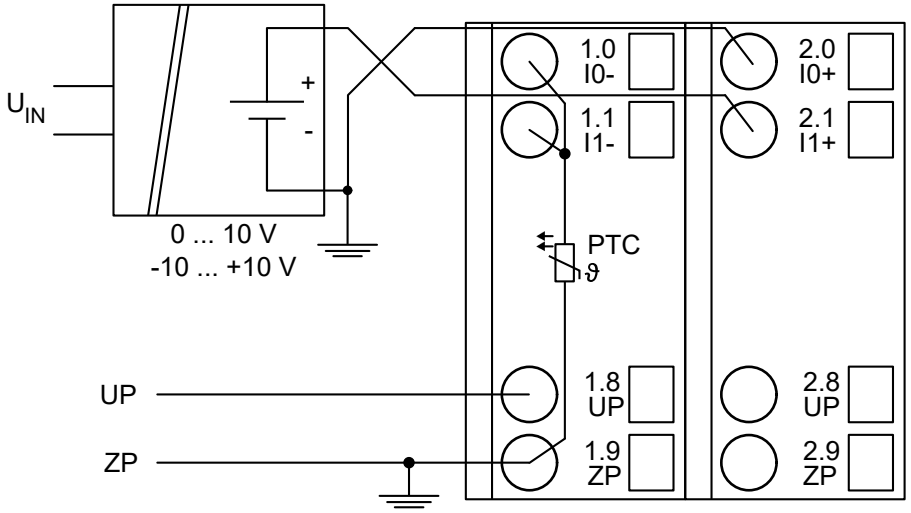


Fig. 157: Connection example

The negative pole of the sensor must be grounded next to the sensor.

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

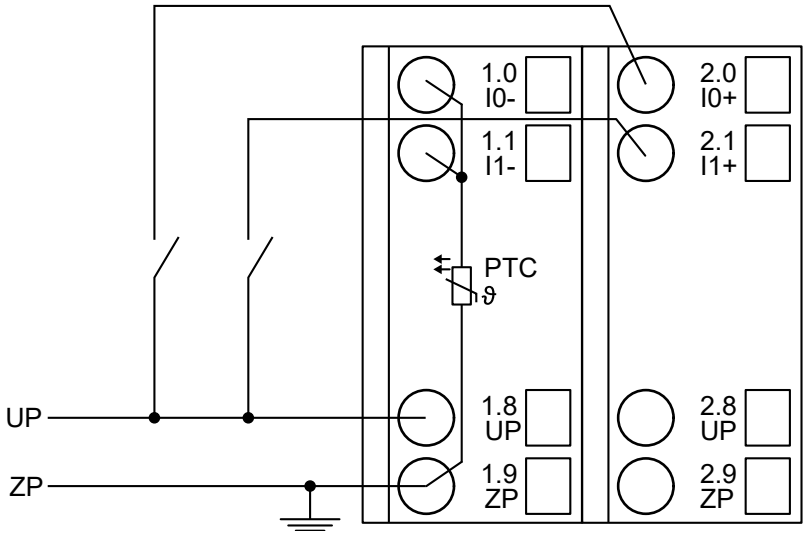


Fig. 158: Connection example

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

Connection of analog output loads (Voltage, current)

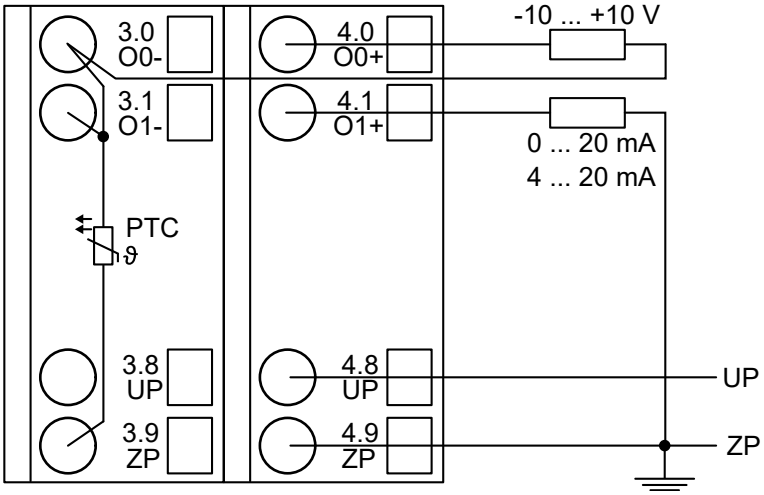


Fig. 159: Connection example

Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

Unused analog outputs can be left open-circuited.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	4
Counter output data (words)	4

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1505 [1])	Word	1505 0x05E1	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
3	Parameter length in bytes	Internal	21	Byte	21-CPU 21-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
7	Channel configuration Input channel 0	See table *Table 507 “Channel configuration 2)” on page 2940*		Byte	Default 0x00	0	19	0x0Y06
8	Channel monitoring Input channel 0	See table *Table 508 “Channel monitoring 3)” on page 2941*		Byte	Default 0x00	0	3	0x0Y07
9 to 14	Channel configuration and channel monitoring of the input channels 1 to 3	See tables *Table 507 “Channel configuration 2)” on page 2940* and *Table 508 “Channel monitoring 3)” on page 2941*		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y08 to 0x0Y0D
15	Channel configuration Output channel 0	See table *Table 507 “Channel configuration 2)” on page 2940*		Byte	Default 0x00	0	130	0x0Y0E
16	Channel monitoring Output channel 0	See table *Table 508 “Channel monitoring 3)” on page 2941*		Byte	Default 0x00	0	3	0x0Y0F
17	Substitute value Output channel 0	only valid for output channel 0	0...0xffff	Word	Default 0x0000	0	65535	0x0Y10
18 to 21	Channel configuration and channel monitoring of the output channels 1 to 2	See tables *Table 507 “Channel configuration 2)” on page 2940* and *Table 508 “Channel monitoring 3)” on page 2941*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y11 to 0x0Y14

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
22	Channel configu-ration Output channel 3	See table ↳ *Table 507 “Chan nel configuration [2])” on page 2940*		Byte	Default 0x00	0	130	0x0Y15
23	Channel moni-toring Output channel 3	See table ↳ *Table 508 “Chan nel monitoring [3])” on page 2941*		Byte	Default 0x00	0	3	0x0Y16
[1]) With CS31 and addresses less than 70 and FBP, the value is increased by 1 [2]) Not with FBP								

GSD file:

Ext_User_Prm_Data_Len =	24
Ext_User_Prm_Data_Const(0) =	0x05, 0xe2, 0x15, \ 0x01, 0x00, 0x00 \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Table 506: Input channel (4x)

No.	Name	Internal value, type	Default
1	Channel configuration see table [2])	Byte	0 0x00 see table [2])
2	Channel monitoring see table [3])	Byte	0 0x00 see table [3])

Table 507: Channel configuration [2])

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default)
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)

Internal value	Operating modes of the analog inputs, individually configurable
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 508: Channel monitoring [3])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit
3	No monitoring

Table 509: Output channel 0 (1 channel)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [4])	see table [4])	Byte	see table [4])
2	Channel monitoring	see table [5])	see table [5])	Byte	see table [5])
3	Substitute value see table [6])	0...65535	0... 0xffff	Word	0

Table 510: Output channels 1...3 (3x)

No.	Name	Internal value, type
1	Channel configuration see table [4])	Byte
2	Channel monitoring see table [6])	Byte

Table 511: Channel configuration [4])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V

Internal value	Operating modes of the analog outputs, individually configurable
129	Analog output 0 mA...20 mA (not with the channels 4...7 and 12...15)
130	Analog output 4 mA...20 mA (not with the channels 4...7 and 12...15)

Table 512: Channel monitoring [5])

Internal value	Monitoring
0	Plausibility, open circuit (broken wire) and short circuit (default)
3	No monitoring

Table 513: Substitute value [6])

Intended behaviour of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

Table 514: Possible diagnosis of I/O channels

Output range	Condition	
	Output value in the PLC underflow	Output value in the PLC overflow
0..20 mA	Error identifier = 7	Error identifier = 4
4..20 mA		
-10..+10 V		

Input range	Condition			
	Short circuit	Wire break	Input value underflow	Input value overflow
0..20 mA	no diagnosis possible	no diagnosis possible	no diagnosis possible	Error identifier = 48
4..20 mA	Error identifier = 7	Error identifier = 7	Error identifier = 7	Error identifier = 48
-10..+10 V	no diagnosis possible	Error identifier = 48	Error identifier = 7	Error identifier = 48

Table 515: Content of diagnosis messages

E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31		19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		40	Different hard-/firm-ware versions in the module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31		26	Parameter error	Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31		11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31		45	Process voltage is switched off (ON –> OFF)	Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
				AX521	AX522			
4	14	1...10	1	0...3	0...7	48	Analog value over-flow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	7	Analog value under-flow at an analog input	Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10					

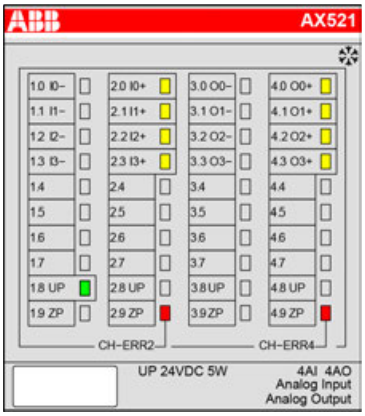
E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	3	4...7	8...15	4	Analog value over-flow at an analog output	Check output value
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	7	Analog value under-flow at an analog output	Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I3	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	Outputs O0...O3	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Measuring ranges

Input ranges of voltage, current and digital input

The represented resolution corresponds to 16 bits.

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 . . 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 . . 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	OFF	0	0000
	0.0000	-0.0004 : :		3.9994		-1 -4864 -6912	FFFF ED00 E500

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 : -1.7593	: -10.0000				: -27648	: 9400
Meas-ured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Under-flow	<-1.7593	<-11.7589	<0.0000	<1.1858		-32768	8000

Input ranges resistance temperature detector

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation	0.15 A + output loads
Inrush current from UP (at power up)		0.020 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm²	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	4	
Distribution of channels into groups	1 group of 4 channels	
Connections of the channels I0- to I3-	Terminals 1.0 to 1.3	
Connections of the channels I0+ to I3+	Terminals 2.0 to 2.3	
Input type	Bipolar (not with current or Pt100/Pt1000/Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs Current: 100 µs	
Indication of the input signals	One LED per channel	
Conversion cycle	2 ms (for 8 inputs + 8 outputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between input signal and hex code	See tables *Chapter 1.6.3.6.2.2.5.9.1 “Input ranges of voltage, current and digital input” on page 2945*	

Parameter	Value
Unused voltage inputs	Are configured as "unused"
Unused current inputs	Have a low resistance, can be left open-circuited
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels I0+ to I3+		Terminals 2.0 to 2.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		4, all channels for voltage and current
Distribution of channels into groups		1 group of 4 channels
	Channels O0-...O3-	Terminals 3.0...3.3
	Channels O0+...O3+	Terminals 4.0...4.3
Output type		Bipolar with voltage, unipolar with current
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		Max. ±10 mA
Indication of the output signals		One LED per channel
Resolution		12 bits (+ sign)

Parameter	Value	
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between output signal and hex code	See table *Chapter 1.6.3.6.2.2.5.9.3 "Output ranges voltage and current" on page 2946*	
Unused outputs	Can be left open-circuited	

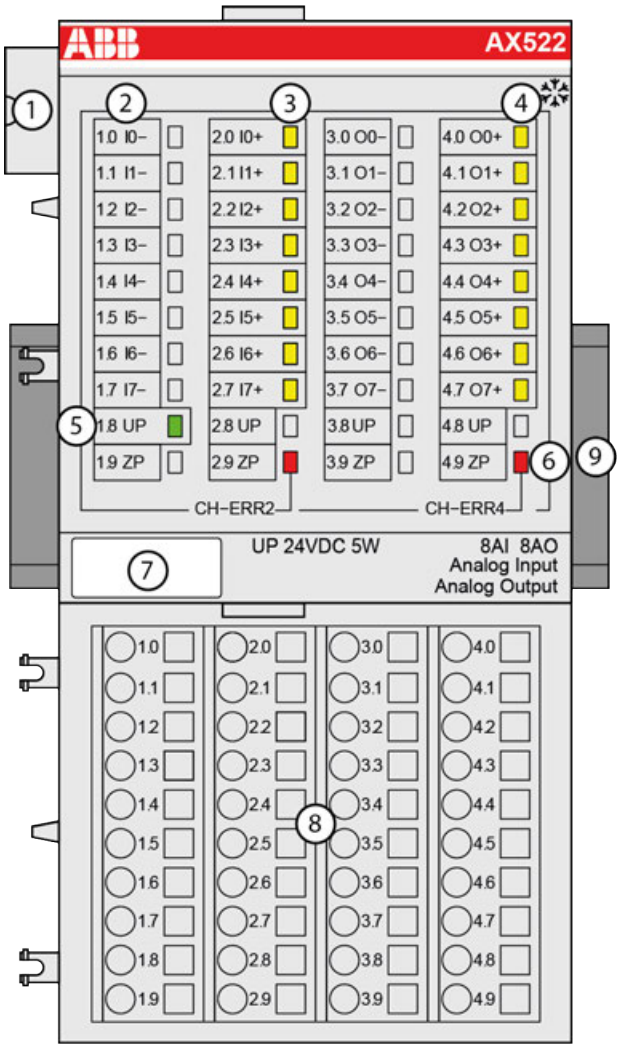
Ordering Data

Part no.	Description	Product life cycle phase *)
1SAP 250 100 R0001	AX521, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires	Active
1SAP 450 100 R0001	AX521-XC, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AX522 - Analog input/output module

- 8 configurable analog inputs (I0 to I7) in 1 group (1.0...2.7)
  Resolution 12 bits plus sign
- 8 configurable analog outputs (O0 to O7) in 1 group (3.0...4.7)
  Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the analog inputs (I0 - I7)
4 8 yellow LEDs to display the signal states at the analog outputs (O0 - O7)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

8 analog inputs (I0...I7), individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA

- 4 mA...20 mA
- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

4 analog outputs (O0...O3), individually configurable for

- Unused (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

4 analog outputs (O4...O7), individually configurable for

- Unused (default setting)
- -10 V...+10 V

Parameter		**Value**
Resolution of the analog channels		
	Voltage -10 V...+10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		19 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The modules are plugged on an I/O terminal unit ↳ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8 and 4.8 as well as 1.9, 2.9,3.9 and 4.9 are electrically interconnected within the I/O terminal units and always have the same assignment, independent of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0- to I7-	Negative poles of the 8 analog inputs
2.0 to 2.7	I0+ to I7+	Positive poles of the 8 analog inputs
3.0 to 3.7	O0- to O7-	Negative poles of the 8 analog outputs
4.0 to 4.7	O0+ to O7+	Positive poles of the 8 analog outputs

The negative poles of the analog inputs are connected to each other to form an "Analog Ground" signal for the module.

The negative poles of the analog outputs are connected to each other to form an "Analog Ground" signal for the module.

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per I/O module.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the I/O module.

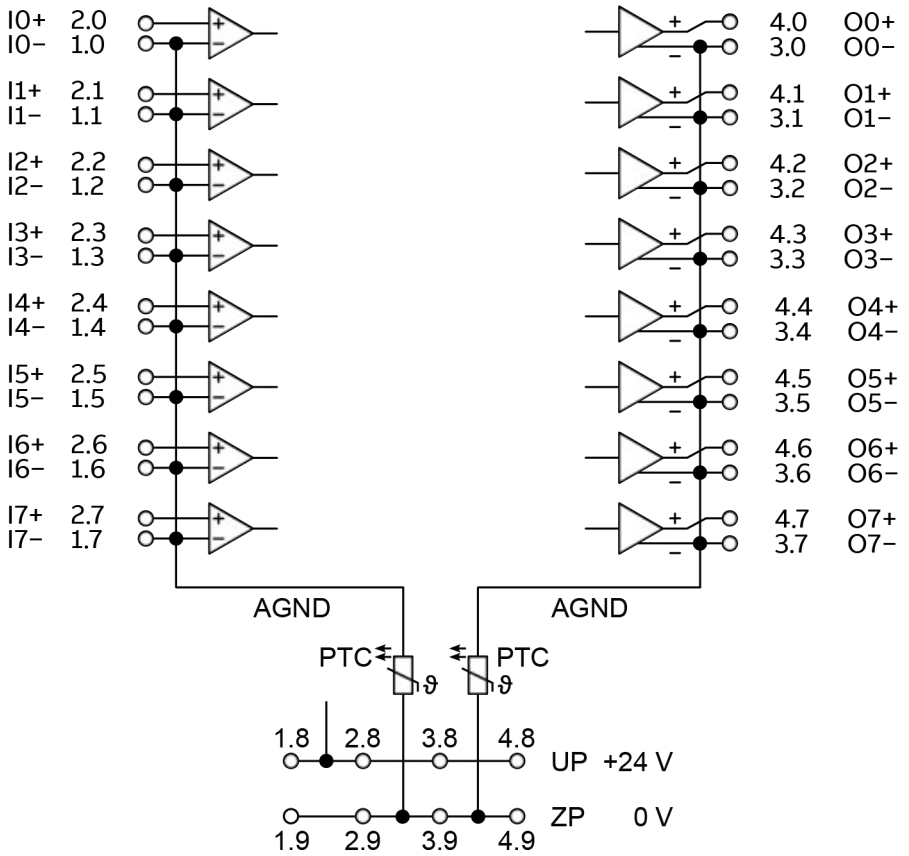


Fig. 160: 8 analog inputs and 8 analog outputs, individually configurable ↳ *Chapter 1.6.3.6.2.2.6.2 “Functionality” on page 2951*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the 8 analog channels.

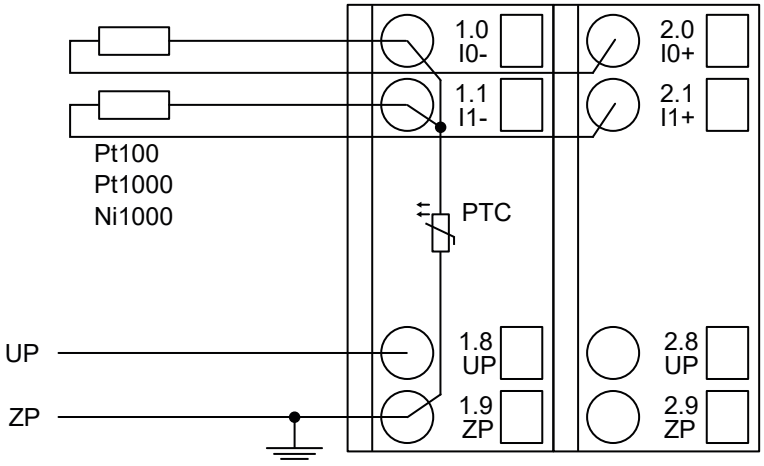


Fig. 161: Connection example

Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.

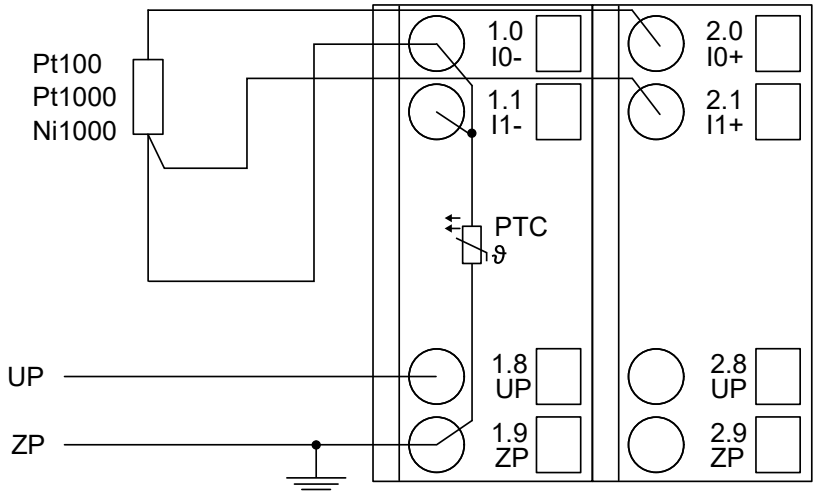


Fig. 162: Connection example

If several measuring points are adjacent to each other, only one return line is necessary. This saves wiring costs.

With the 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

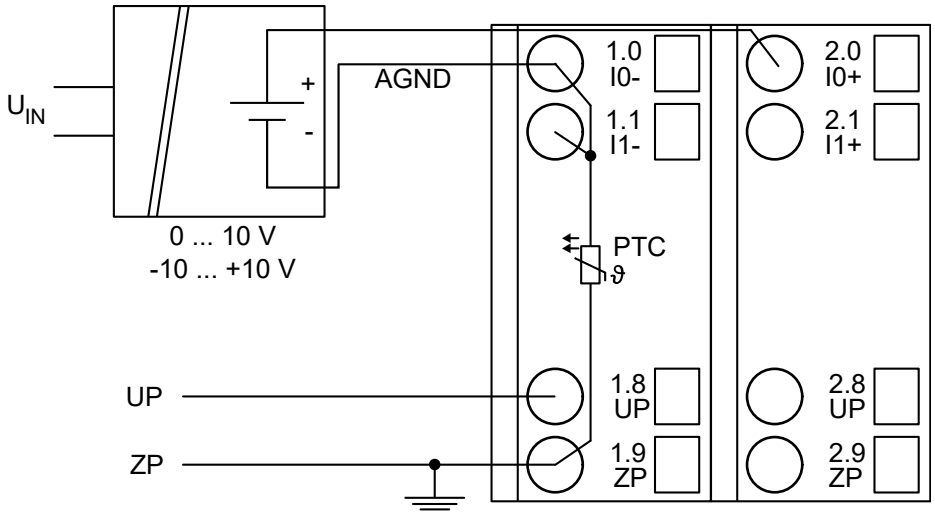


Fig. 163: Connection example

By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured for AX521 ↳ *Chapter 1.6.3.6.2.2.5.6 “Parameterization” on page 2938* and for AX522 ↳ *Chapter 1.6.3.6.2.2.6.6 “Parameterization” on page 2963*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

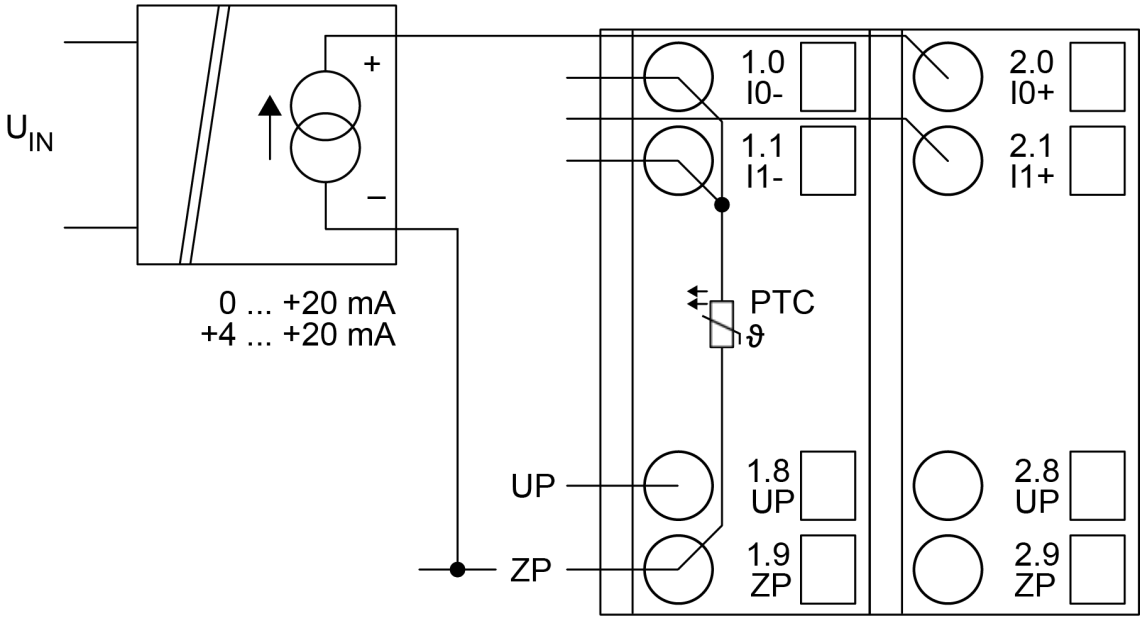


Fig. 164: Connection example

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

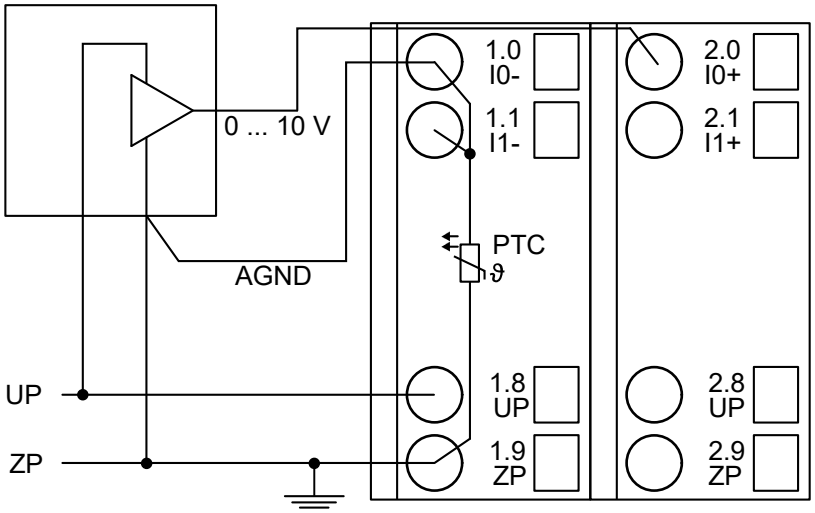


Fig. 165: Connection example

CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1V, not even in case of long lines (see figure Terminal Assignment).

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very small current flows through the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method should be applied.

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used

*) if the sensor can provide this signal range

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

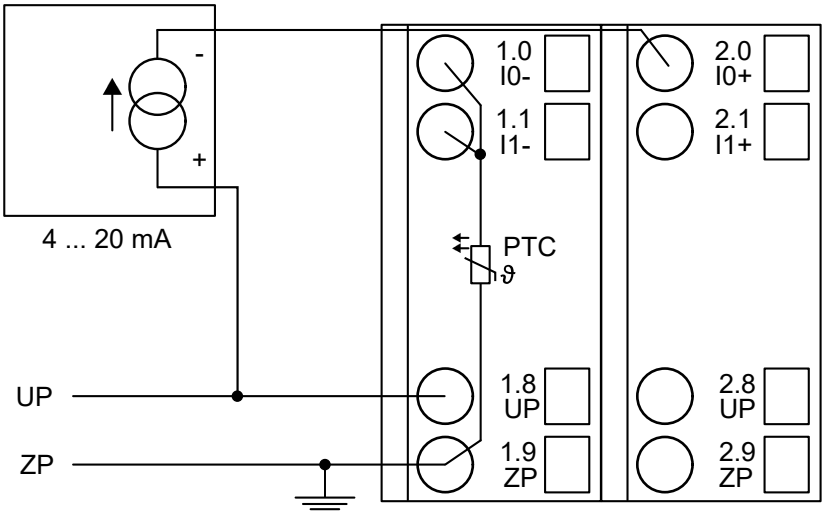


Fig. 166: Connection example

Current	4 mA...20 mA	1 channel used

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second to an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10-volt Zener diode (in parallel to I+ and I-). But, in general, sensors with fast initialization or without current peaks higher than 25 mA are preferrable.

Unused input channels can be left open-circuited because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The use of differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise, problems may occur concerning the common-mode input voltages of the involved analog inputs.

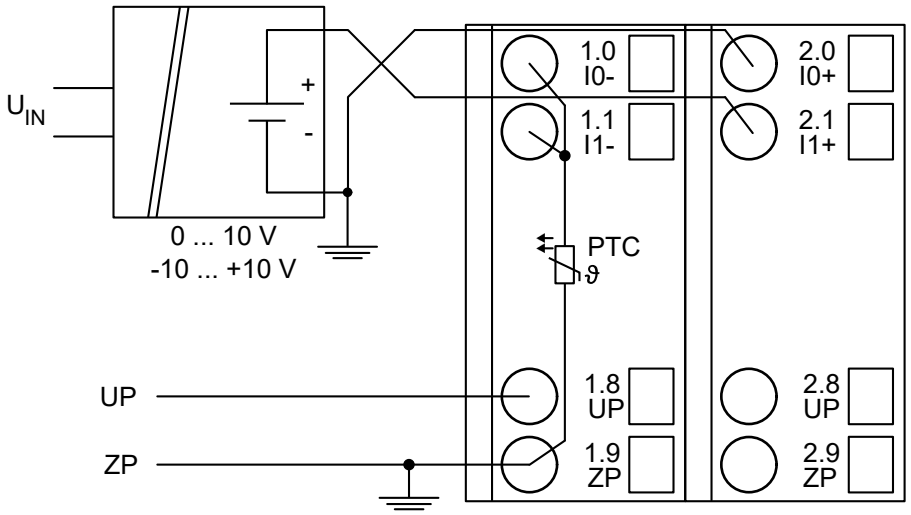


Fig. 167: Connection example

The negative pole of the sensor must be grounded next to the sensor.

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

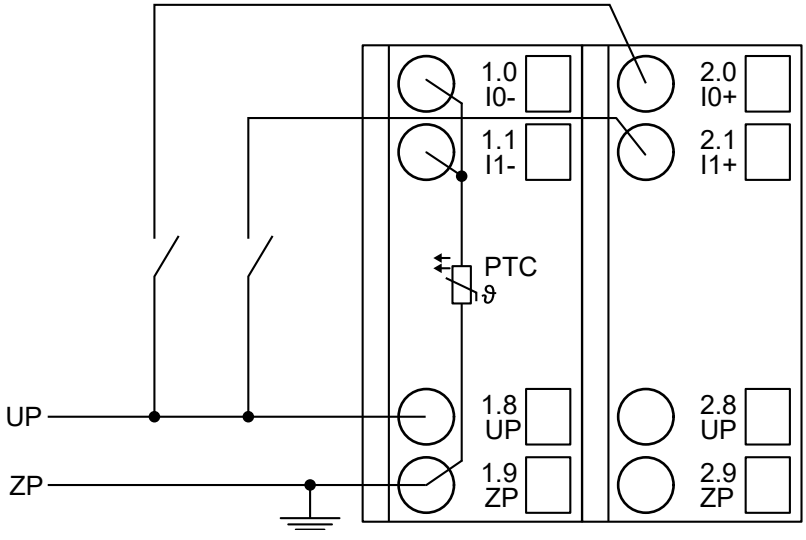


Fig. 168: Connection example

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

Connection of analog output loads (Voltage, current)

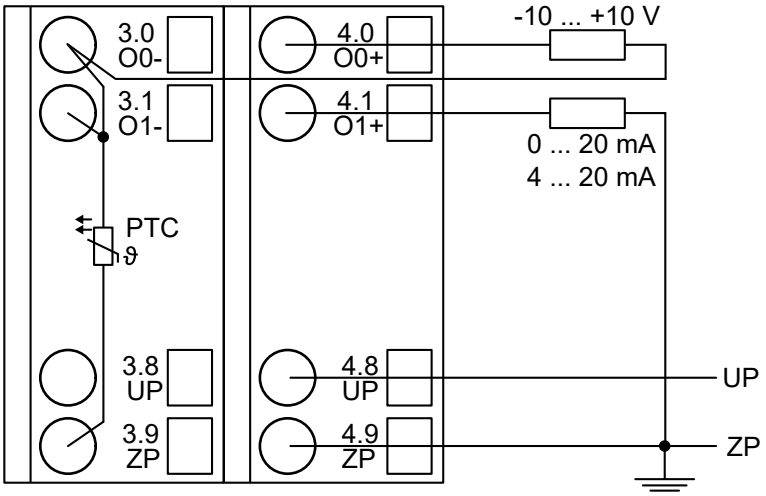


Fig. 169: Connection example

Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

Unused analog outputs can be left open-circuited.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	8
Counter output data (words)	8

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module slot address: Y = 1...7

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1500 [1)]	Word	1500 0x05dc	0	65535	0x0Y01
2	Ignore module [2)]	No Yes	0 1	Byte	No 0x00			not for FBP
3	Parameter length in bytes	Internal	37	Byte	37-CPU 37-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
7	Channel configuration Input channel 0	See *Table 517 "Channel configuration 2)" on page 2965*		Byte	Default 0x00	0	19	0x0Y06
8	Channel monitoring Input channel 0	See *Table 518 "Channel monitoring 3)" on page 2966*		Byte	Default 0x00	0	3	0x0Y07

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
9 to 22	Channel configuration and channel monitoring of the input channels 1 to 7	See ↳ *Table 517 "Channel configuration [2])" on page 2965* and ↳ *Table 518 "Channel monitoring [3])" on page 2966*		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y08 to 0x0Y15
23	Channel configuration Output channel 0	See ↳ *Table 517 "Channel configuration [2])" on page 2965*		Byte	Default 0x00	0	130	0x0Y16
24	Channel monitoring Output channel 0	See ↳ *Table 518 "Channel monitoring [3])" on page 2966*		Byte	Default 0x00	0	3	0x0Y17
25	Substitute value Output channel 0	only valid for output channel 0	0...0xffff	Word	Default 0x0000	0	65535	0x0Y18
26 to 31	Channel configuration and channel monitoring of the output channels 1 to 3	See ↳ *Table 517 "Channel configuration [2])" on page 2965* and ↳ *Table 518 "Channel monitoring [3])" on page 2966*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y19 to 0x0Y1E
32	Channel configuration Output channel 4	See ↳ *Table 517 "Channel configuration [2])" on page 2965*		Byte	Default 0x00	0	128	0x0Y1F

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
33	Channel monitoring Output channel 4	See *Table 518 "Channel monitoring 3)" on page 2966*		Byte	Default 0x00	0	3	0x0Y20
34 to 39	Channel configuration and channel monitoring of the output channels 5 to 7	See *Table 517 "Channel configuration 2)" on page 2965* and *Table 518 "Channel monitoring 3)" on page 2966*		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y21 to 0x0Y26

1) With CS31 and addresses less than 70 and FBP, the value is increased by 1

2) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	24
Ext_User_Prm_Data_Const(0) =	0x05, 0xe2, 0x15, \ 0x01, 0x00, 0x00 \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Table 516: Input channel (4x)

No.	Name	Internal value, type	Default
1	Channel configuration see table 2)	Byte	0 0x00 see table 2)
2	Channel monitoring see table 3)	Byte	0 0x00 see table 3)

Table 517: Channel configuration 2)

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default)
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA

Internal value	Operating modes of the analog inputs, individually configurable
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 518: Channel monitoring [3])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit
3	No monitoring

Table 519: Output channel 0 (1 channel)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [4])	see table [4])	Byte	see table [4])
2	Channel monitoring	see table [5])	see table [5])	Byte	see table [5])
3	Substitute value see table [6])	0...65535	0... 0xffff	Word	0

Table 520: Output channels 1...3 (3x)

No.	Name	Internal value, type
1	Channel configuration see table [4])	Byte
2	Channel monitoring see table [6])	Byte

Table 521: Channel configuration [4])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V
129	Analog output 0 mA...20 mA (not with the channels 4...7 and 12...15)
130	Analog output 4 mA...20 mA (not with the channels 4...7 and 12...15)

Table 522: Channel monitoring [5])

Internal value	Monitoring
0	Plausibility, open circuit (broken wire) and short circuit (default)
3	No monitoring

Table 523: Substitute value [6])

Intended behaviour of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

Table 524: Possible diagnosis of I/O channels

Output range	Condition	
	Output value in the PLC underflow	Output value in the PLC overflow
0..20 mA	Error identifier = 7	Error identifier = 4
4..20 mA		
-10..+10 V		

Input range	Condition			
	Short circuit	Wire break	Input value under-flow	Input value over-flow
0..20 mA	no diagnosis possible	no diagnosis possible	no diagnosis possible	Error identifier = 48
4..20 mA	Error identifier = 7	Error identifier = 7	Error identifier = 7	Error identifier = 48
-10..+10 V	no diagnosis possible	Error identifier = 48	Error identifier = 7	Error identifier = 48

Table 525: Content of diagnosis messages

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diag-nosis block		
Class	Interface	Device	Module	Channel	Error Identifier	Error message		Remedy
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firm-ware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON –> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
				AX521	AX522			

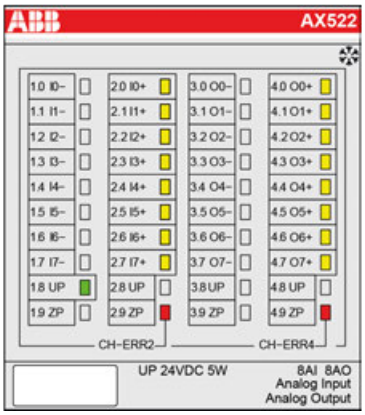
E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	1	0...3	0...7	48	Analog value over-flow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	7	Analog value under-flow at an analog input	Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	4	Analog value over-flow at an analog output	Check output value
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	7	Analog value under-flow at an analog output	Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	Outputs O0...O7	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Measuring ranges

Input ranges of voltage, current and digital input

The represented resolution corresponds to 16 bits.

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 . . 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 . . 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
		0.0000	0	4	OFF	0	0000
	0.0000	-0.0004 : :		3.9994		-1 -4864 -6912	FFFF ED00 E500

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 : -1.7593	: -10.0000				: -27648	: 9400
Meas-ured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Under-flow	<-1.7593	<-11.7589	<0.0000	<1.1858		-32768	8000

Input ranges resistance temperature detector

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation	0.15 A + output loads
Inrush current from UP (at power up)		0.020 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm²	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	8	
Distribution of channels into groups	1 group of 8 channels	
Connections of the channels I0- to I7-	Terminals 1.0 to 1.7	
Connections of the channels I0+ to I7+	Terminals 2.0 to 2.3	
Input type	Bipolar (not with current or Pt100/Pt1000/Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs current: 100 µs	
Indication of the input signals	One LED per channel	
Conversion cycle	2 ms (for 8 inputs + 8 outputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Unused voltage inputs	Are configured as "unused"	

Parameter	Value
Unused current inputs	Have a low resistance, can be left open-circuited
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital Inputs

Parameter		Value
Number of channels per module		Max. 8
Distribution of channels into groups		1 group of 8 channels
Connections of the channels I0+ to I7+		Terminals 2.0 to 2.7
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		8, all channels for voltage, the first 4 channels also for current
Distribution of channels into groups		1 group of 8 channels
	Channels O0-...O7-	Terminals 3.0...3.7
	Channels O0+...O7+	Terminals 4.0...4.7
Output type		Bipolar with voltage, unipolar with current
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		Max. ±10 mA
Indication of the output signals		One LED per channel
Resolution		12 bits (+ sign)

Parameter	Value	
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between output signal and hex code	See table, *Chapter 1.6.3.6.2.2.6.9.3 "Output ranges voltage and current" on page 2971*	
Unused outputs	Can be left open-circuited	

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 000 R0001	AX522, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires	Active
1SAP 450 000 R0001	AX522-XC, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

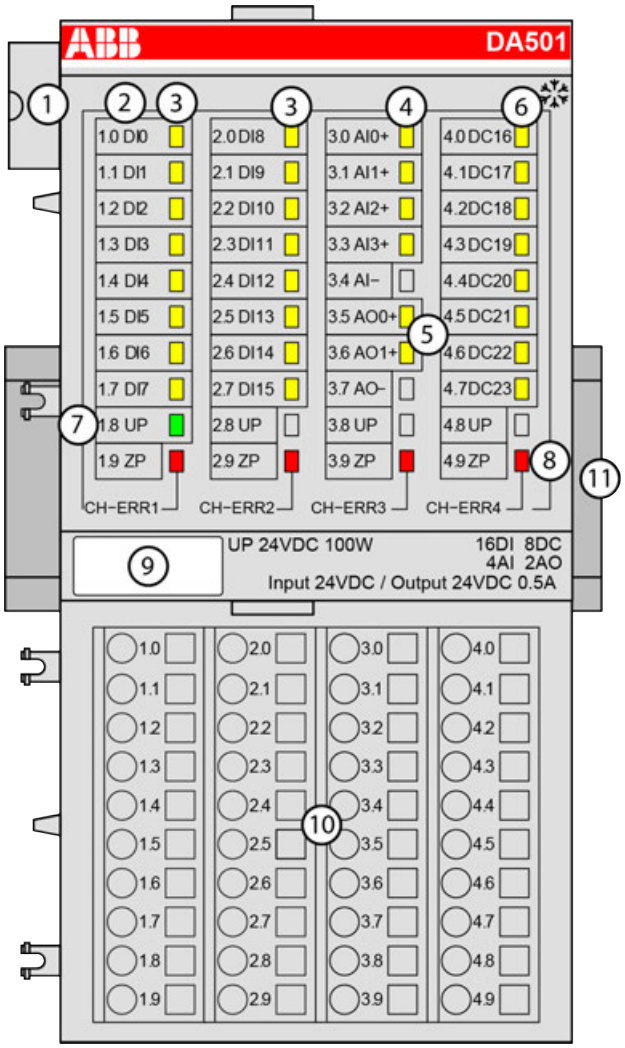
1.6.3.6.3 Digital/Analog I/O modules

S500

DA501 - Digital/Analog input/output module

- 16 digital inputs 24 V DC
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- 4 analog inputs, voltage, current and RTD.
  Resolution 12 bits plus sign
- 2 analog outputs, voltage and current
  Resolution 12 bits plus sign
- Fast counter

- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states of the digital inputs DI0 to DI15
4 4 yellow LEDs to display the signal states of the analog inputs AI0 to AI3
5 2 yellow LEDs to display the signal states of the analog outputs AO0 to AO1
6 8 yellow LEDs to display the signal state of the configurable digital inputs/outputs DC16 to DC23
7 1 green LED to display the state of the process supply voltage UP
8 4 red LEDs to display errors
9 Label
10 Terminal unit
11 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

- 16 digital inputs 24 V DC
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.

- 4 analog inputs, voltage, current and RTD.
  Resolution 12 bits plus sign
- 2 analog outputs, voltage and current
  Resolution 12 bits plus sign
- Fast counter

Parameter	Value
Fast Counter	Integrated, many configurable operating modes
Power supply	From the process supply voltage UP
LED displays	For system displays, signal states, errors and power supply
Internal supply voltage	Via the I/O bus interface (I/O bus)
External supply voltage	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU515 or TU516 ↬ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection is carried out by using the 40 terminals of the terminal unit TU515/TU516 ↬ *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*.

The assignment of the terminals:

Terminal	Signal	Description
1.0	DI0	Signal of the digital input DI0
1.1	DI1	Signal of the digital input DI1
1.2	DI2	Signal of the digital input DI2
1.3	DI3	Signal of the digital input DI3
1.4	DI4	Signal of the digital input DI4
1.5	DI5	Signal of the digital input DI5
1.6	DI6	Signal of the digital input DI6
1.7	DI7	Signal of the digital input DI7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI8	Signal of the digital input DI8
2.1	DI9	Signal of the digital input DI9
2.2	DI10	Signal of the digital input DI10

Terminal	Signal	Description
2.3	DI11	Signal of the digital input DI11
2.4	DI12	Signal of the digital input DI12
2.5	DI13	Signal of the digital input DI13
2.6	DI14	Signal of the digital input DI14
2.7	DI15	Signal of the digital input DI15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	AI0+	Positive pole of analog input signal 0
3.1	AI1+	Positive pole of analog input signal 1
3.2	AI2+	Positive pole of analog input signal 2
3.3	AI3+	Positive pole of analog input signal 3
3.4	AI-	Negative pole of analog input signals 0 to 3
3.5	AO0+	Positive pole of analog output signal 0
3.6	AO1+	Positive pole of analog output signal 1
3.7	AO-	Negative pole of analog output signals 0 and 1
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	C16	Signal of the configurable digital input/ output C16
4.1	C17	Signal of the configurable digital input/ output C17
4.2	C18	Signal of the configurable digital input/ output C18
4.3	C19	Signal of the configurable digital input/ output C19
4.4	C20	Signal of the configurable digital input/ output C20
4.5	C21	Signal of the configurable digital input/ output C21
4.6	C22	Signal of the configurable digital input/ output C22
4.7	C23	Signal of the configurable digital input/ output C23
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DA501.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

CAUTION!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalization of a low resistance to avoid high potential differences between different parts of the plant.

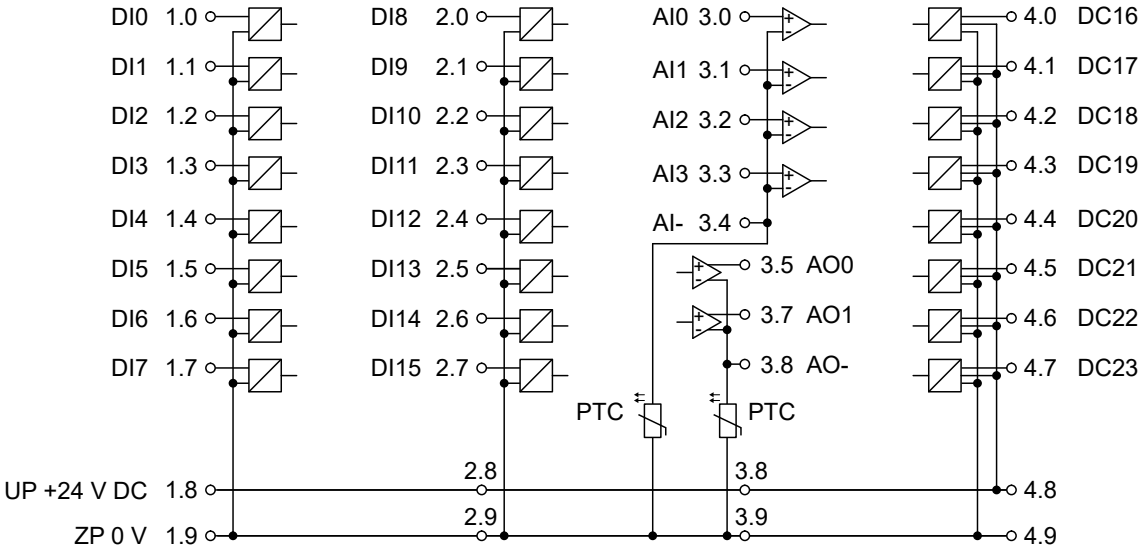


Fig. 170: Terminal assignment of the module

The module provides several diagnosis functions *Chapter 1.6.3.6.3.1.1.7 “Diagnosis” on page 2996*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI15 in the same way.

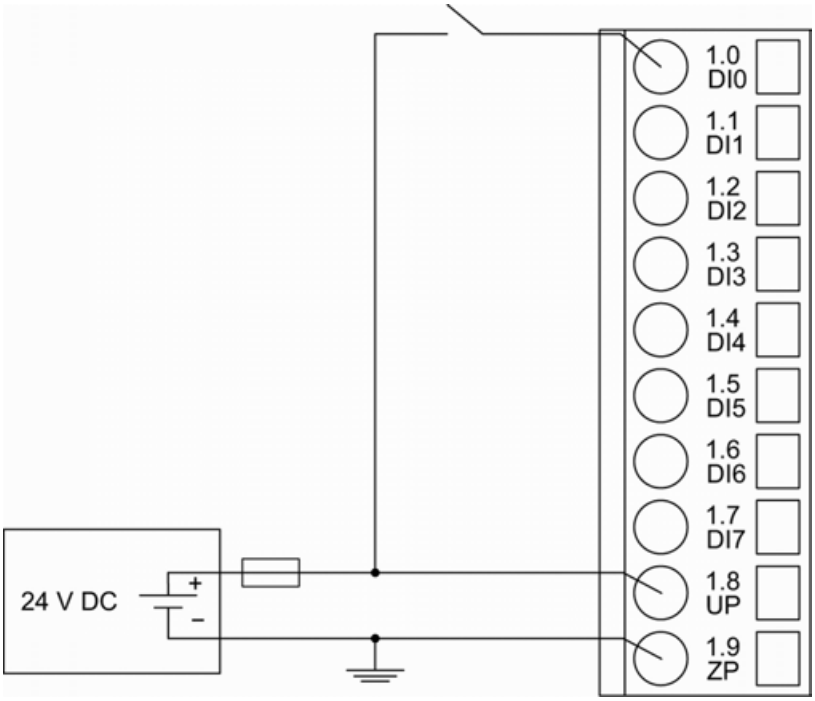


Fig. 171: Connection of the module

The meaning of the LEDs is described in the Displays *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999* chapter.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC16 and DC17. DC16 is connected as an input and DC17 is connected as an output. Proceed with the configurable digital inputs/outputs DC18 to DC23 in the same way.

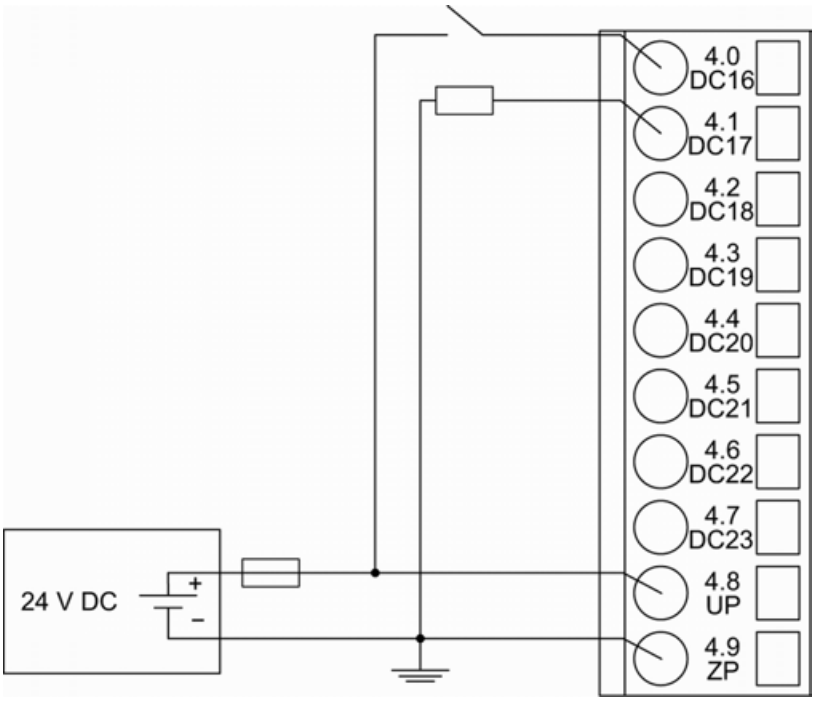


Fig. 172: Connection of configurable digital inputs/outputs to the module

CAUTION!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of DA501.

If the inputs are used as fast counter inputs, connect a 470 Ω / 1 W resistor in series to inputs DC16/DC17.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA501 provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

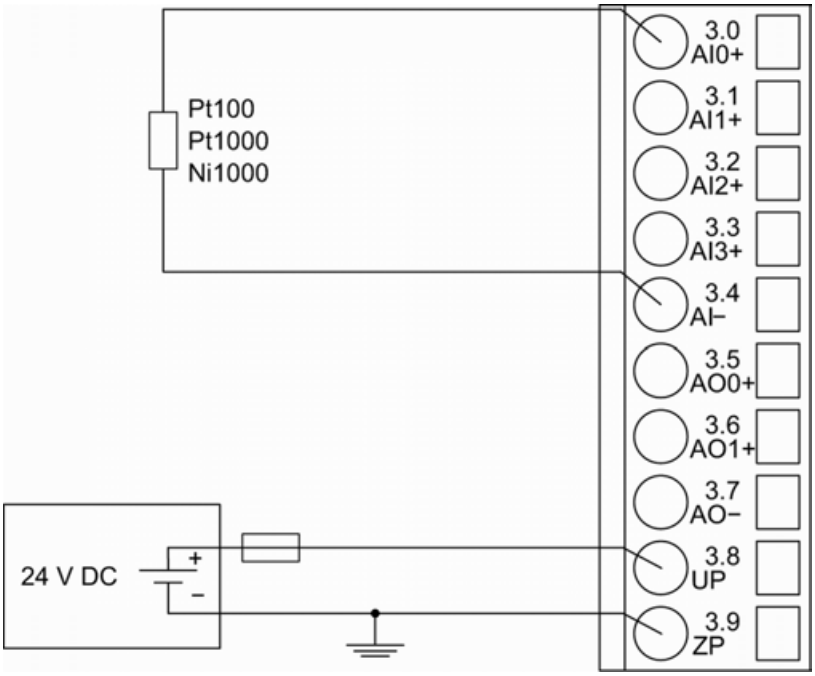


Fig. 173: Connection of resistance thermometers in 2-wire configuration to the analog inputs

The following measuring ranges can be configured *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

The module DA501 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA501 provides a constant current source which is multiplexed over the max. 4 analog input channels.

0

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

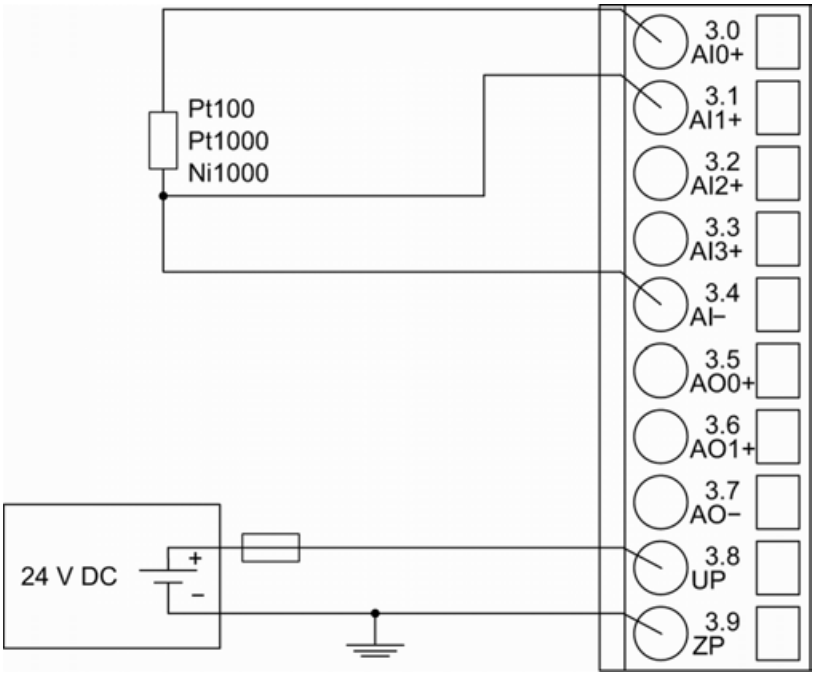


Fig. 174: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.1.7 “Diagnosis” on page 2996*.

0

The module DA501 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

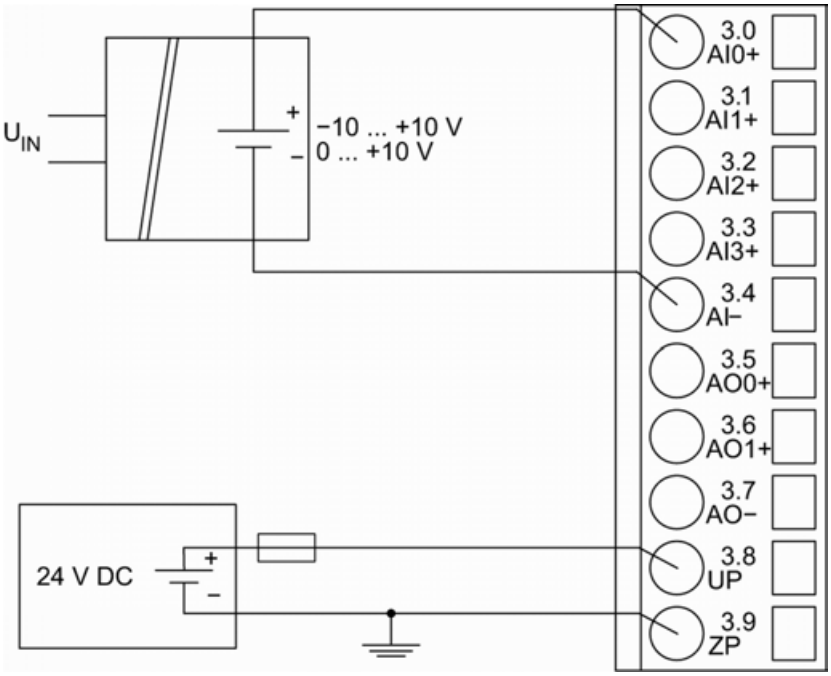


Fig. 175: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

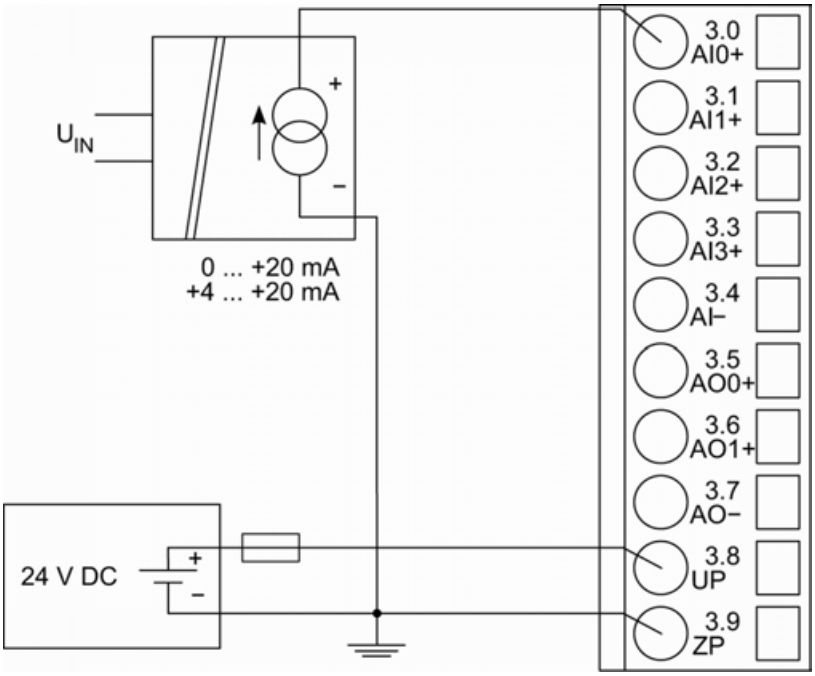


Fig. 176: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

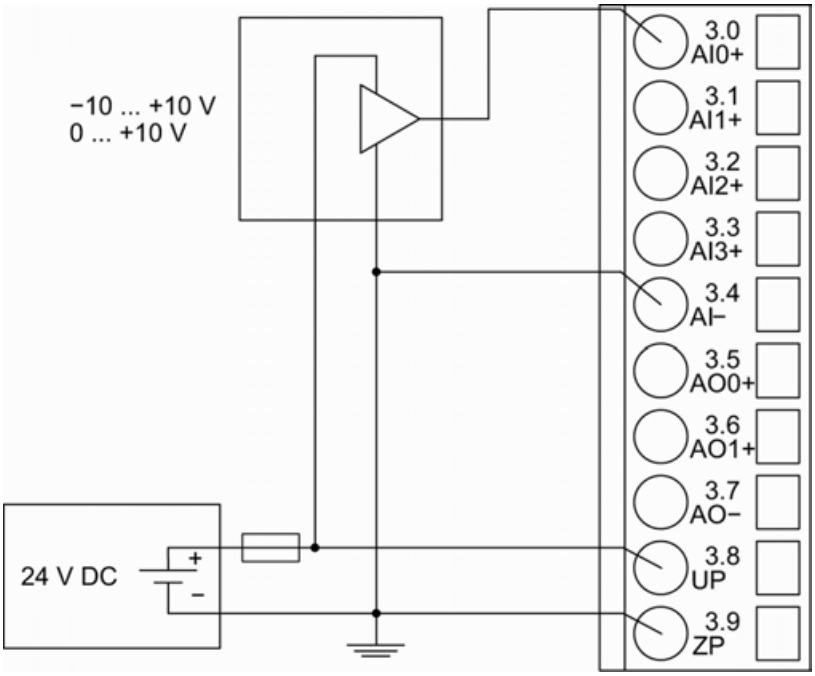


Fig. 177: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↳ *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

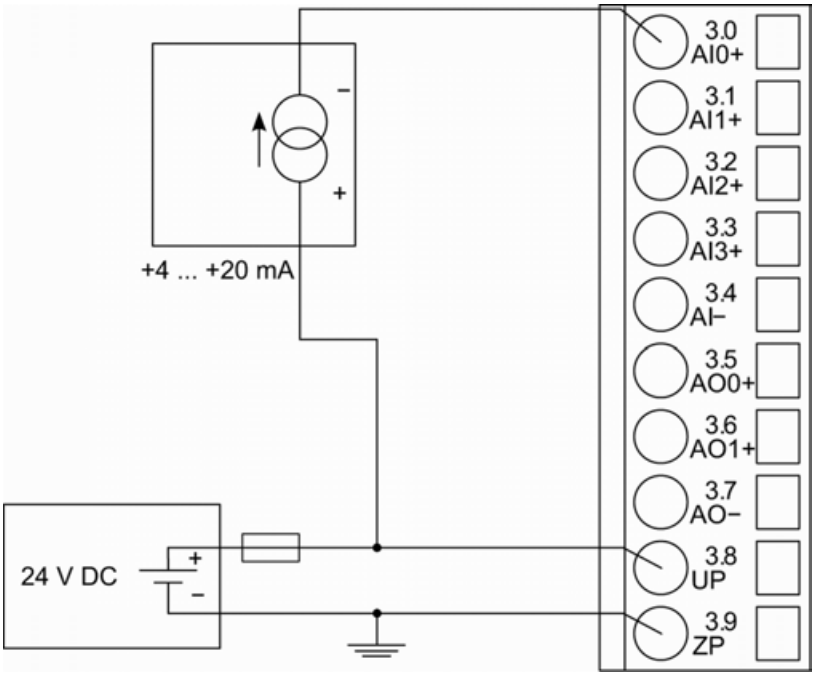


Fig. 178: Connection of passive-type analog sensors (current) to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 "Parameterization" on page 2992*:

Current	4 mA...20 mA	1 channel used

For a description of function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↳ *Chapter 1.6.3.6.3.1.1.8 "State LEDs" on page 2999*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Only use sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt Zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

Using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

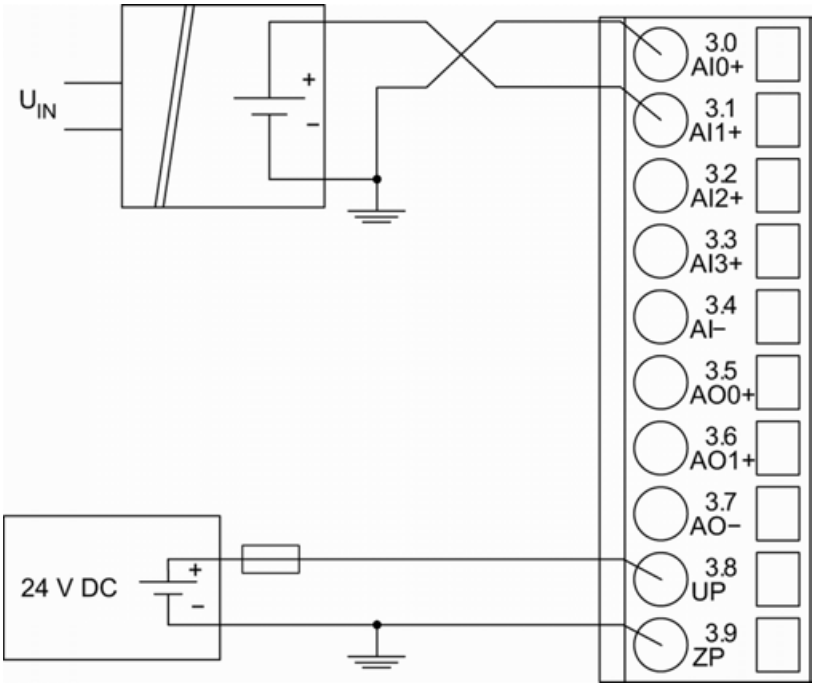


Fig. 179: Connection of active-type analog sensors (voltage) to differential analog inputs

The following measuring ranges can be configured *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

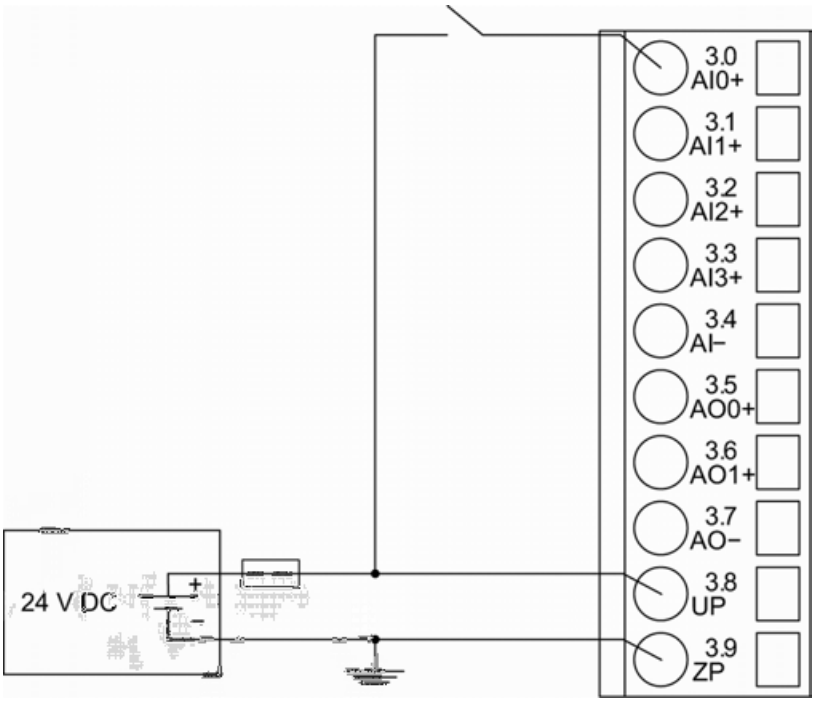


Fig. 180: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

Digital input	24 V	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↳ *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

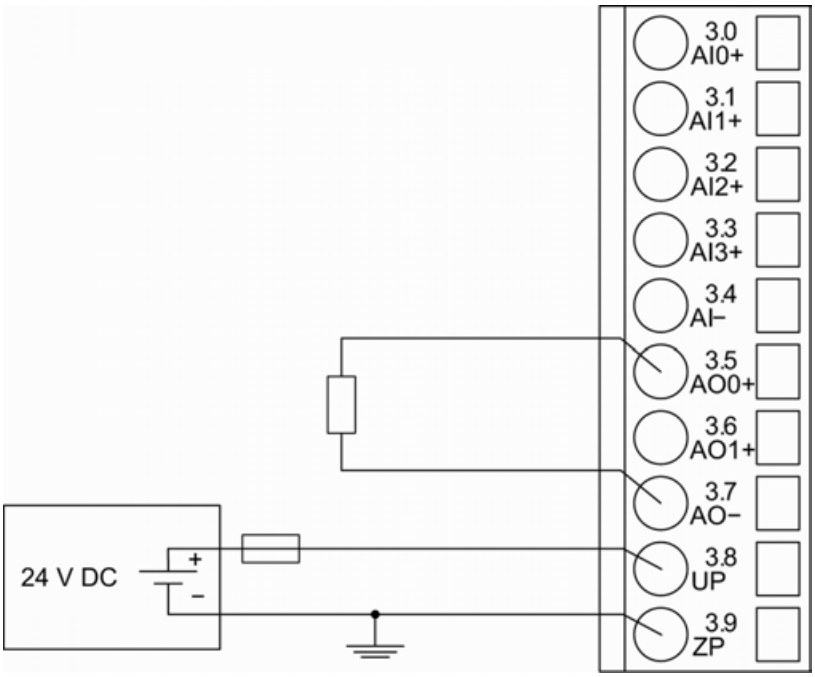


Fig. 181: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992* :

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↳ *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

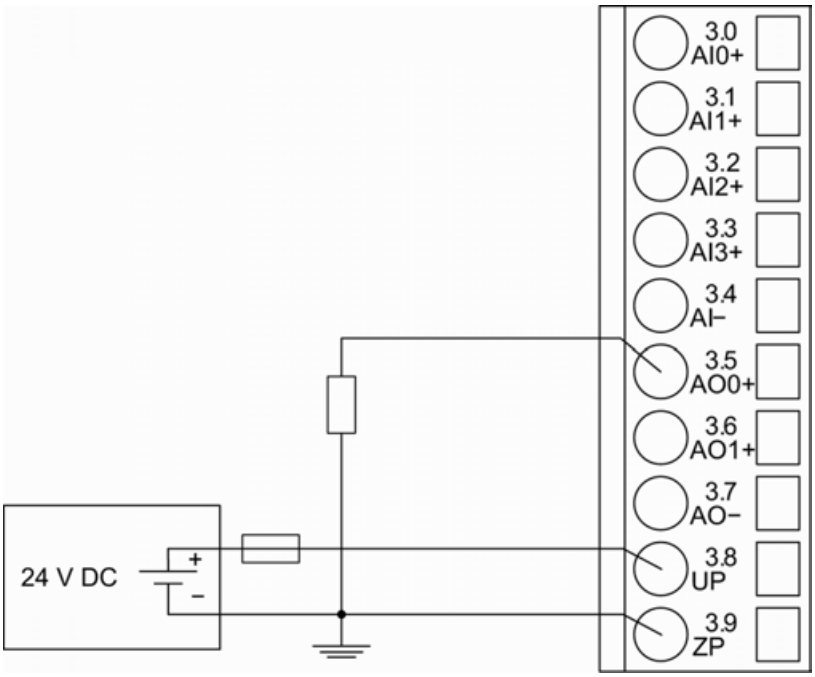


Fig. 182: Connection of analog output loads (current)

The following measuring ranges can be configured ↬ *Chapter 1.6.3.6.3.1.1.6 “Parameterization” on page 2992*:

0

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↬ *Chapter 1.6.3.6.3.1.1.8 “State LEDs” on page 2999*.

Unused analog outputs can be left open-circuited.

Internal data exchange

	Without the fast counter	**With the fast counter (only with AC500)**
Digital inputs (bytes)	3	5
Digital outputs (bytes)	1	3
Analog inputs (words)	4	4
Digital outputs (words)	2	2
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Module ID [1])	Internal	1810	WORD	1810	0x0Y01
Ignore module see table [2])	Internal	Yes No	BYTE	No	not for FBP
Parameter length	Internal	8	BYTE	8	0xY02
Check supply	off	0	BYTE	1	0xY03
	on	1			
Fast counter [3])	0 : 10 [4])	0 : 10	BYTE	0	not for FBP
Behavior outputs at comm. error [5])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00	0x0Y07

[2])	Setting	Description
	On	Error LED lights up at errors of all error classes, Failsafe mode off
	Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
	Off by E3	Error LED lights up at errors of error classes E1 and E2, Failsafe mode off
	On +Failsafe	Error LED lights up at errors of all error classes, Failsafe mode on *)

[2])	Setting	Description
	Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
	Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

Remarks:

[1]) With a faulty ID, the Modules reports a "parameter error" and does not perform cyclic process data transmission

[2]) Not for FBP

[3]) With FBP or CS31 without the parameter "Fast Counter"

The fast counter of the module does not work if the module is connected to an FBP interface module or CS31 bus module.

[4]) For counter operating modes, please refer to the description of the fast counter ↬ *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*

[5]) The parameter Behavior outputs at comm. error is only analyzed if the Failsafe-mode is ON.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0x0Y05
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01	0x0Y06
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000	0x0Y08

*) The parameters Behavior DO at comm. error is only analyzed if the Failsafe mode is ON.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Analog data format	Standard Reserved	0 255	BYTE	0	0x0Y04

*) The parameter Behavior AO at comm. error is only analyzed if the Failsafe mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input 0, Channel configuration	see *Table 526 “ Channel configuration” on page 2994*	see *Table 526 “ Channel configuration” on page 2994*	BYTE	0	0x0Y09
Input 0, Check channel	see *Table 527 “ Channel monitoring” on page 2995*	see *Table 527 “ Channel monitoring” on page 2995*	BYTE	0	0x0Y0A
:	:	:	:	:	
:	:	:	:	:	
Input 3, Channel configuration	see *Table 526 “ Channel configuration” on page 2994*	see *Table 526 “ Channel configuration” on page 2994*	BYTE	0	0x0Y0F
Input 3, Check channel	see *Table 527 “ Channel monitoring” on page 2995*	see *Table 527 “ Channel monitoring” on page 2995*	BYTE	0	0x0Y10

Table 526: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0 V...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

Internal value	Operating modes of the analog inputs, individually configurable
19	3-wire Ni1000 -50 °C...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 527: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
0 Output 0, Channel configuration	see *Table 528 “Channel configuration” on page 2996*	see *Table 528 “Channel configuration” on page 2996*	BYTE	0	0x0Y11
Output 0, Check channel	see *Table 529 “Channel monitoring” on page 2996*	see *Table 529 “Channel monitoring” on page 2996*	BYTE	0	0x0Y12
Output 0, Substitute value	see *Table 530 “Substitute value” on page 2996*	see *Table 530 “Substitute value” on page 2996*	WORD	0	0x0Y13
Output 1, Channel configuration	see *Table 528 “Channel configuration” on page 2996*	see *Table 528 “Channel configuration” on page 2996*	BYTE	0	0x0Y14
Output 1, Check channel	see *Table 529 “Channel monitoring” on page 2996*	see *Table 529 “Channel monitoring” on page 2996*	BYTE	0	0x0Y15
Output 1, Substitute value	see *Table 530 “Substitute value” on page 2996*	see *Table 530 “Substitute value” on page 2996*	WORD	0	0x0Y16

Table 528: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Table 529: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 530: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behavior of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

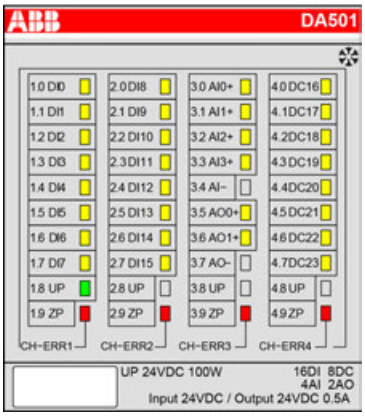
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
0	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
3	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error DA501							
4	14	1...10	2	22...29 [5])	47	Short circuit at a digital output	Check connection
	11 / 12	ADR	1...10				
Channel error DA501							
4	14	1...10	1	16...19 [6])	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	16...19 [6])	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				
4	14	1...10	1	16...19 [6])	47	Short circuit at an analog input	Check ter-minal
	11 / 12	ADR	1...10				
4	14	1...10	3	20...21 [7])	4	Analog value overflow at an analog output	Check output value
	11 / 12	ADR	1...10				

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	3	20...21 [7])	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO, 4 = DC); COM1/ COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.
[5])	Ch = 22...29 indicates the digital inputs/outputs DC16...DC23
[6])	Ch = 16...19 indicates the analog inputs AI0...AI3
[7])	Ch = 20...21 indicates the analog outputs AO0...AO1

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	DI0 to DI15	Digital input	Yellow	Input is OFF	Input is ON [1])	--
	DC16 to DC23	Digital input/ output	Yellow	Input/output is OFF	Input/output is ON [1])	--
	AI0 to AI3	Analog input	Yellow	Input is OFF	Input is ON [2])	--
	AO0 to AO1	Analog output	Yellow	Output is OFF	Output is ON [2])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Severe error within the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [3])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) Brightness depends on the value of the analog signal					
	[3]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input
Overflow	> 11.7589	> 11.7589	> 23.5178	> 22.8142	
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006	
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	on
	0.0000	0.0000	0	4	off
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0	

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input
Measured value too low		-10.0004 : -11.7589			
Underflow	< 0.0000	< -11.7589	< 0.0000	< 0.0000	

Range	Digital value	
	Decimal	Hex.
Overflow	32767	7FFF
Measured value too high	32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	27648 : 1	6C00 : 0001
	0	0000
	-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low	-27649 : -32512	93FF : 8100
Underflow	-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C
Measured value too high		450.0 °C : 400.1 °C	
			160.0 °C : 150.1 °C
	80.0 °C : 70.1 °C		

Range	**Pt100 / Pt1000** **-50...70 °C**	**Pt100 / Pt1000** **-50...400 °C**	**Ni1000** **-50...150 °C**
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C
	0.0 °C	0.0 °C	0.0 °C
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C

Range	**Digital value**	
	Decimal	**Hex.**
Overflow	32767	7FFF
Measured value too high	4500 : 4001	1194 : 0FA1
	1600 : 1501	0640 : 05DD
	800 : 701	0320 : 02BD
Normal range	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0	0000
	-1 : -500	FFFF : FE0C
Measured value too low	-501 : -600	FE0B : FDA8
Underflow	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA
Overflow	>11.7589 V	>23.5178 mA	>22.8142 mA
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA
	0.0000 V	0.0000 mA	4.0000 mA
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA
Underflow	0 V	0 mA	0 mA

Range	Digital value	
	Decimal	Hex.
Overflow	> 32511	> 7EFF
Value too high	32511 : 27649	7EFF : 6C01
Normal range	27648 : 1	6C00 : 0001
	0	0000
	-1 -6912 -27648	FFFF E500 9400
Value too low	-27649 : -32512	93FF : 8100
Underflow	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

Technical data of the module

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		**Value**
Process supply voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for UP (+24 V DC) and 1.9, 2.9, 3.9 and 4.9 for ZP (0 V DC)
	Protection against reverse voltage	yes
	Rated protection fuse at UP	10 A fast
	Rated value	24 V DC
	Max. ripple	5 %
Current consumption		
	From UP	0.07 A + max. 0.5 A per output
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	ca. 2 mA
	Inrush current from UP (at power-up)	0.04 A^2s
Galvanic isolation		Yes, per module
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal mounting or vertical with derating (output load reduced to 50 % at 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		16
Distribution of the channels into groups		2 groups of 8 channels
Terminals of the channels DI0 to DI7		Terminals 1.0 to 1.7
Terminals of the channels DI8 to DI15		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator		LED is part of the input circuitry
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable digital I/O channels can be defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC16...DC23	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels DC16...DC23	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)

Parameter	Value
Monitoring point of input/output indicator	LED is part of the input circuitry
Galvanic isolation	Yes, per module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC16 to DC23		Terminals 4.0 to 4.7
Reference potential for all inputs		Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	shielded	1000 m
	unshielded	600 m

* Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DC16 to DC23	Terminals 4.0 to 4.7

Parameter	Value
Reference potential for all outputs	Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the supply voltage, signal name UP)
Output voltage for signal 1	UP (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	
rated value per channel	500 mA at UP = 24 V
max. value (all channels together)	4 A
Leakage current with signal 0	< 0.5 mA
Fuse for UP	10 A fast
Demagnetization with inductive DC load	Via internal varistors (see figure below this table)
Output switching frequency	
With resistive load	On request
With inductive loads	Max. 0.5 Hz
With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes (software-controlled supervision)
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

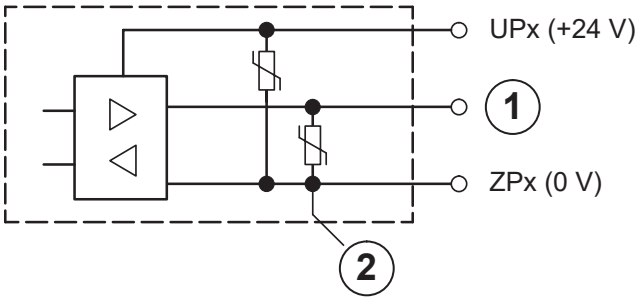


Fig. 183: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to an FBP interface module or CS31 bus module.

Parameter	Value
Used inputs	DC16 / DC17
Used outputs	DC18
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for AI0+ to AI3+		Terminal 3.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C

Parameter	Value
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 % For XC version below 0 °C and above 60 °C: on request
Relationship between input signal and hex code	↳ *Chapter 1.6.3.6.3.1.1.9.1 “Input ranges voltage, current and digital input” on page 2999* ↳ *Chapter 1.6.3.6.3.1.1.9.2 “Input ranges resistance temperature detector” on page 3000*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 3.5 and 3.6
Reference potential for AO0+ to AO1+		Terminal 3.7 (AO-) for voltage output Terminals 1.9, 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current

Parameter	Value
Bipolar	Voltage
Galvanic isolation	Against internal supply and other modules
Configurability	-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load) as current output	0 Ω...500 Ω
Output loadability as voltage output	±10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	*Chapter 1.6.3.6.3.1.1.9.3 “Output ranges voltage and current” on page 3002*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	3	5
Digital outputs (bytes)	1	3
Analog inputs (words)	4	4
Analog outputs (words)	2	2
Counter input data (words)	0	4
Counter output data (words)	0	8

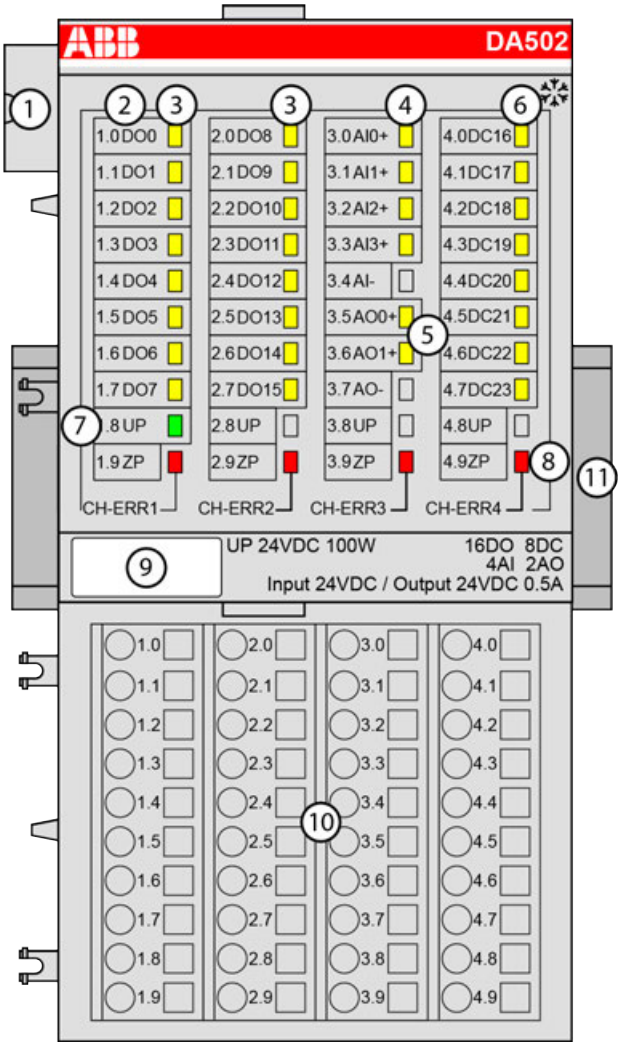
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 700 R0001	DA501, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO	Active
1SAP 450 700 R0001	DA501-XC, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DA502 - Digital/Analog input/output module

- 16 digital outputs, 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- 4 analog inputs, voltage, current and RTD, resolution 12 bits plus sign
- 2 analog outputs, voltage and current, resolution 12 bits plus sign
- Fast counter
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states of the digital outputs DO0 to DO15
4 4 yellow LEDs to display the signal states of the analog inputs AI0 to AI3
5 2 yellow LEDs to display the signal states of the analog outputs AO0 to AO1
6 8 yellow LEDs to display the signal states of the configurable digital inputs/outputs DC16 to DC23
7 1 green LED to display the state of the process supply voltage UP
8 4 red LEDs to display errors
9 Label
10 Terminal unit
11 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes
Power supply	From the process supply voltage UP
LED displays	For system displays, signal states, errors and power supply
Internal supply voltage	Via the I/O bus interface (I/O bus)
External supply voltage	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU515 or TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection is carried out by using the 40 terminals of the terminal unit TU515/TU516 *Chapter 1.6.3.5.2 “TU515, TU516, TU541 and TU542 for I/O modules” on page 2553*.

The assignment of the terminals:

Terminal	Signal	Description
1.0	DO0	Signal of the digital output DO0
1.1	DO1	Signal of the digital output DO1
1.2	DO2	Signal of the digital output DO2
1.3	DO3	Signal of the digital output DO3
1.4	DO4	Signal of the digital output DO4
1.5	DO5	Signal of the digital output DO5
1.6	DO6	Signal of the digital output DO6
1.7	DO7	Signal of the digital output DO7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DO8	Signal of the digital output DO8
2.1	DO9	Signal of the digital output DO9

Terminal	Signal	Description
2.2	DO10	Signal of the digital output DO10
2.3	DO11	Signal of the digital output DO11
2.4	DO12	Signal of the digital output DO12
2.5	DO13	Signal of the digital output DO13
2.6	DO14	Signal of the digital output DO14
2.7	DO15	Signal of the digital output DO15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	AI0+	Positive pole of analog input signal 0
3.1	AI1+	Positive pole of analog input signal 1
3.2	AI2+	Positive pole of analog input signal 2
3.3	AI3+	Positive pole of analog input signal 3
3.4	AI-	Negative pole of analog input signals 0 to 3
3.5	AO0+	Positive pole of analog output signal 0
3.6	AO1+	Positive pole of analog output signal 1
3.7	AO-	Negative pole of analog output signals 0 and 1
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DC16	Signal of the configurable digital input/output DC16
4.1	DC17	Signal of the configurable digital input/output DC17
4.2	DC18	Signal of the configurable digital input/output DC18
4.3	DC19	Signal of the configurable digital input/output DC19
4.4	DC20	Signal of the configurable digital input/output DC20
4.5	DC21	Signal of the configurable digital input/output DC21
4.6	DC22	Signal of the configurable digital input/output DC22
4.7	DC23	Signal of the configurable digital input/output DC23
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DA502.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.3.6 “I/O modules” on page 2569*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

CAUTION!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalization of a low resistance to avoid high potential differences between different parts of the plant.

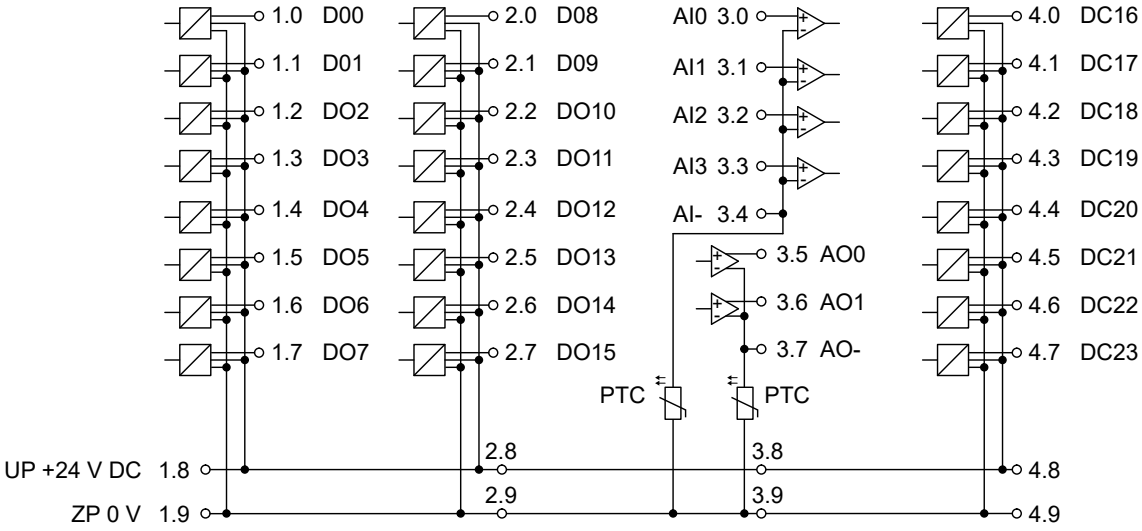
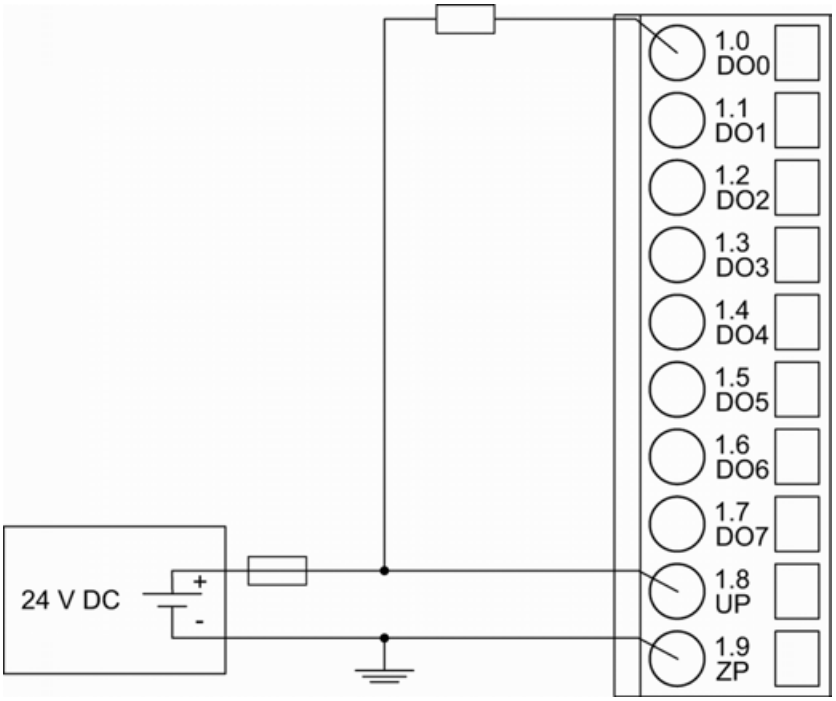


Fig. 184: Terminal assignment of the module

The module provides several diagnosis functions *Chapter 1.6.3.6.3.1.2.7 “Diagnosis” on page 3030*.

Connection of the digital outputs

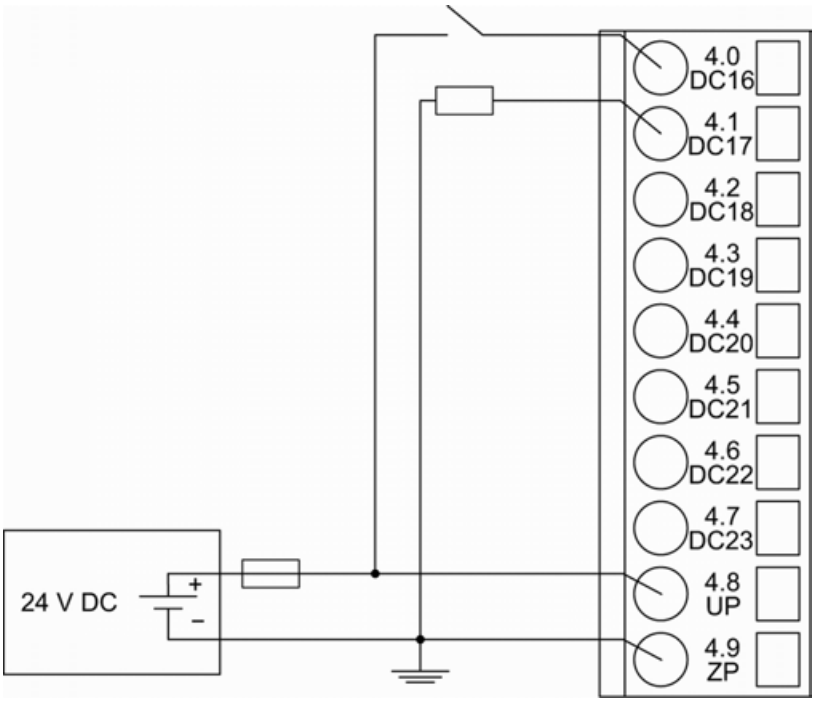
The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 to DO15 in the same way.



For a description of the meaning of the LEDs, please refer to the Displays chapter *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC16 and DC17. DC16 is connected as an input and DC17 is connected as an output. Proceed with the configurable digital inputs/outputs DC18 to DC23 in the same way.



NOTICE!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of DA502.

Connect a 470 Ω / 1 W resistor in series to inputs DC16/DC17 if they are used as fast counter inputs to avoid any influences.

For a description of tthe meaning of the LEDs, please refer to the Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033* chapter.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA502 provides a constant current source which is multiplexed over max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

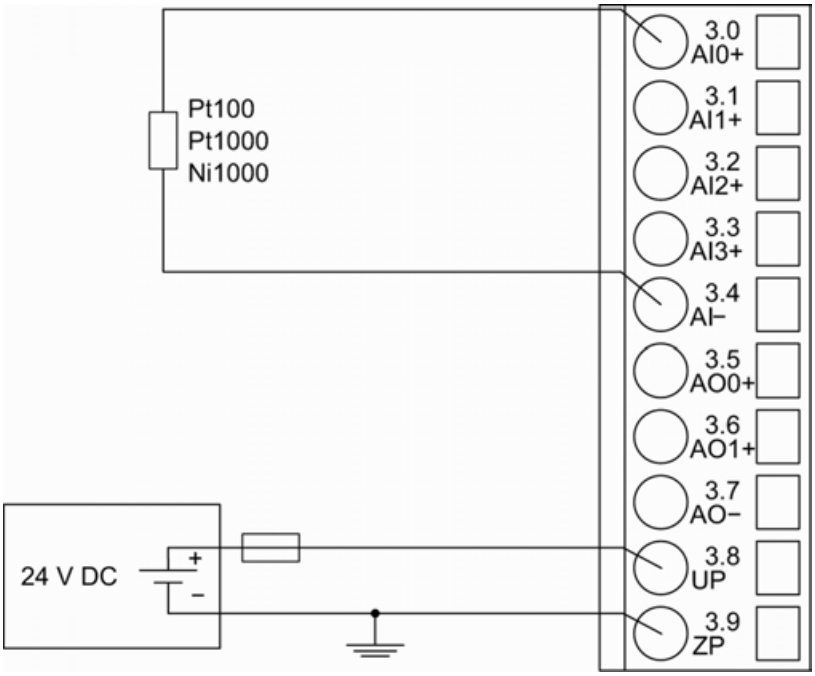


Fig. 185: Connection of resistance thermometers in 2-wire configuration to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

The module DA502 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA502 provides a constant current source which is multiplexed over max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

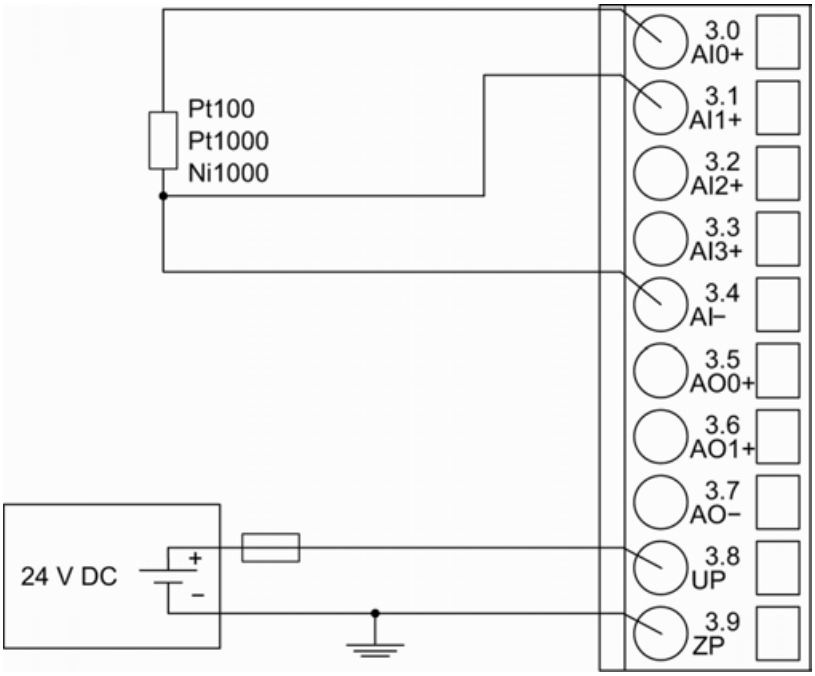


Fig. 186: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

The module DA502 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

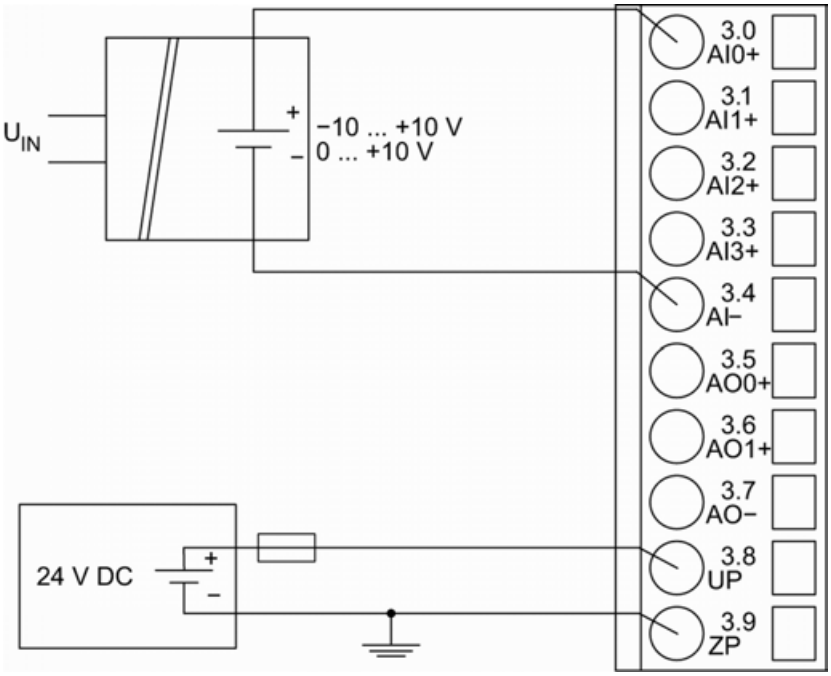


Fig. 187: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

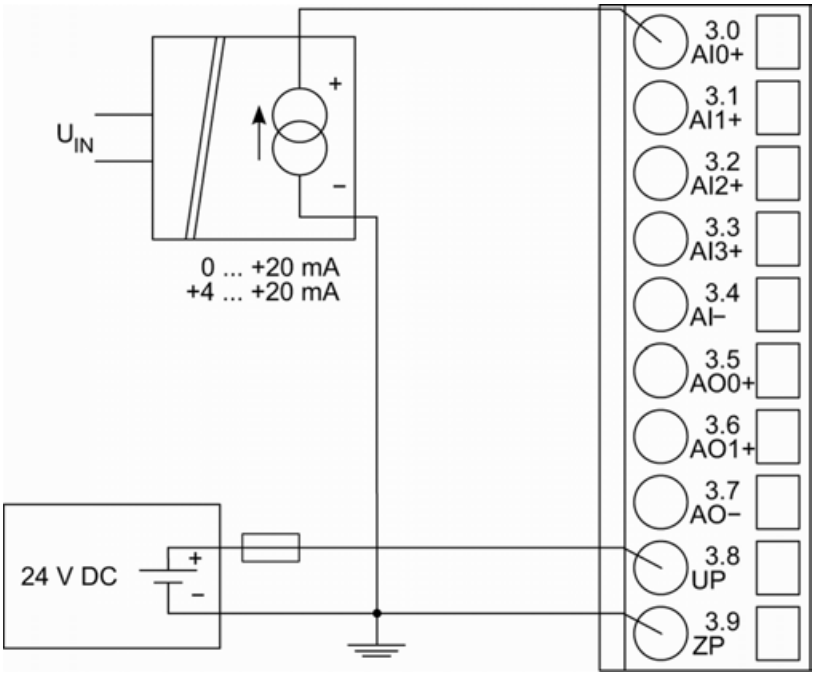


Fig. 188: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

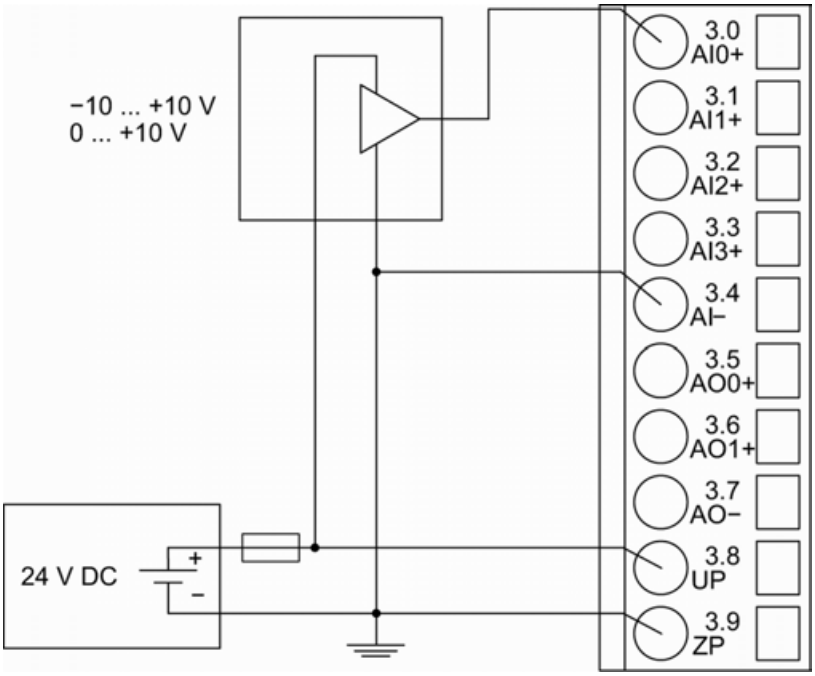


Fig. 189: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following measuring ranges can be configured ↬ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↬ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↬ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

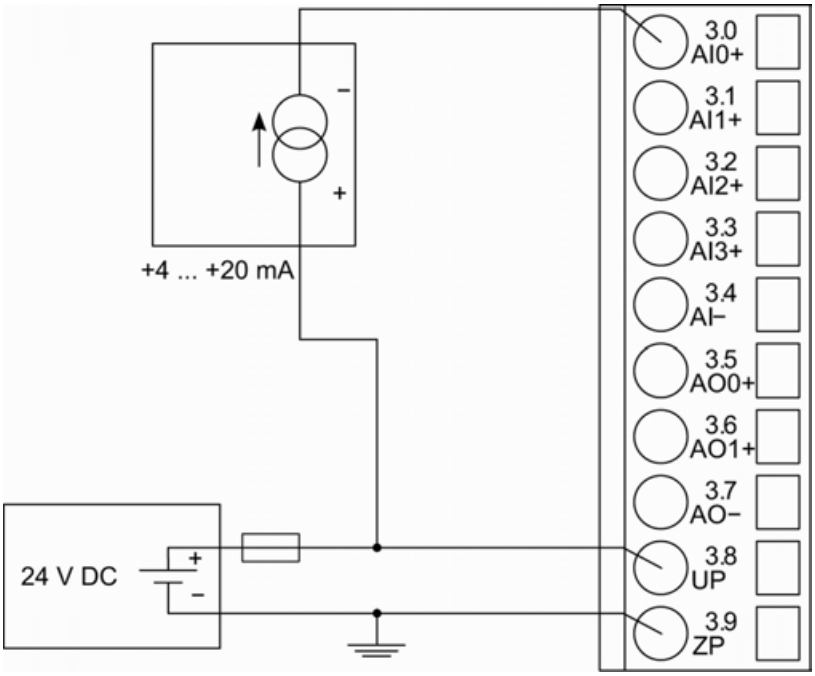


Fig. 190: Connection of passive-type analog sensors (current) to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Current	4 mA...20 mA	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

NOTICE!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt Zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

Using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

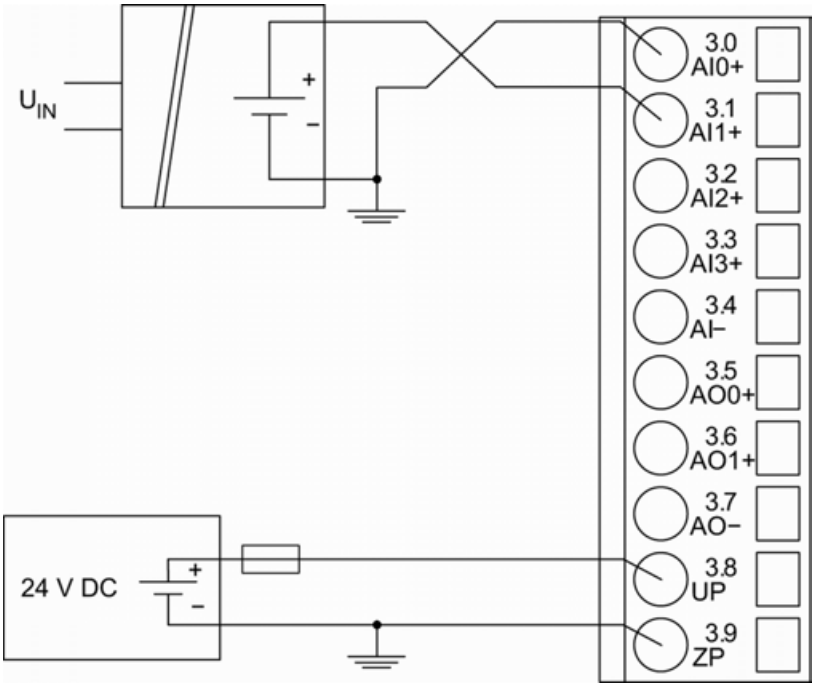


Fig. 191: Connection of active-type analog sensors (voltage) to differential analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

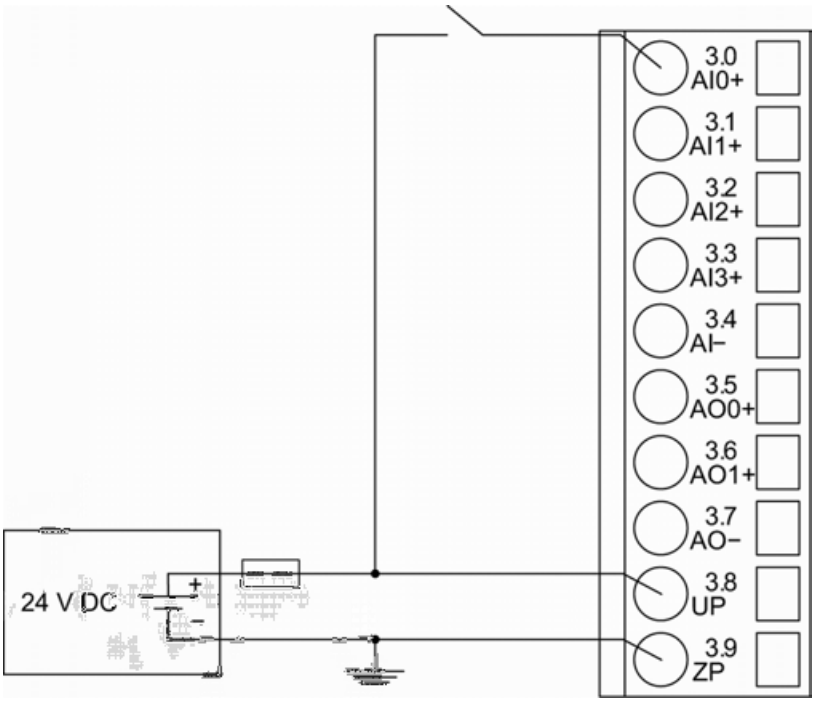


Fig. 192: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033* :

Digital input	24 V	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

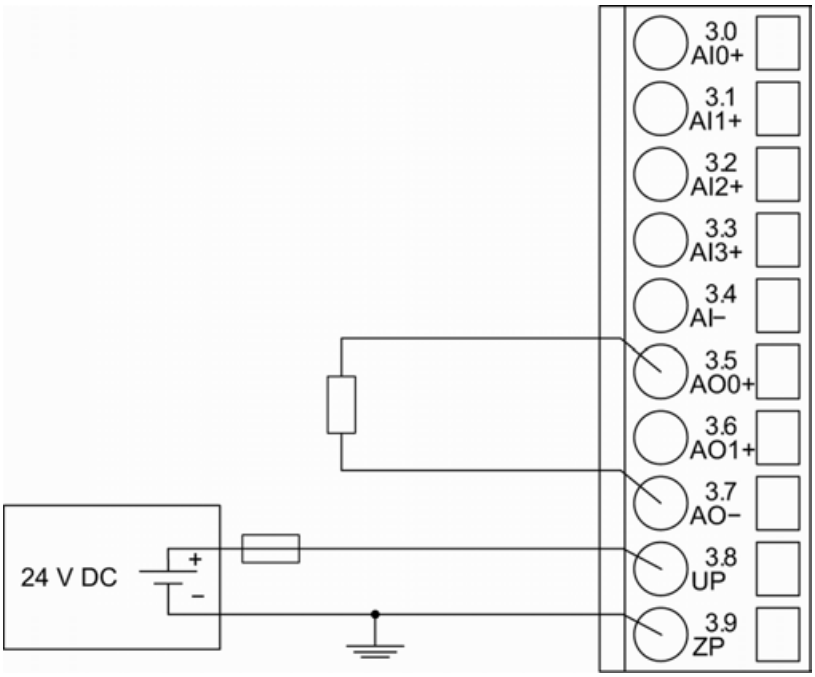


Fig. 193: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

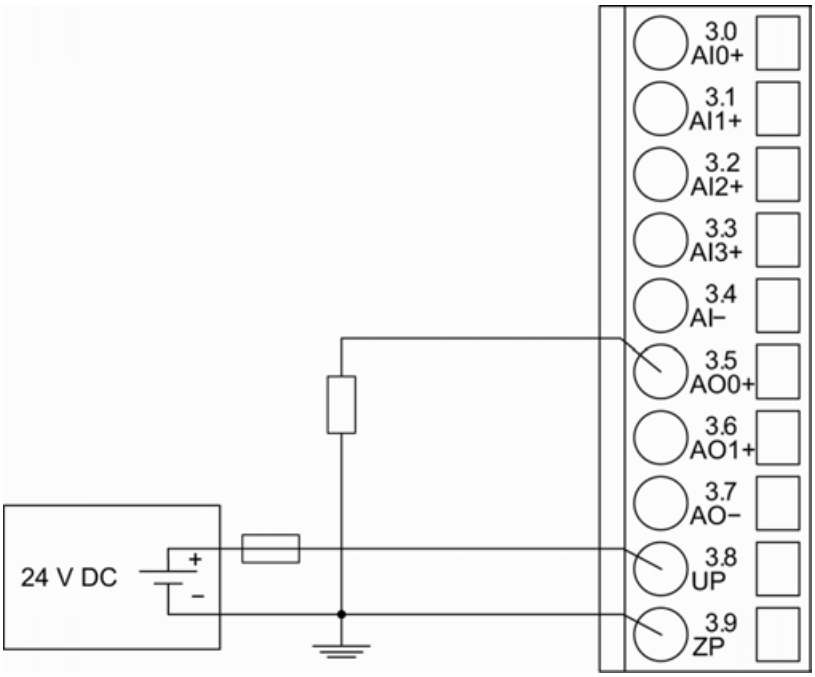


Fig. 194: Connection of analog output loads (current)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.6.3.1.2.6 “Parameterization” on page 3026* ↳ *Chapter 1.6.3.6.3.1.2.9 “Measuring ranges” on page 3033*:

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.3.6.3.1.2.8 “State LEDs” on page 3033*.

Unused analog outputs can be left open-circuited.

Internal data exchange

	Without the fast counter	**With the fast counter (only with AC500)**
Digital inputs (bytes)	1	1
Digital outputs (bytes)	3	3
Analog inputs (words)	4	4
Analog outputs (words)	2	2
Counter input data (words)	0	5
Counter output data (words)	0	9

I/O configuration

The module itself does not store configuration data. It draws its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Module ID [1])	Internal	1815	WORD	1815	0x0Y01
Ignore module	Internal	Yes No	BYTE	No	
Parameter length	Internal	8	BYTE	8	0xY02
Check supply	off	0	BYTE	1	0xY03
	on	1			
Fast counter [3])	0 : 10 [2])	0 : 10	BYTE	0	Not for FBP
Behavior outputs at comm. error [5])	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	Off 0x00	0x0Y07

[2])	Setting	Description
	On	Error LED lights up at errors of all error classes, Failsafe mode off
	Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
	Off by E3	Error LED lights up at errors of error classes E1 and E2, Failsafe mode off

[2])	Setting	Description
	On +Failsafe	Error LED lights up at errors of all error classes, Failsafe mode on *)
	Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
	Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission

[2]) For a description of the counter operating modes, please refer to the 'Fast Counter' section
↳ *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*

[3]) With CS31 without the parameter "Fast Counter"

The fast counter of the module does not work if the module is connected to a CS31 bus module.

[5]) The parameter Behavior outputs at comm. error is only analyzed if the Failsafe mode is ON.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0x0Y05
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01	0x0Y06
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000	0x0Y08

*) The parameters Behavior DO at comm. error is only analyzed if the Failsafe mode is ON.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Analog data format	Standard Reserved	0 255	BYTE	0	0x0Y04

*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input 0, Channel configuration	see *Table 531 “ Channel configuration” on page 3028*	see *Table 531 “ Channel configuration” on page 3028*	BYTE	0	0x0Y09
Input 0, Check channel	see *Table 532 “ Channel monitoring” on page 3029*	see *Table 532 “ Channel monitoring” on page 3029*	BYTE	0	0x0Y0A
:	:	:	:	:	
:	:	:	:	:	
Input 3, Channel configuration	see *Table 531 “ Channel configuration” on page 3028*	see *Table 531 “ Channel configuration” on page 3028*	BYTE	0	0x0Y0F
Input 3, Check channel	see *Table 532 “ Channel monitoring” on page 3029*	see *Table 532 “ Channel monitoring” on page 3029*	BYTE	0	0x0Y10

Table 531: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0 V...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

Internal value	Operating modes of the analog inputs, individually configurable
19	3-wire Ni1000 -50 °C...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 532: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
0 Output 0, Channel configuration	see *Table 533 “ Channel configuration” on page 3030*	see *Table 533 “ Channel configuration” on page 3030*	BYTE	0	0x0Y11
Output 0, Check channel	see *Table 534 “ Channel monitoring” on page 3030*	see *Table 534 “ Channel monitoring” on page 3030*	BYTE	0	0x0Y12
Output 0, Substitute value	see *Table 535 “ Substitute value” on page 3030*	see *Table 535 “ Substitute value” on page 3030*	WORD	0	0x0Y13
Output 1, Channel configuration	see *Table 533 “ Channel configuration” on page 3030*	see *Table 533 “ Channel configuration” on page 3030*	BYTE	0	0x0Y14
Output 1, Check channel	see *Table 534 “ Channel monitoring” on page 3030*	see *Table 534 “ Channel monitoring” on page 3030*	BYTE	0	0x0Y15
Output 1, Substitute value	see *Table 535 “ Substitute value” on page 3030*	see *Table 535 “ Substitute value” on page 3030*	WORD	0	0x0Y16

Table 533: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Table 534: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 535: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behavior of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 s	0
Last value for 10 s and then turn off	Last value 10 s	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 s	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 s	Depending on configuration

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus, an acknowledgement of the errors is not necessary. The error message is stored via the LED.

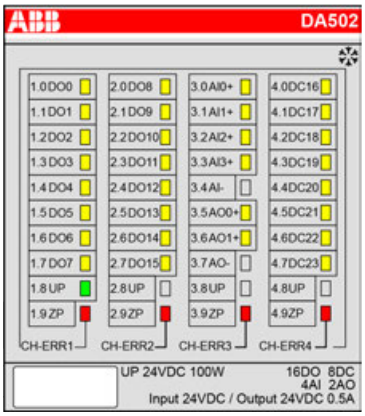
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error DA502								
4	14	1...10	2	0...15	47	Short-circuit at a digital output		Check connection
	11 / 12	ADR	1...10	22...29 [5])				
Channel error DA502								
4	14	1...10	1	16...19 [6])	48	Analog value overflow or broken wire at an analog input		Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	16...19 [6])	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	16...19 [6])	47	Short circuit at an analog input		Check ter-minal
	11 / 12	ADR	1...10					
4	14	1...10	3	20...21 [7])	4	Analog value overflow at an analog output		Check output value
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	3	20...21 [7])	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus: 31 = Module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus = module type (1 = AI, 3 = AO, 4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.
[5])	Ch = 22...29 indicate the digital inputs/outputs DC16...DC23
[6])	Ch = 16...19 indicates the analog inputs AI0...AI3
[7])	Ch = 20...21 indicates the analog outputs AO0...AO1

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	DO0 to DO15	Digital output	Yellow	Output is OFF	Output is ON	--
	DC16 to DC23	Digital input/ output	Yellow	Input/output is OFF	Input/output is ON [1])	--
	AI0 to AI3	Analog input	Yellow	Input is OFF	Input is ON [2])	--
	AO0 to AO1	Analog output	Yellow	Output is OFF	Output is ON [2])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Severe error within the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [3])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) Brightness depends on the value of the analog signal					
	[3]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	> 11.7589	> 11.7589	> 23.5178	> 22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	< 0.0000	< -11.7589	< 0.0000	< 0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	150.0 °C : : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

Technical data of the module

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for UP (+24 V DC) and 1.9, 2.9, 3.9 and 4.9 for ZP (0 V)
	Protection against reverse voltage	yes
	Rated protection fuse at UP	10 A fast
	Rated value	24 V DC
	Max. ripple	5 %
Current consumption		
	From UP	0.07 A + max. 0.5 A per output
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	ca. 2 mA
	Inrush current from UP (at power-up)	0.04 A^2s
Galvanic isolation		Yes, per module
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal mounting or vertical with derating (output load reduced to 50% at 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		16 outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
Connection of the channels		
	DO0 to DO7	Terminals 1.0 to 1.7
	DO8 to DO15	Terminals 2.0 to 2.7

Parameter		Value
Indication of the output signals		1 yellow LED per channel, the LED is ON if the output signal is high (signal 1)
Monitoring point of output indicator		LED is controlled by process CPU
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (channels O0 to O15)	4 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable digital I/O channels can be defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC16...DC23	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels DC16...DC23	Terminals 4.0...4.7

Parameter	Value
Indication of the input/output signals	1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator	LED is part of the input circuitry
Galvanic isolation	Yes, per module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC16 to DC23		Terminals 4.0 to 4.7
Reference potential for all inputs		Terminals 1.9...4.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

* Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC16 to DC23		Terminals 4.0 to 4.7
Reference potential for all outputs		Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	rated value per channel	500 mA at UP = 24 V
	max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
Fuse for UP		10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

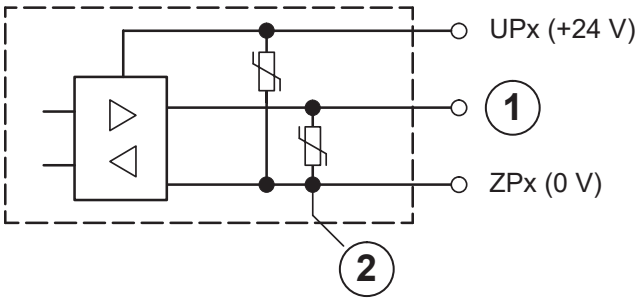


Fig. 195: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a CS31 bus module.

Parameter	Value
Counting frequency	Max. 50 kHz

Chapter 1.6.5.1.12 “Fast counters” on page 3570

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for AI0+ to AI3+		Terminal 3.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 % For XC version below 0 °C and above 60 °C: on request

Parameter	Value
Relationship between input signal and hex code	↳ *Chapter 1.6.3.6.3.1.2.9.1 “Input ranges voltage, current and digital input” on page 3033* ↳ *Chapter 1.6.3.6.3.1.2.9.2 “Input ranges resistance temperature detector” on page 3034*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 3.5 and 3.6
Reference potential for AO0+ to AO1+		Terminal 3.7 (AO-) for voltage output Terminals 1.9, 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules

Parameter	Value
Configurability	-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load), as current output	0 Ω...500 Ω
Output loadability, as voltage output	±10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	↬ *Chapter 1.6.3.6.3.1.2.9.3 “Output ranges voltage and current” on page 3035*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 800 R0001	DA502, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO	Active
1SAP 450 800 R0001	DA502-XC, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.7 Communication interface modules (S500)

Hot swap

System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor modules PM56xx-2ETH with firmware version as of V3.2.0.*

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

Conditions for hot swapping

- *Digital outputs are not under load.*
- *Input/output voltages above safety extra low voltage/ protective extra low voltages (SELV/PELV) are switched off.*
- *Modules are completely plugged on the terminal unit with both snap fit engaged before switching on loads or input/output voltage.*

Hot swap

Further information about hot swap: ↬ Chapter 1.6.5.1.8 “Hot swap” on page 3523.

1.6.3.7.1 Compatibility of communication modules and communication interface modules

Table 536: Modbus TCP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard Ethernet interface	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O

Table 537: PROFINET IO RT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	--	--	hot swap I/O

Table 538: CANopen

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard CAN interface	CI581-CN CI582-CN	--	--	--	remote I/O

Table 539: EtherCAT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-ETHCAT master	CI511-ETHCAT CI512-ETHCAT	x	x	--	remote I/O

1.6.3.7.2 CANopen

Comparison CI581 and CI582

CI581/CI582:
Technical data

Parameter	Value
Interface	CAN
Protocol	CANopen
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the CANopen Node ID for configuration purposes (00h to FFh)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Transmission rates	10 / 20 / 50 / 125 / 250 / 500 / 800 kbit/s 1 Mbit/s Auto transmission rate detection is supported
Bus connection	Depending on used terminal unit TU510: 9-pin D-sub connector TU518: 10-pin terminal block

Parameter		Value
Processor		Hilscher NETX 100
Expandability		CI58x can only be used on onboard CAN interface and without any I/O expansion module *Table 538 “CANopen” on page 3044*.
State display		Module state: PWR/RUN, CN-RUN, CN-ERR, E-ERR, I/O bus
Adjusting elements		2 rotary switches for generation of the node address
Ambient temperature		System data AC500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* System data AC500 XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*
Current consumption		UP: 0.2 A UP3: 0.06 A + 0.5 A max. per output
Weight (without terminal unit)		Ca. 125 g
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	CANopen interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the CANopen Node ID identifier		With 2 rotary switches at the front side of the module
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V
Required terminal unit		TU509, TU510, TU517 or TU518 *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559*

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

The difference of those devices can be found in their input and output characteristics.

CI581-CN: Input/ Output characteristics

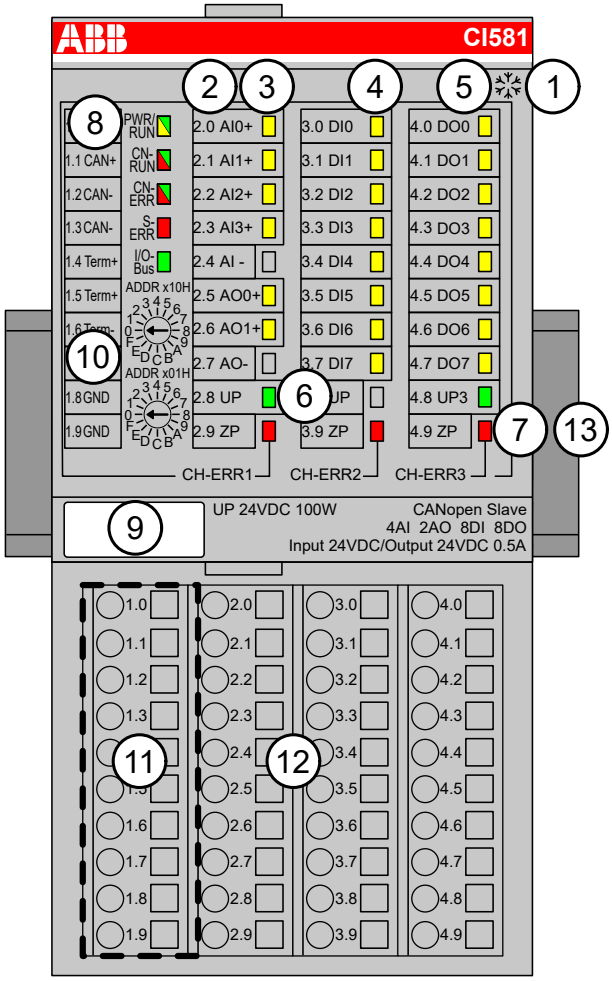
Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC; delay time configurable via software) 8 digital transistor outputs (24 V DC, 0.5 A max.) 4 analog inputs, configurable as: ● -10 V...+10 V ● 0 V...+10 V ● -10 V...+10 V (differential voltage) ● 0 mA...20 mA ● 4 mA...20 mA ● Pt100 , Pt1000, Ni1000 (for each 2-wire and 3-wire) ● 24 V digital input function 2 analog outputs, configurable as: ● -10 V...+10 V ● 0 mA...20 mA ● 4 mA...20 mA
Resolution of the analog channels	12 bits
Fast counter	Integrated, configurable operating modes

CI582-CN: Input/ Output characteristics

Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC) 8 digital transistor outputs (24 V DC, 0.5 A max.) 8 configurable digital inputs/outputs (24 V DC, 0.5 A max.)

CI581-CN

- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal No. and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 System LEDs: PWR/RUN, CN-RUN, CN-ERR, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the CANopen Node ID
11 10 terminals to connect the CANopen bus signals
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The CANopen communication interface module CI581-CN is used as decentralized I/O module in CANopen networks. Depending on the used terminal unit the network connection is performed either via 9-pin female D-sub or via 10 terminals (screw or spring terminals) which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (2.0...2.3)
- 2 analog outputs (2.5...2.6)
- 8 digital inputs 24 V DC in 1 group (3.0...3.7)
- 8 digital outputs 24 V DC in 1 group (4.0...4.7)

The inputs/outputs are galvanically isolated from the CANopen network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter		Value
Interface		CAN
Protocol		CANopen
Power supply		From the process supply voltage UP
Supply of the electronic circuitry of the I/O modules attached		Through the I/O bus interface (I/O bus)
Rotary switches		For setting the CANopen Node ID for configuration purposes (00h to FFh)
LED displays		For system displays, signal states, errors and power supply
External supply voltage		Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Transmission rates		10 / 20 / 50 / 125 / 250 / 500 / 800 kbit/s 1 Mbit/s Auto transmission rate detection is supported
Bus connection		Depending on used terminal unit TU510: 9-pin D-sub connector TU518: 10-pin terminal block
Processor		Hilscher NETX 100
Expandability		CI58x can only be used on onboard CAN interface and without any I/O expansion module ↳ *Table 538 “CANopen” on page 3044*.
State display		Module state: PWR/RUN, CN-RUN, CN-ERR, E-ERR, I/O bus
Adjusting elements		2 rotary switches for generation of the node address
Ambient temperature		System data AC500 ↳ *Chapter 1.6.4.6.1 “System data AC500” on page 3398* System data AC500 XC ↳ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*
Current consumption		UP: 0.2 A UP3: 0.06 A + 0.5 A max. per output
Weight (without terminal unit)		Ca. 125 g
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	CANopen interface against the rest of the module
	Inrush current from UP (at power up)	On request

Parameter		Value
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the CANopen Node ID identifier		With 2 rotary switches at the front side of the module
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V
Required terminal unit		TU509, TU510, TU517 or TU518 *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559*

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

CI581-CN: Input/ Output characteristics

Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC; delay time configurable via software) 8 digital transistor outputs (24 V DC, 0.5 A max.) 4 analog inputs, configurable as: ● -10 V...+10 V ● 0 V...+10 V ● -10 V...+10 V (differential voltage) ● 0 mA...20 mA ● 4 mA...20 mA ● Pt100 , Pt1000, Ni1000 (for each 2-wire and 3-wire) ● 24 V digital input function 2 analog outputs, configurable as: ● -10 V...+10 V ● 0 mA...20 mA ● 4 mA...20 mA
Resolution of the analog channels	12 bits
Fast counter	Integrated, configurable operating modes

Connections

The CANopen communication interface module is plugged on the I/O terminal units TU517 ↳ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559* or TU518 ↳ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559* and accordingly TU509 or TU510. Properly position the module and press until it locks in place.

The connection of the I/O channels is established using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC

Terminal 4.8: process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to the digital outputs!

Reason: External voltages at an output or several outputs may cause other outputs to be supplied via that voltage instead of voltage UP3 (reverse voltage). This ist not the intended use.

CAUTION!

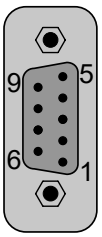
Risk of malfunctions by unintended use!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is connected at the outputs DO0..DO7 and DC0..DC7.

Possibilities of connection

Mounting on terminal units TU509 or TU510

The assignment of the 9-pin female D-sub for the CANopen signals



1	---	Reserved
2	CAN-	Inverted signal of the CAN bus
3	CAN_GND	Ground potential of the CAN bus
4	---	Reserved
5	---	Reserved
6	---	Reserved
7	CAN+	Non-inverted signal of the CAN bus
8	---	Reserved
9	---	Reserved
Shield	Cable shield	Functional earth

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

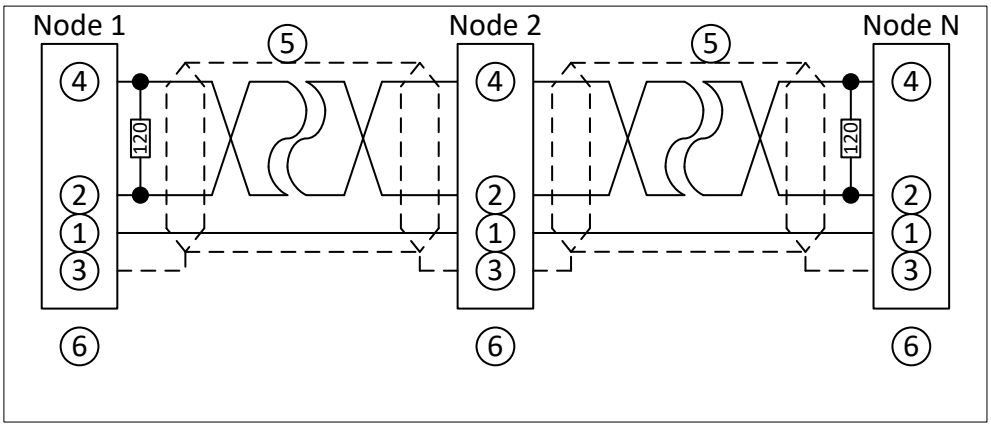


Fig. 196: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

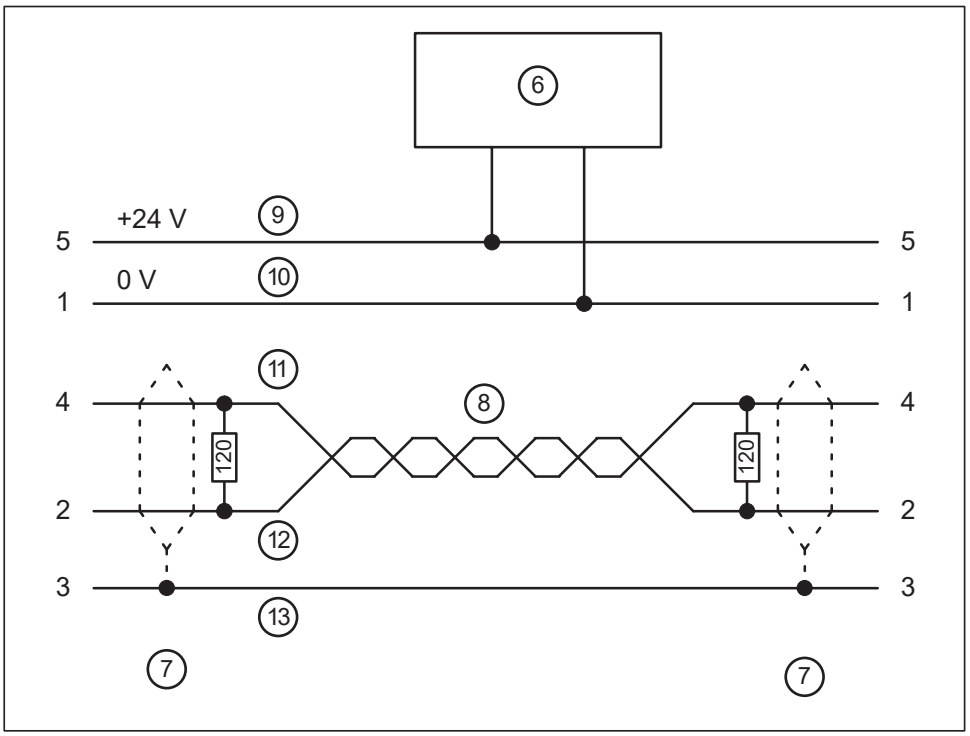


Fig. 197: DeviceNet interface, bus terminating resistors connected to the line ends

6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to Chapter 1.6.4.6.1 “System data AC500” on page 3398.

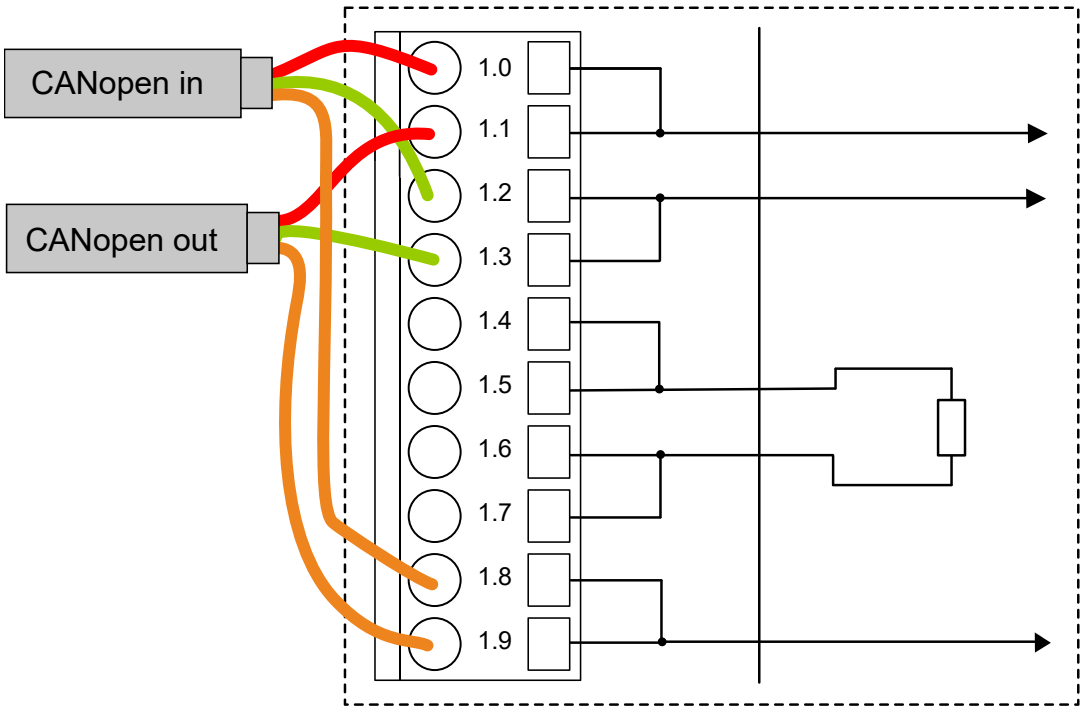
Mounting on terminal units TU517 or TU518

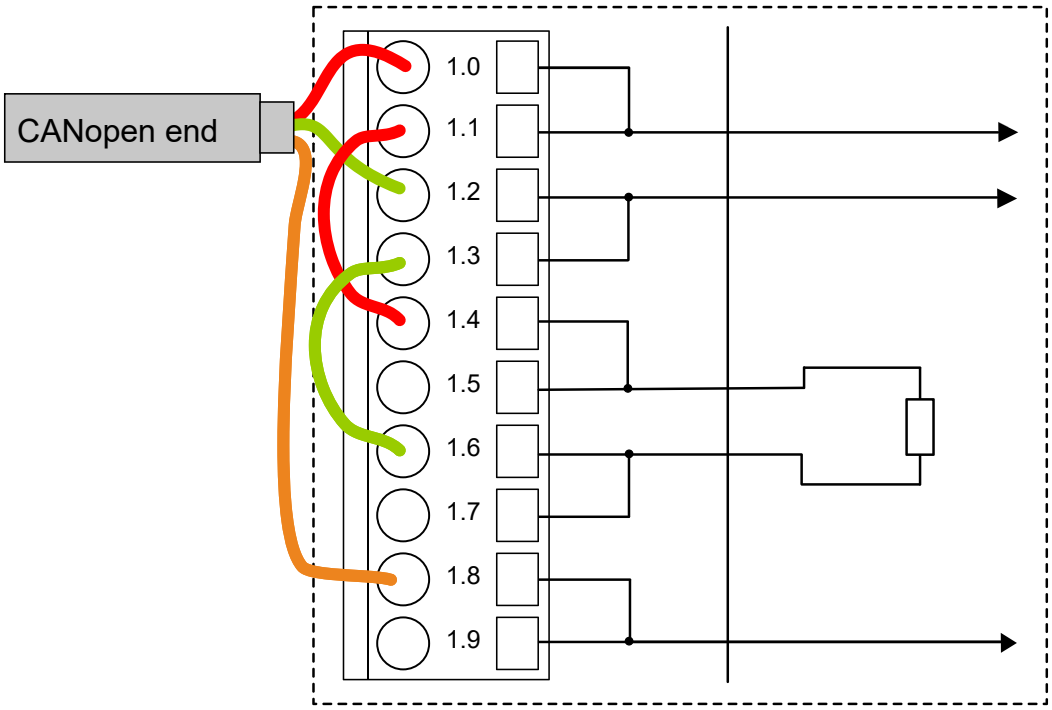
Table 540: Assignment of the terminals

Terminal	Signal	Description
1.0	CAN+	Non-inverted signal of the CAN bus
1.1	CAN+	Non-inverted signal of the CAN bus
1.2	CAN-	Inverted signal of the CAN bus
1.3	CAN-	Inverted signal of the CAN bus
1.4	Term+	CAN bus termination for CAN+ (for bus termination, Term+ must be connected with CAN+)
1.5	Term+	CAN bus termination for CAN+ (connecting alternative for terminal 1.4)
1.6	Term-	CAN bus termination for CAN- (for bus termination, Term- must be connected with CAN-)
1.7	Term-	CAN bus termination for CAN- (connecting alternative for terminal 1.6)
1.8	CAN-GND	Ground potential of the CAN bus
1.9	CAN-GND	Ground potential of the CAN bus

At the line ends of a bus segment, terminating resistors must be connected. If TU517 or TU518 is used, the bus terminating resistors can be enabled by connecting the terminals Term+ and Term- to the data lines CAN+ and CAN- (no external terminating resistors are required, see figure below).

The following figures show the different connection options for the CANopen communication interface module:





In the case of TU517/TU518, the terminating resistors are not located inside the TU but inside the communication interface module CI581-CN. Hence, when removing the device from the TU, the bus terminating resistors are no longer connected to the bus. The bus itself will not be disconnected if a device is removed.

The grounding of the shield should take place at the switchgear cabinet. Please refer to the AC500 System-Data Chapter 1.6.4.6.1 “System data AC500” on page 3398.

Table 541: Assignment of the other terminals

Terminal	Signal	Description
2.0	AI0+	Positive pole of analog input signal 0
2.1	AI1+	Positive pole of analog input signal 1
2.2	AI2+	Positive pole of analog input signal 2
2.3	AI3+	Positive pole of analog input signal 3
2.4	AI-	Negative pole of analog input signals 0 to 3
2.5	AO0+	Positive pole of analog output signal 0
2.6	AO1+	Positive pole of analog output signal 1
2.7	AI-	Negative pole of analog output signals 0 and 1
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DI0	Signal of the digital input DI0
3.1	DI1	Signal of the digital input DI1
3.2	DI2	Signal of the digital input DI2
3.3	DI3	Signal of the digital input DI3

Terminal	Signal	Description
3.4	DI4	Signal of the digital input DI4
3.5	DI5	Signal of the digital input DI5
3.6	DI6	Signal of the digital input DI6
3.7	DI7	Signal of the digital input DI7
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DO0	Signal of the digital output DO0
4.1	DO1	Signal of the digital output DO1
4.2	DO2	Signal of the digital output DO2
4.3	DO3	Signal of the digital output DO3
4.4	DO4	Signal of the digital output DO4
4.5	DO5	Signal of the digital output DO5
4.6	DO6	Signal of the digital output DO6
4.7	DO7	Signal of the digital output DO7
4.8	UP3	Process voltage UP3 (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

Connection of CANopen communication interface module CI581-CN:

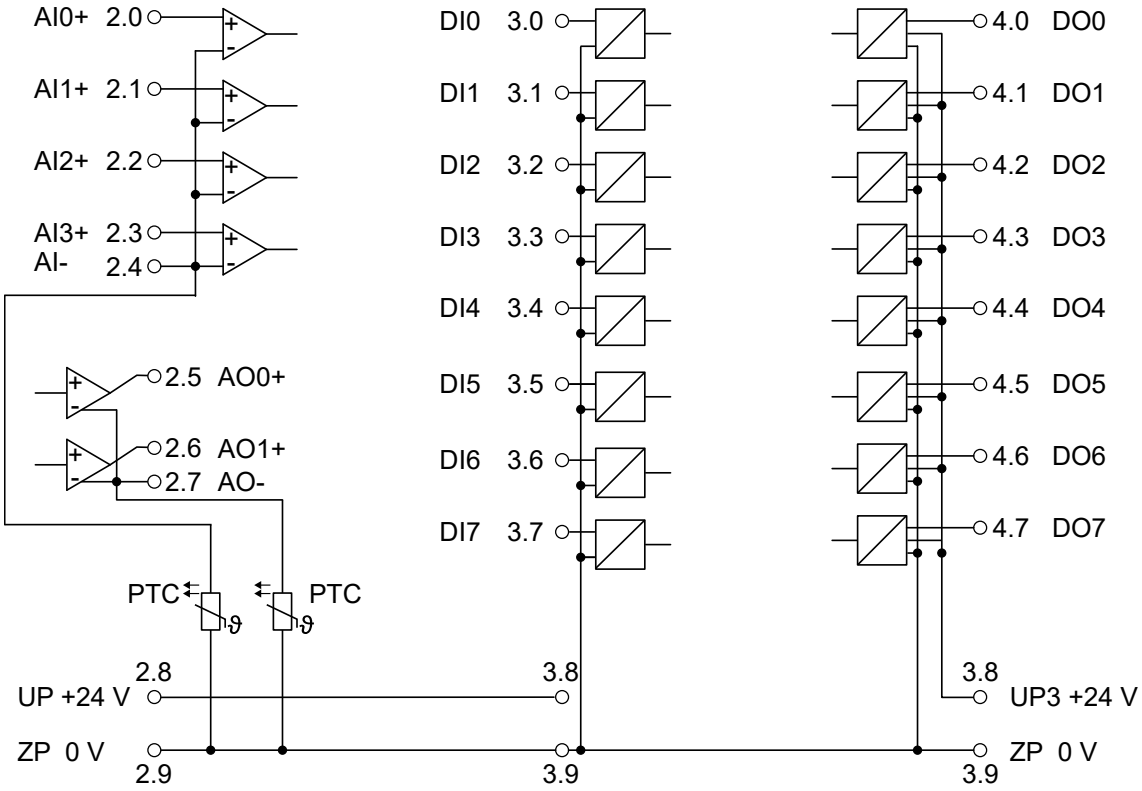


Fig. 198: Connection of the communication interface module CI581-CN

The module provides several diagnosis functions ↳ *Chapter 1.6.3.7.2.2.8 “Diagnosis” on page 3072*.

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

The meaning of the LEDs is described in the section for the state LEDs ↳ *Chapter 1.6.3.7.2.2.9 “State LEDs” on page 3075*.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
50 kbit/s	1000 m

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.

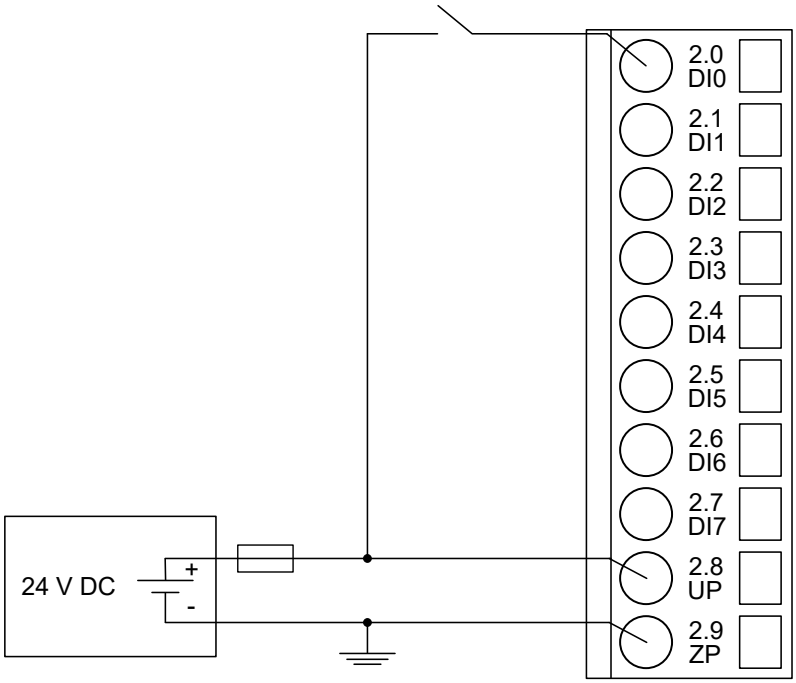


Fig. 199: Connection of the digital inputs to the module CI581-CN

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

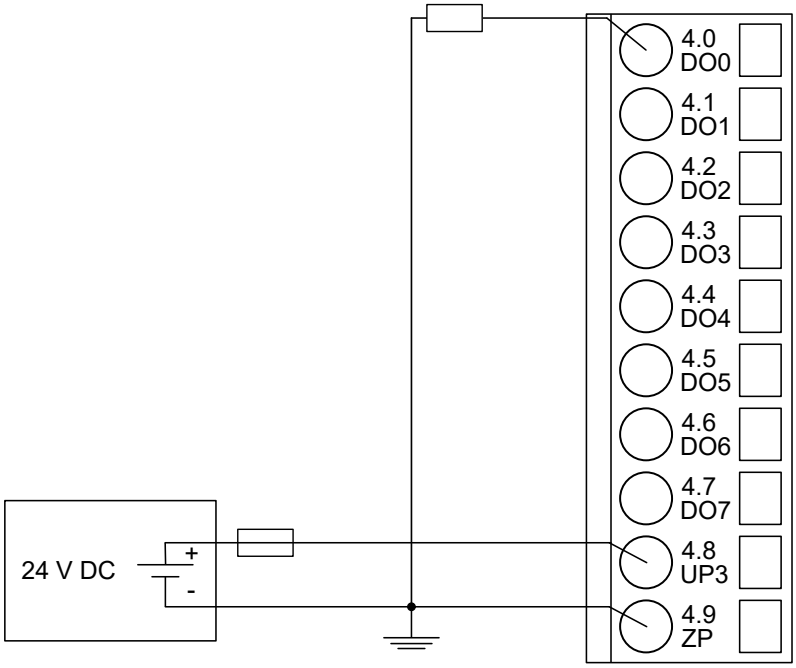


Fig. 200: Connection of configurable digital inputs/outputs to the module CI581-CN

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow to build the necessary voltage drop for the evaluation. For this, the module CI581-CN provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

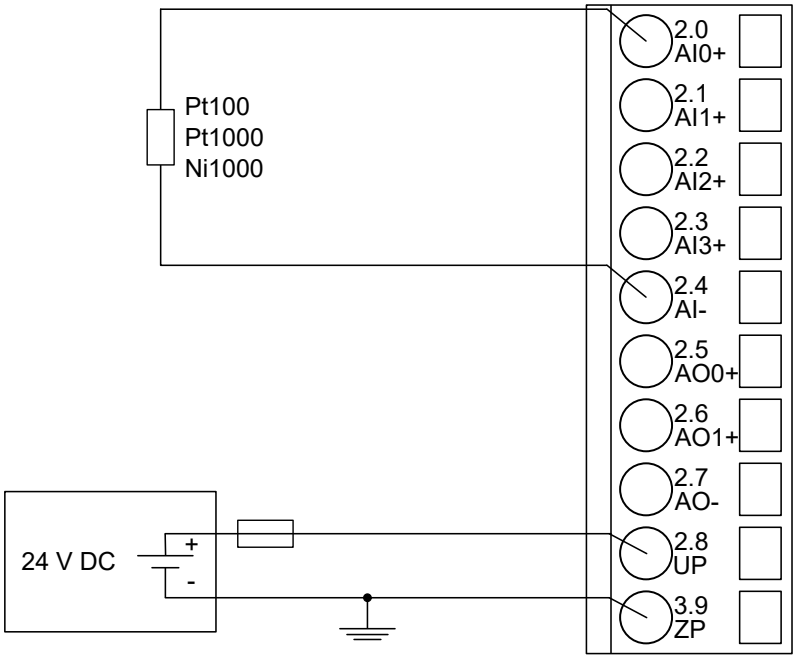


Fig. 201: Connection of resistance thermometers in 2-wire configuration to the analog inputs

Pt100	2-wire configuration, 1 channel used
Pt1000	2-wire configuration, 1 channel used
Ni1000	2-wire configuration, 1 channel used

For the measuring ranges that can be configured, please refer to sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

The module CI581-CN performs a linearization of the resistance characteristic.

To avoid error messages, configure unused analog input channels as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI581-CN provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

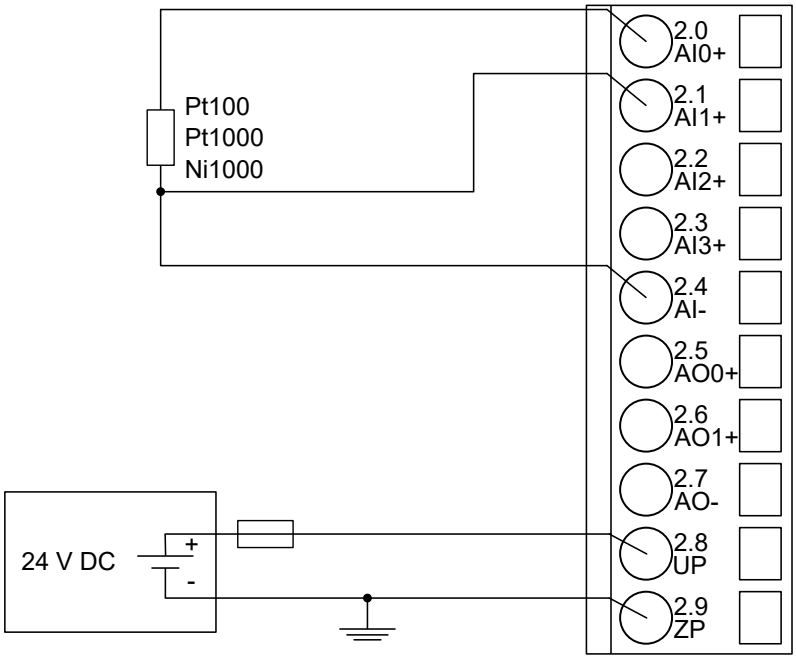


Fig. 202: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	3-wire configuration, 2 channels used
Pt1000	3-wire configuration, 2 channels used
Ni1000	3-wire configuration, 2 channels used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

The module CI581-CN performs a linearization of the resistance characteristic.

To avoid error messages, configure unused analog input channels as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

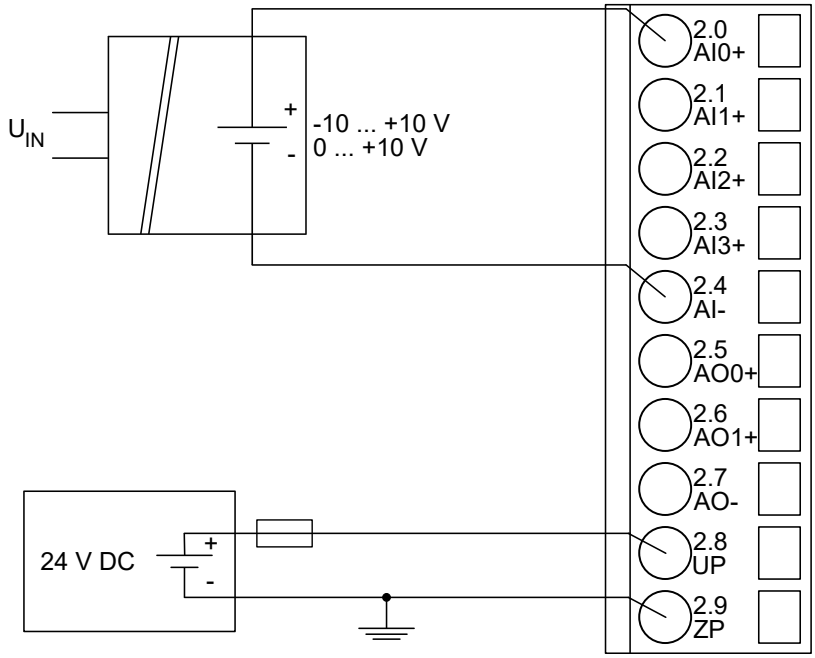


Fig. 203: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

To avoid error messages, configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

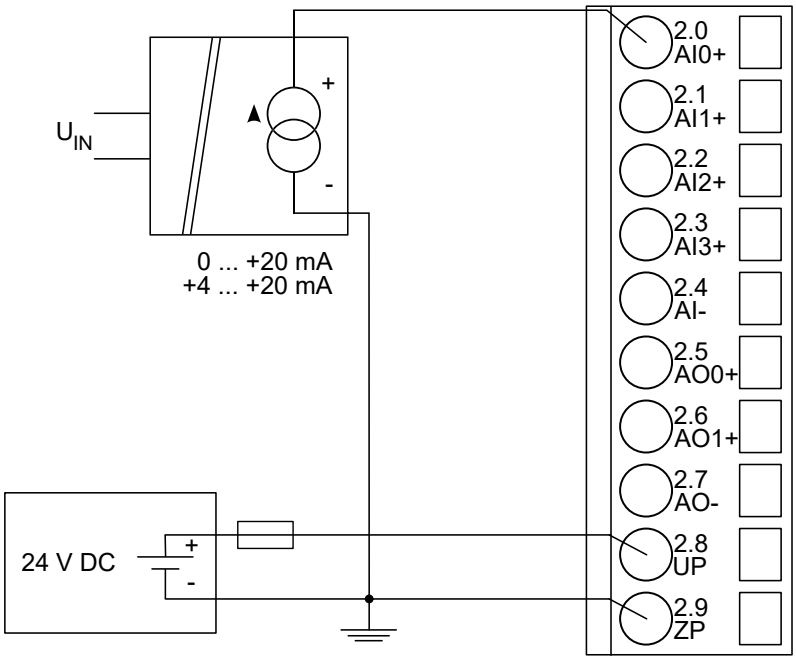


Fig. 204: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

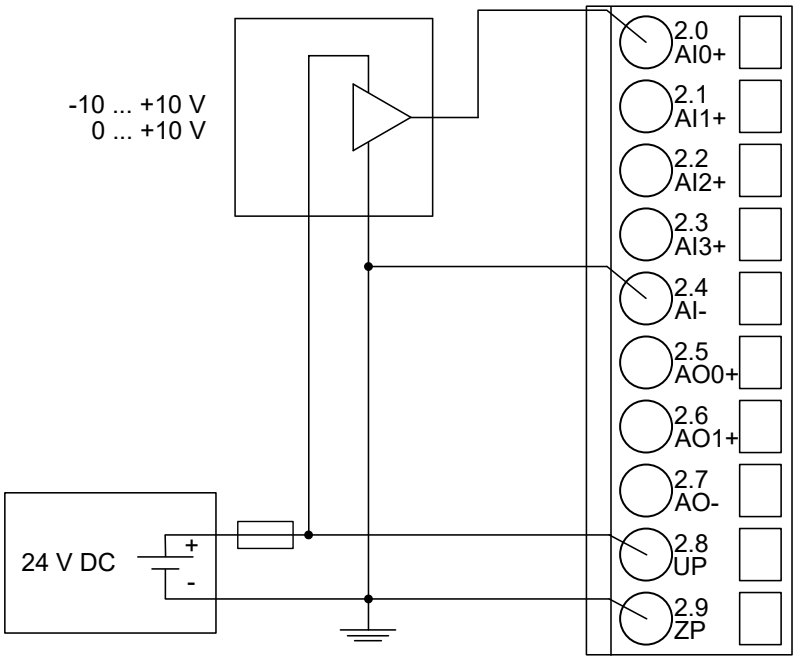


Fig. 205: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For the measuring ranges that can be configured, plese refer to the sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

To avoid error messages, configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

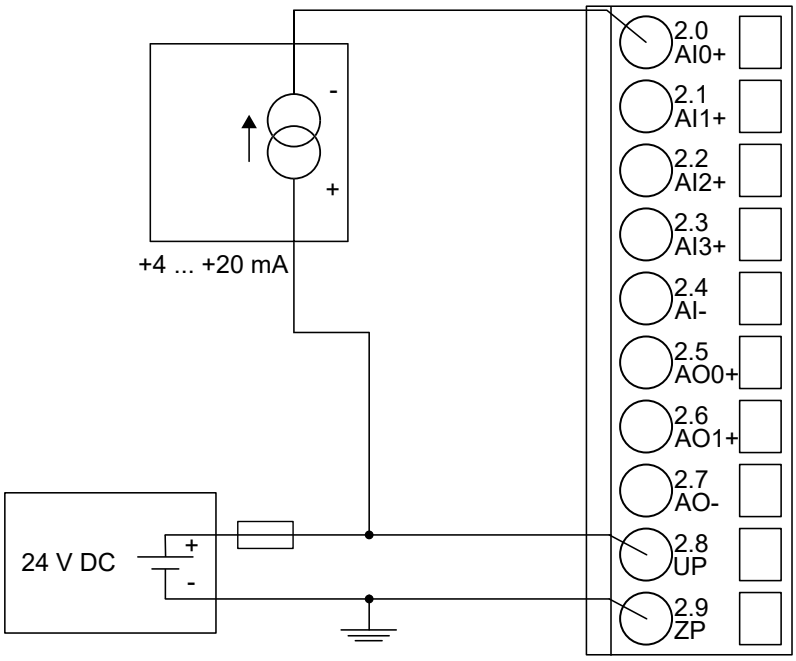


Fig. 206: Connection of passive-type analog sensors (current) to the analog inputs

Current	4...20 mA	1 channel used

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Only use sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt Zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

Using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

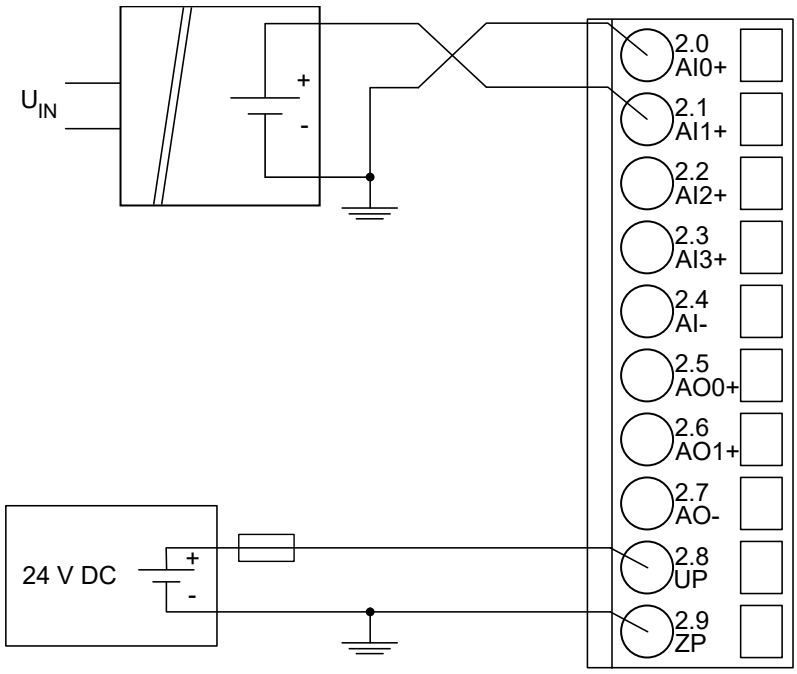


Fig. 207: Connection of active-type analog sensors (voltage) to differential analog inputs

Voltage	0...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↳ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

To avoid error messages, configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

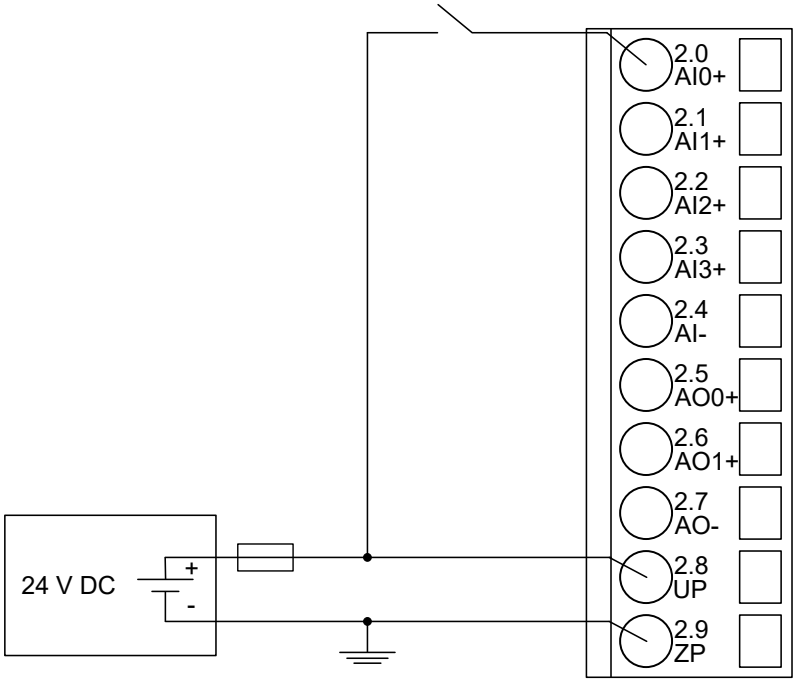


Fig. 208: Use of analog inputs as digital inputs

Digital input	24 V	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↬ *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization ↬ *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

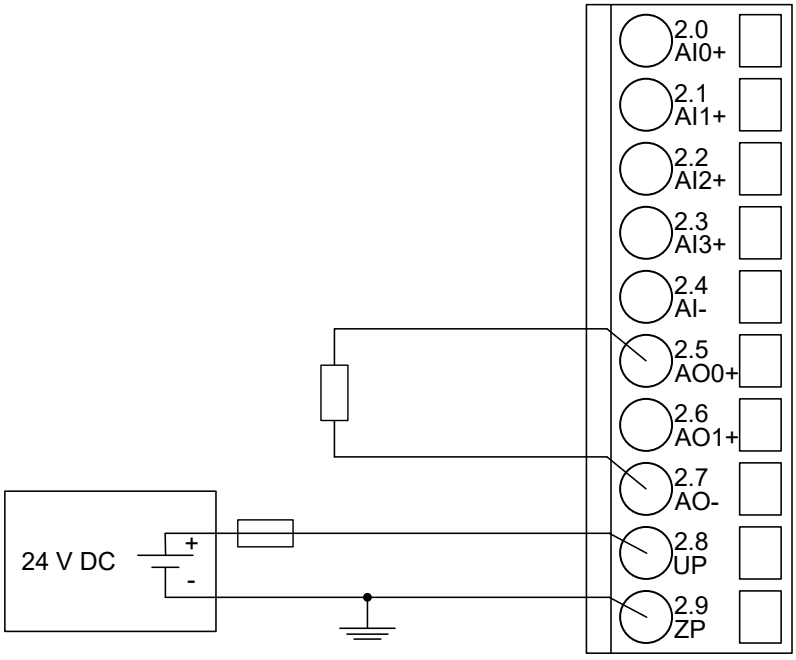


Fig. 209: Connection of analog output loads (voltage)

Voltage	-10 V...+10 V	Load ± 10 mA max.	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

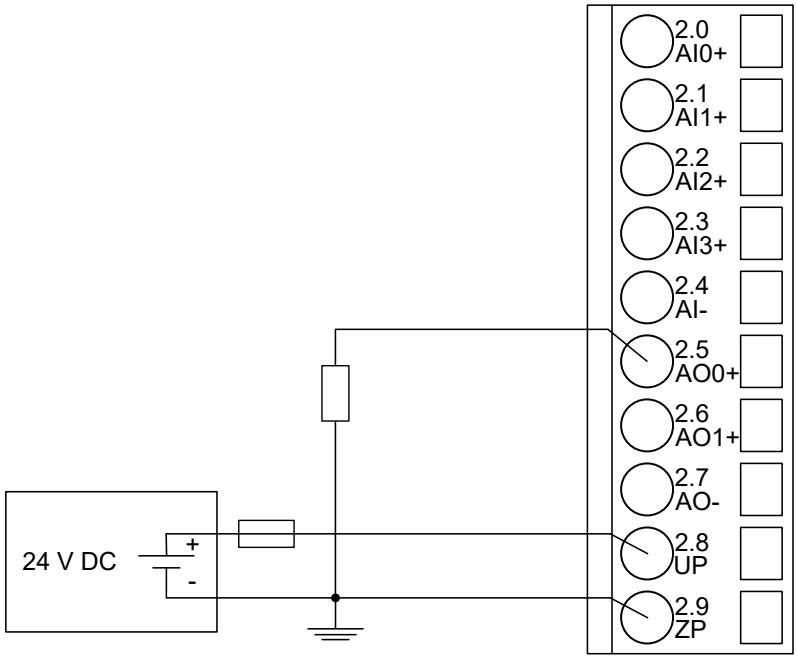


Fig. 210: Connection of analog output loads (current)

Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges *Chapter 1.6.3.7.2.2.10 “Measuring ranges” on page 3077* and Parameterization *Chapter 1.6.3.7.2.2.7 “Parameterization” on page 3067*.

Unused analog outputs can be left open-circuited.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

A detailed description concerning addressing can be found in the documentation of ABB Control Builder Plus Software.

The CANopen communication interface module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

The range of permitted CANopen slave addresses is 1 to 127. Setting a higher address (> 128) does not lead to an error response, but results in a special mode (DS401). In this special mode, the device creates the node address by subtracting the value 128 from the address switch's value.

I/O configuration

The CI582-CN CANopen bus configuration is handled by CANopen master with the exception of the slave node ID (via rotary switches) and the transmision rate (automatic detection).

The digital I/O channels and the fast counter are configured via software.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	0x1C84	WORD	0x1C84
Parameter length	Internal	54	BYTE	54
Error LED / Failsafe function (table error LED / Failsafe function ↳ *Further information on page 3067*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	2		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	18		
Reserved	0	0	ARRAY of 24 BYTES	
Check supply (UP and UP3)	On	0	BYTE	
	Off	1		1
Fast counter	0	0	BYTE	0
	:	:		
	10 [2])	10		

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission

[2]) For a description of the counter operating modes, please refer to the fast counter section ↳ *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*.

Table 542: Settings "Error LED / Failsafe function"

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode on *)
*) The parameters Behaviour analog outputs at communication error and Behaviour digital outputs at communication error are only evaluated if the failsafe function is enabled.	

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard Reserved	0 255	BYTE	0
Behavior analog outputs at communication error *)	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	0
*) The parameter Behavior analog outputs at communication error is only analyzed if the failsafe mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Operation modes of analog inputs	Operation modes of analog inputs	BYTE	0
Input 0, Check channel	Settings channel monitoring	Settings channel monitoring	BYTE	0
:	:	:	:	:
:	:	:	:	:

Name	Value	Internal value	Internal value, type	Default
Input 3, Channel configuration	Operation modes of analog inputs	Operation modes of analog inputs	BYTE	0
Input 3, Check channel	Settings channel monitoring	Settings channel monitoring	BYTE	0

Table 543: Channel configuration - Operating modes of the analog inputs

Internal Value	Operating Modes (individually configurable)
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Table 544: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Operation modes of analog outputs	Operation modes of analog outputs	BYTE	0
Output 0, Check channel	Channel monitoring	Channel monitoring	BYTE	0
Output 0, Substitute value	Substitute value	Substitute value	WORD	0
Output 1, Channel configuration	Operation modes of analog outputs	Operation modes of analog outputs	BYTE	0
Output 1, Check channel	Channel monitoring	Channel monitoring	BYTE	0
Output 1, Substitute value	Substitute value	Substitute value	WORD	0

Table 545: Channel configuration - Operating modes of the analog outputs

Internal value	Operating Modes (individually configurable)
0 (default)	Not used
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 546: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 547: Substitute value

Intended Behavior of Output Channel when the Control System Stops	Required Setting of the Module Parameter "Behavior of Outputs in Case of a Communication Error"	Required Setting of the Channel Parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration

Intended Behavior of Output Channel when the Control System Stops	Required Setting of the Module Parameter "Behavior of Outputs in Case of a Communication Error"	Required Setting of the Channel Parameter "Substitute value"
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behavior digital outputs at communcation error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 255	00h ... FFh	BYTE	0 0x00
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	Off 0x00

[1]) The parameter Behavior digital outputs at communcation error is only analyzed if the failsafe mode is ON.

[2]) The state "externally voltage detected" appears if the output of a channel DC0..DC7 is to be switched on while an external voltage is connected ↳ *Chapter 1.6.3.7.2.2.3 “Connections” on page 3050*. In this case, the start-up is disabled as long as the external voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".

Diagnosis

Byte Number	Description	Possible Values
1	Diagnosis byte, slot number	31 = CI581-CN (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus, an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket		Replace I/O module / check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])		Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module		Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low		Check process supply voltage

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**CANopen diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Voltage feedback on deactivated digital output [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at digital output [7])	Check terminals
Channel error analog							
4	-	31	1	0..3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0..3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0..3	47	Short circuit at an analog input	Check terminals
4	-	31	3	0..1	4	Analog value overflow at an analog output	Check output value
4	-	31	3	0..1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500, the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = position of the communication module;14 = I/O bus; 31 = module itself The identifier is not contained in the CI541-DP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = module itself; 1..10 = decentralized communication interface module
[3])	With "Module" the following allocation applies: 31 = module itself Channel error: module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears if external voltages at one or more terminals DO0..DO7 cause other digital outputs to be fed by that voltage (voltage feedback, description in 'Connections' ↯ *Chapter 1.6.3.7.2.2.3 “Connections” on page 3050*). All outputs of the digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0..DO7 has overrun the process supply voltage UP3 (description in 'Connections' ↯ *Chapter 1.6.3.7.2.2.3 “Connections” on page 3050*). Diagnosis message appears for the whole module.
[6])	This message appears if the output of a channel DO0..DO7 is to be switched on while an external voltage is connected. In this case, start-up is disabled while the external voltage is connected. Otherwise, this could produce reverse voltage flowing from this output to other digital outputs. This diagnosis message appears for each channel.
[7])	Short circuit: After a short circuit has been detected, the output is deactivated for 100ms seconds. Subsequently, a new start-up will be executed. This diagnosis message appears for each channel.

State LEDs

The state LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, CN-RUN, CN-ERR, S-ERR and I/O bus) show the operation states of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O controller	Start-up / preparing communication
	Yellow	---	---	---

LED	Color	OFF	ON	Flashing
CN-RUN	Green	---	Device configured, CANopen bus in OPERATIONAL state and cyclic data exchange running	Flashing: CANopen bus in PRE-OPERATIONAL state and slave is being configured Single flash: CANopen bus in STOPPED state. Flickering: Autodetect is active
CN-ERR	Red	No system error	CANopen Bus is OFF	Flashing: Configuration error Single flash: error counter overflow due to too many error frames Double flash: A node-guard or a heartbeat event occurred Flickering: Autodetect is active
S-ERR	Red	No error	Internal error	--
I/O bus	Green	No decentralized I/O modules connected or communication error	Decentralized I/O modules connected and operational	---

States of the 27 process LEDs:

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--

LED	Color	OFF	ON	Flashing
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
			800 : 701	0320 : 02BD
Normal range	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 3.0 to 3.7
Reference potential for all inputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC
Signal 0	-3 V...+5 V
Undefined signal	> +5 V...< +15 V
Signal 1	+15 V...+30 V
Ripple with signal 0	Within -3 V...+5 V
Ripple with signal 1	Within +15 V...+30 V
Input current per channel	
Input voltage +24 V	Typ. 5 mA
Input voltage +5 V	> 1 mA

Parameter	Value
Input voltage +15 V	> 2 mA
Input voltage +30 V	< 8 mA
Max. cable length	
Shielded	1000 m
Unshielded	600 m

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DO0 to DO7	Terminals 4.0 to 4.7
Reference potential for all outputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1	UP3 (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	
Rated value per channel	500 mA at UP3 = 24 V
Max. value (all channels together)	4 A
Leakage current with signal 0	< 0.5 mA
Fuse for UP3	10 A fast
Demagnetization with inductive DC load	Via internal varistors (see figure below this table)
Output switching frequency	
With resistive load	On request
With inductive loads	Max. 0.5 Hz
With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes (software-controlled supervision)
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

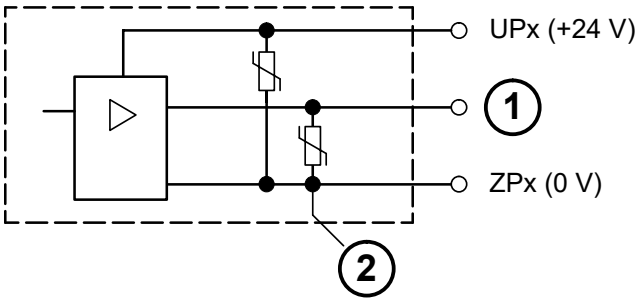


Fig. 211: Digital input/output (circuit diagram)

1	Digital output
2	Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 2.0 to2.3
Reference potential for AI0+ to AI3+		Terminal 2.4 (AI-) for voltage and RTD measurement Terminal 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10...+10 V
Galvanic isolation		Against CANopen Bus
Configurability		0...10 V, -10...+10 V, 0/4...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C

Parameter	Value
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Tables Input Ranges Voltage, Current ↳ *Chapter 1.6.3.7.2.2.10.1 “Input ranges voltage, current and digital input” on page 3077* and Digital Input and IInput range resistance temperature detector ↳ *Chapter 1.6.3.7.2.2.10.2 “Input ranges resistance temperature detector” on page 3077*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 2.0 to 2.3
Reference potential for the inputs		Terminals 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 VDC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+15 V
	Signal 1	+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 1.5...1.6
Reference potential for AO0+ to AO1+		Terminal 2.7 (AO-) for voltage output Terminal 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current

Parameter		Value
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10...+10 V, 0...20 mA, 4...20 mA (each output can be configured individually)
Output resistance (load), as current output		0...500 Ω
Output loadability, as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		See *Chapter 1.6.3.7.2.2.10.3 “Output ranges voltage and current” on page 3078*
Unused outputs		Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 3.0 (DI0), 3.1 (DI1)
Used outputs	Terminal 4.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	Fast Counter *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
Operating modes	Operating modes *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*

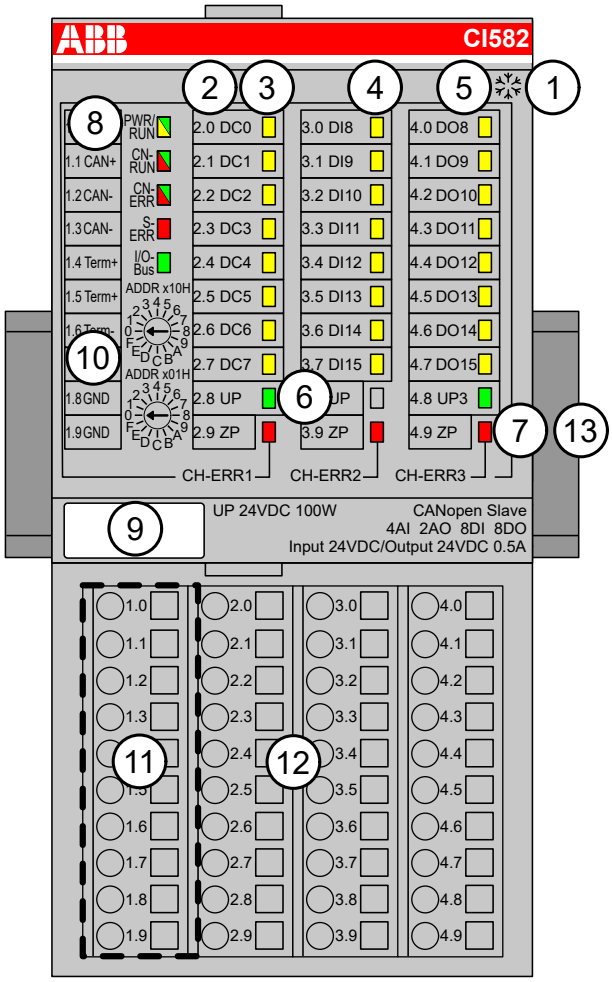
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 228 100 R0001	CI581-CN, CANopen communication interface module with 8 DI, 8 DO, 4 AI and 2 AO	Active
1SAP 428 100 R0001	CI581-CN-XC, CANopen communication interface module with 8 DI, 8 DO, 4 AI and 2 AO, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CI582-CN

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the configurable digital inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 System LEDs: PWR/RUN, CN-RUN, CN-ERR, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the CANopen node ID

11 10 terminals to connect the CANopen bus signals
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The CANopen communication interface module CI582-CN is used as decentralized I/O module in CANopen networks. Depending on the terminal unit used, the network connection is performed either via a female 9-pin D-sub connector or via 10 terminals (screw or spring terminals) which are integrated in the terminal unit. The communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the CANopen network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Interface	CAN
Protocol	CANopen
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the CANopen Node ID for configuration purposes (00h to FFh)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Transmission rates	10 / 20 / 50 / 125 / 250 / 500 / 800 kbit/s 1 Mbit/s Auto transmission rate detection is supported
Bus connection	Depending on used terminal unit TU510: 9-pin D-sub connector TU518: 10-pin terminal block
Processor	Hilscher NETX 100
Expandability	CI58x can only be used on onboard CAN interface and without any I/O expansion module ↳ *Table 538 “CANopen” on page 3044*.
State display	Module state: PWR/RUN, CN-RUN, CN-ERR, E-ERR, I/O bus
Adjusting elements	2 rotary switches for generation of the node address

Parameter		Value
Ambient temperature		System data AC500 ↳ *Chapter 1.6.4.6.1 "System data AC500" on page 3398* System data AC500 XC ↳ *Chapter 1.6.4.7.1 "System data AC500-XC" on page 3450*
Current consumption		UP: 0.2 A UP3: 0.06 A + 0.5 A max. per output
Weight (without terminal unit)		Ca. 125 g
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	CANopen interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the CANopen Node ID identifier		With 2 rotary switches at the front side of the module
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V
Required terminal unit		TU509, TU510, TU517 or TU518 ↳ *Chapter 1.6.3.5.3 "TU517 and TU518 for communication interface modules" on page 2559*

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

CI582-CN: Input/ Output characteristics

Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC) 8 digital transistor outputs (24 V DC, 0.5 A max.) 8 configurable digital inputs/outputs (24 V DC, 0.5 A max.)

Connections

The CANopen communication interface module is plugged on the I/O terminal units TU517 ↬ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559* or TU518 ↬ *Chapter 1.6.3.5.3 “TU517 and TU518 for communication interface modules” on page 2559* and accordingly TU509 or TU510. Properly position the module and press until it locks in place.

The connection of the I/O channels is established using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC

Terminal 4.8: process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

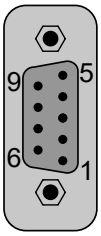
For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Possibilities of connection

Mounting on terminal units TU509 or TU510

The assignment of the 9-pin female D-sub for the CANopen signals



1	---	Reserved
2	CAN-	Inverted signal of the CAN bus
3	CAN_GND	Ground potential of the CAN bus
4	---	Reserved
5	---	Reserved
6	---	Reserved
7	CAN+	Non-inverted signal of the CAN bus
8	---	Reserved

	9	---	Reserved
	Shield	Cable shield	Functional earth

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

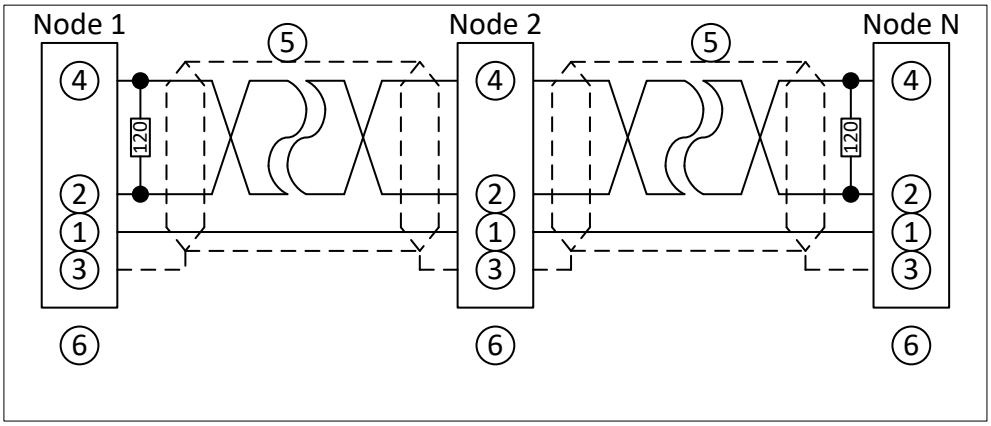


Fig. 212: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

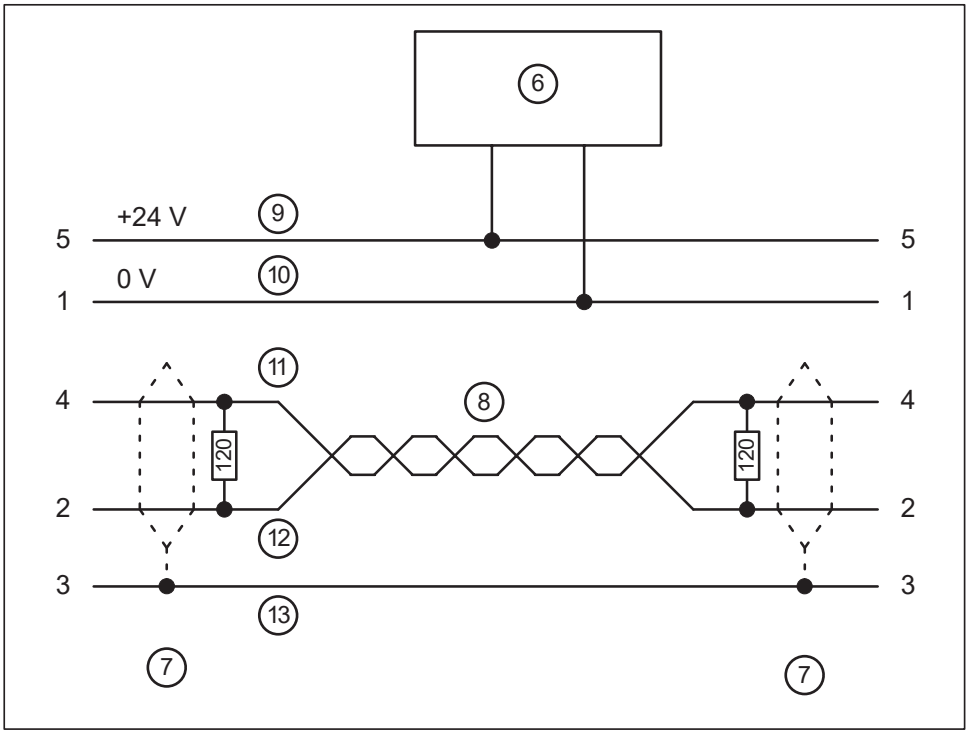


Fig. 213: DeviceNet interface, bus terminating resistors connected to the line ends

6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to ↳ *Chapter 1.6.4.6.1 “System data AC500” on page 3398.*

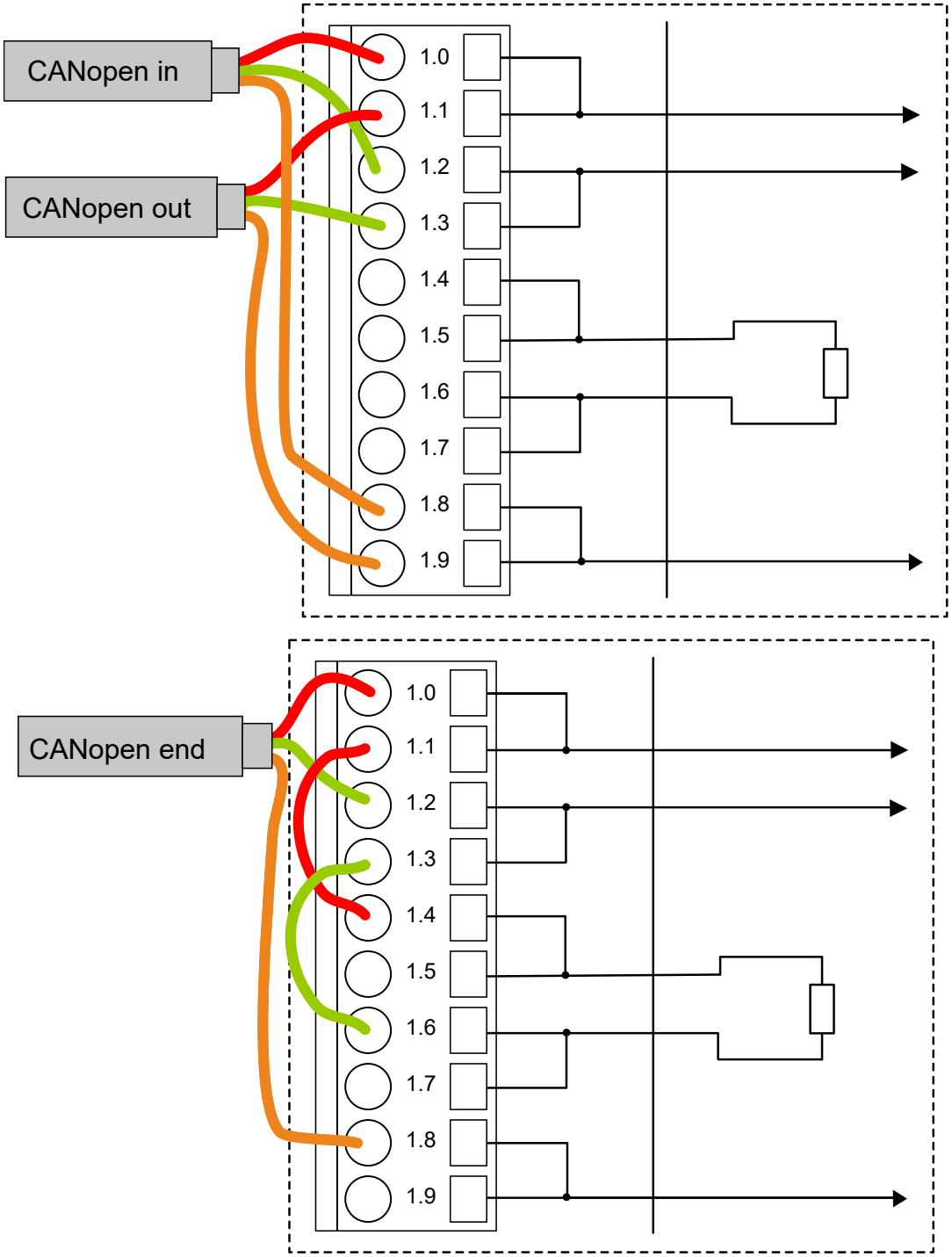
Mounting on terminal units TU517 or TU518

Table 548: Assignment of the terminals

Terminal	Signal	Description
1.0	CAN+	Non-inverted signal of the CAN bus
1.1	CAN+	Non-inverted signal of the CAN bus
1.2	CAN-	Inverted signal of the CAN bus
1.3	CAN-	Inverted signal of the CAN bus
1.4	Term+	CAN bus termination for CAN+ (for bus termination, Term+ must be connected with CAN+)
1.5	Term+	CAN bus termination for CAN+ (connecting alternative for terminal 1.4)
1.6	Term-	CAN bus termination for CAN- (for bus termination, Term- must be connected with CAN-)
1.7	Term-	CAN bus termination for CAN- (connecting alternative for terminal 1.6)
1.8	CAN-GND	Ground potential of the CAN bus
1.9	CAN-GND	Ground potential of the CAN bus

At the line ends of a bus segment, terminating resistors must be connected. If TU517 or TU518 is used, the bus terminating resistors can be enabled by connecting the terminals Term+ and Term- to the data lines CAN+ and CAN- (no external terminating resistors are required, see figure below).

The following figures show the different connection options for the CANopen communication interface module:



In the case of TU517/TU518, the terminating resistors are not located inside the TU but inside the communication interface module CI581-CN. Hence, when removing the device from the TU, the bus terminating resistors are no longer connected to the bus. The bus itself will not be disconnected if a device is removed.

The grounding of the shield should take place at the switchgear cabinet. Please refer to the AC500 System-Data ↳ Chapter 1.6.4.6.1 “System data AC500” on page 3398.

Table 549: Assignment of the other terminals

Terminal	Signal	Description
2.0	DC0	Signal of the configurable digital input/output DC0
2.1	DC1	Signal of the configurable digital input/output DC1
2.2	DC2	Signal of the configurable digital input/output DC2
2.3	DC3	Signal of the configurable digital input/output DC3
2.4	DC4	Signal of the configurable digital input/output DC4
2.5	DC5	Signal of the configurable digital input/output DC5
2.6	DC6	Signal of the configurable digital input/output DC6
2.7	DC7	Signal of the configurable digital input/output DC7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DI8	Signal of the digital input DI8
3.1	DI9	Signal of the digital input DI9
3.2	DI10	Signal of the digital input DI10
3.3	DI11	Signal of the digital input DI11
3.4	DI12	Signal of the digital input DI12
3.5	DI13	Signal of the digital input DI13
3.6	DI14	Signal of the digital input DI14
3.7	DI15	Signal of the digital input DI15
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DO8	Signal of the digital output DO8
4.1	DO9	Signal of the digital output DO9
4.2	DO10	Signal of the digital output DO10
4.3	DO11	Signal of the digital output DO11
4.4	DO12	Signal of the digital output DO12
4.5	DO13	Signal of the digital output DO13
4.6	DO14	Signal of the digital output DO14
4.7	DO15	Signal of the digital output DO15
4.8	UP3	Process voltage UP3 (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Connection of CANopen communication interface module CI582-CN:

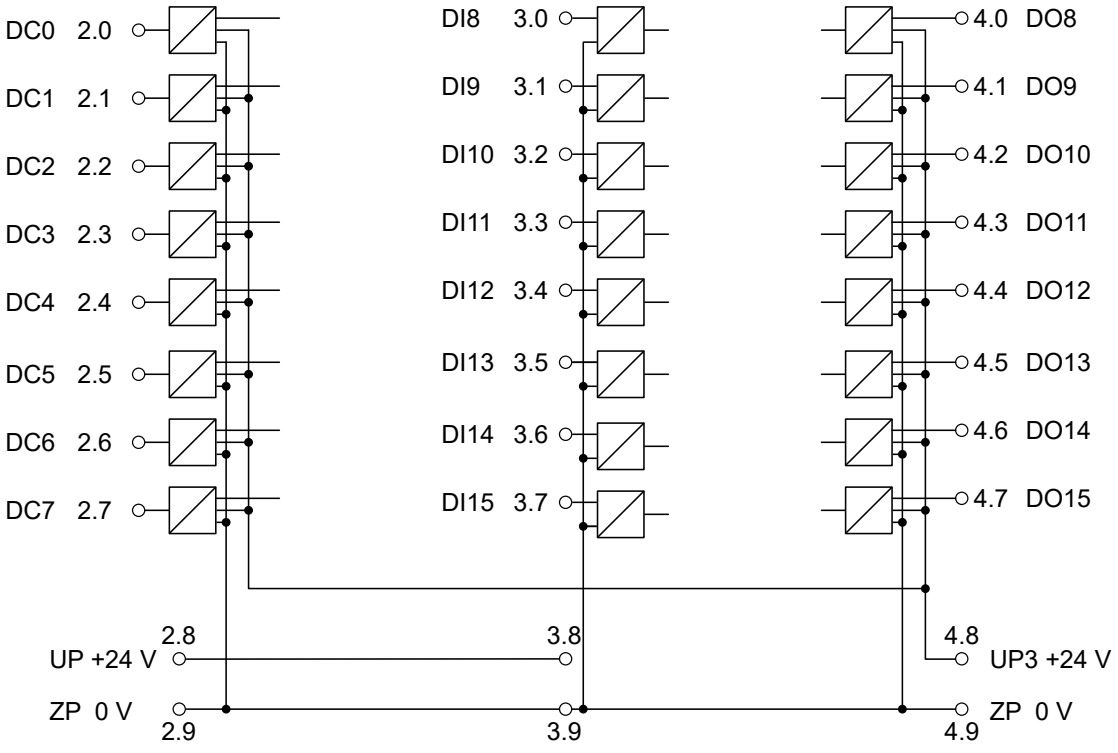


Fig. 214: Connection of the communication interface module CI582-CN

For a description of the meaning of the LEDs, please refer to the section for the state LEDs ↳ *Chapter 1.6.3.7.2.3.9 “State LEDs” on page 3101*.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
50 kbit/s	1000 m

Connection of the digital inputs

The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.

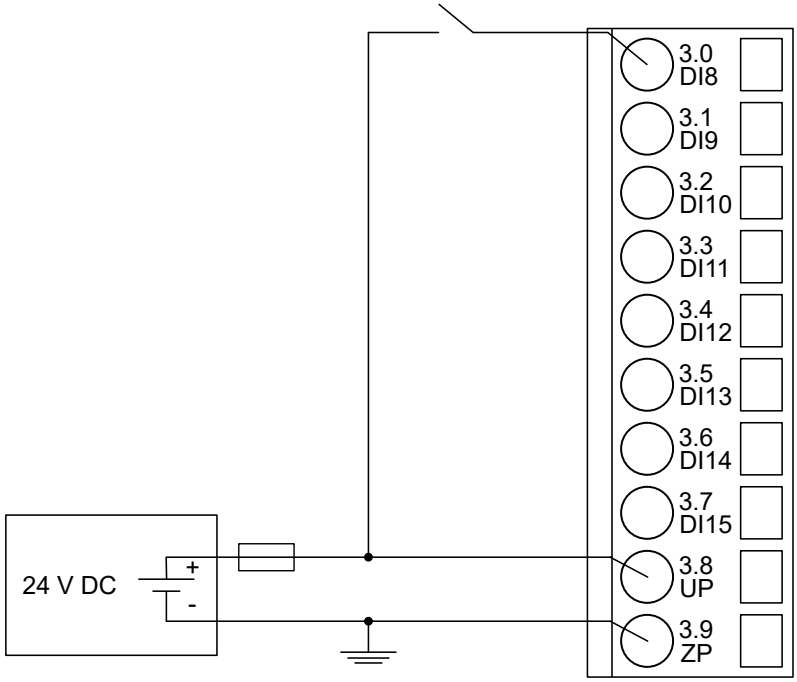


Fig. 215: Connection of the digital inputs to the module CI582-CN

Connection of the digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

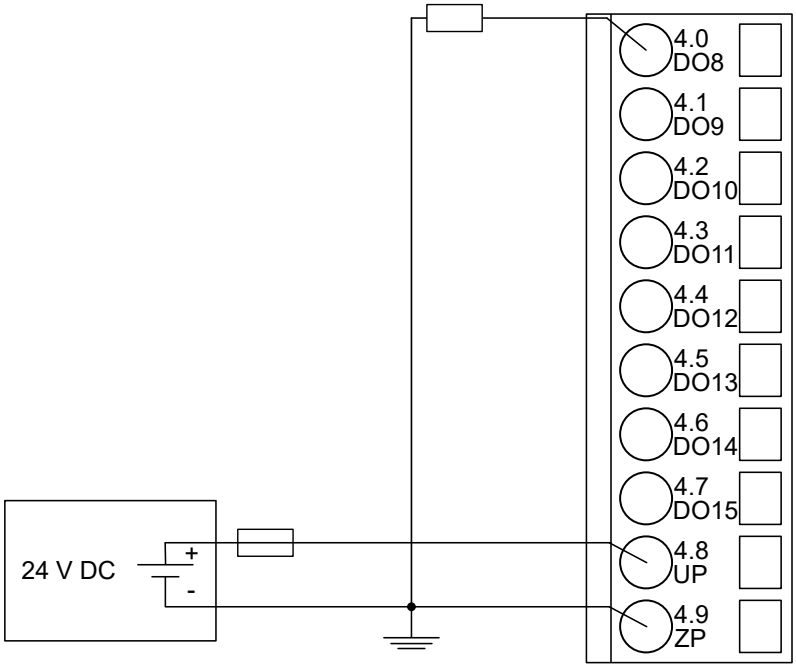


Fig. 216: Connection of configurable digital inputs/outputs to the module CI582-CN

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

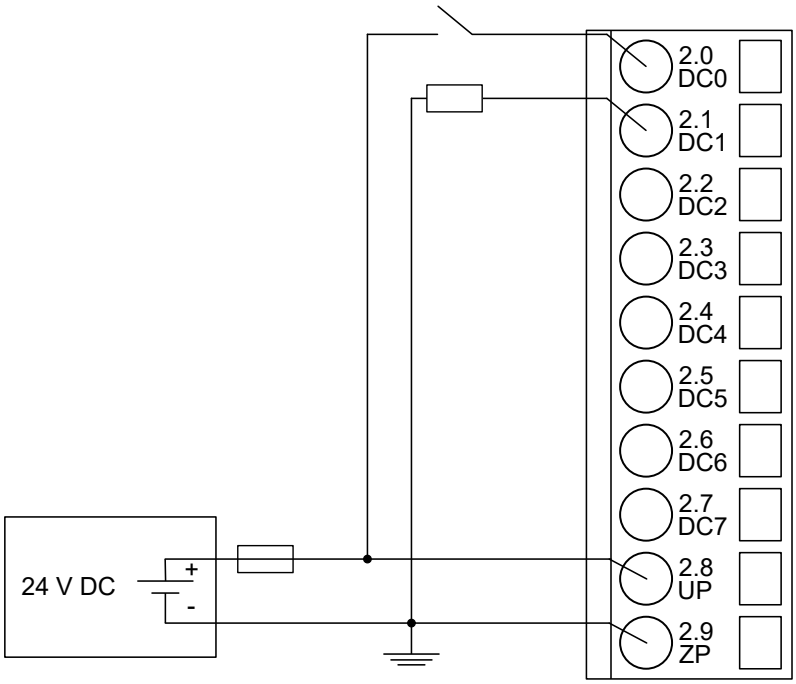


Fig. 217: Connection of configurable digital inputs/outputs to the module CI582-CN

Internal data exchange

Parameter	Value
Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

A detailed description concerning addressing can be found in the documentation of ABB Control Builder Plus Software.

The CANopen communication interface module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

The range of permitted CANopen slave addresses is 1 to 127. Setting a higher address (> 128) does not lead to an error response, but results in a special mode (DS401). In this special mode, the device creates the node address by subtracting the value 128 from the address switch's value.

I/O configuration

The CI582-CN CANopen bus configuration is handled by CANopen master with the exception of the slave node ID (via rotary switches) and the transmision rate (automatic detection).

The digital I/O channels and the fast counter are configured via software.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	0x1C89	WORD	0x1C89
Parameter length	Internal	38	BYTE	38
Error LED / failsafe function table error LED / failsafe function ↳ *Table 550 “Error LED / Failsafe function” on page 3096*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	2		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	18		
Reserved	0	0	ARRAY of 24 BYTES	
Check supply	On	0	BYTE	
	Off	1		1

Name	Value	Internal value	Internal value, type	Default
Fast counter	0	0	BYTE	0
	:	:		
	10 [2])	10		

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.

[2]) For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*.

Table 550: Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode on *)
*) The parameter Behavior DO at comm. error is only analyzed if the failsafe mode is ON.	

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms	0	BYTE	0.1 ms
	1 ms	1		0x00
	8 ms	2		
	32 ms	3		
Detect short circuit at outputs	Off	0	BYTE	On
	On	1		0x01
Behavior DO at comm. error [1])	Off	0	BYTE	Off
	Last value	1		0x00
	Last value 5 sec	6		
	Last value 10 sec	11		
	Substitute value	2		
	Substitute value 5 sec	7		
	Substitute value 10 sec	12		

Name	Value	Internal value	Internal value, type	Default
Substitute value at output	0 ... 65535	0000h ... FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behavior DO at comm. error is applied to DC and DO channels and only analyzed if the failsafe mode is ON.
[2])	The state "externally voltage detected" appears if the output of a channel DC0..DC7 is to be switched on while an external voltage is connected. In this case, start-up is disabled while the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears if external voltage at digital outputs DC0..DC7 and DO0..DO7 has exceeded the process supply voltage UP3 (see 'Connections' ↳ *Chapter 1.6.3.7.2.3.3 “Connections” on page 3087*). The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters to "OFF". This parameter should only be disabled in exceptional cases as voltage overflow may produce reverse voltage.

Diagnosis

Byte Number	Description	Possible Values
1	Diagnosis byte, slot number	31 = CI582-CN (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master

Byte Number	Description	Possible Values
4	Diagnosis byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to Bit 5, coded error description
5	Diagnosis byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus, an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500 display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**CANopen diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket	Replace I/O module / check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**CANopen diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	8...15	46	Externally voltage detected at digital output DO0..DO7 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0..DC7 [6])	Check terminals
4	-	31	4	0...7	47	Short circuit at digital output DC0..DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO0..DO7[7])	Check terminals

Remarks:

[1])	In AC500, the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = position of the communication module;14 = I/O bus; 31 = module itself The identifier is not contained in the CI542-DP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = module itself, 1..10 = expansion module
[3])	With "Module" the following allocation applies depending on the master: Module error: 31 = module itself Channel error: module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears if external voltages at one or more terminals DC0..DC7 or DO0..DO7 cause other digital outputs to be supplied by that voltage (voltage feedback, see 'Connections' ↳ *Chapter 1.6.3.7.2.3.3 “Connections” on page 3087*). All outputs of the digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.

[5])	The voltage at digital outputs DC0..DC7 and DO0..DO7 has exceeded the process supply voltage UP3 (see 'Connections' *Chapter 1.6.3.7.2.3.3 “Connections” on page 3087*). A diagnosis message appears for the whole module.
[6])	This message appears if the output of a channel DC0..DC7 or DO0..DO7 should be switched on while an external voltage is connected. In this case the start-up is disabled while the external voltage is connected. Otherwise, this could produce reverse voltage flowing from this output to other digital outputs. This diagnosis message appears for each channel.
[7])	Short circuit: After a short circuit has been detected, the output is deactivated for 100ms. Subsequently, a new start-up will be executed. This diagnosis message appears for each channel.

State LEDs

The LEDs are located at the front of the module. There are 2 different groups:

- The 5 system LEDs (PWR, CN-RUN, CN-ERR, S-ERR and I/O bus) show the operation states of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O controller	Start-up / preparing communication
	Yellow	---	---	---
CN-RUN	Green	---	Device configured, CANopen bus in OPERATIONAL state and cyclic data exchange running	Flashing: CANopen bus in PRE-OPERATIONAL state and slave is being configured Single flash: CANopen bus in STOPPED state. Flickering: Autodetect is active
CN-ERR	Red	No system error	CANopen Bus is OFF	Flashing: Configuration error Single flash: error counter overflow due to too many error frames Double flash: A node-guard or a heartbeat event occurred Flickering: Autodetect is active

LED	Color	OFF	ON	Flashing
S-ERR	Red	No error	Internal error	--
I/O bus	Green	No decentralized I/O modules connected or communication error	Decentralized I/O modules connected and operational	---

States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/output is OFF	Input/output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 3.0 to 3.7
Reference potential for all inputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)

Parameter	Value
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC
Signal 0	-3 V...+5 V
Undefined signal	> +5 V...< +15 V
Signal 1	+15 V...+30 V
Ripple with signal 0	Within -3 V...+5 V
Ripple with signal 1	Within +15 V...+30 V
Input current per channel	
Input voltage +24 V	Typ. 5 mA
Input voltage +5 V	> 1 mA
Input voltage +15 V	> 2 mA
Input voltage +30 V	< 8 mA
Max. cable length	
Shielded	1000 m
Unshielded	600 m

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DO0 to DO7	Terminals 4.0 to 4.7
Reference potential for all outputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1	UP3 (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	
Rated value per channel	500 mA at UP3 = 24 V
Max. value (all channels together)	4 A
Leakage current with signal 0	< 0.5 mA
Fuse for UP3	10 A fast
Demagnetization with inductive DC load	Via internal varistors (see figure below this table)
Output switching frequency	
With resistive load	On request
With inductive loads	Max. 0.5 Hz

Parameter	Value
With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes (software-controlled supervision)
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

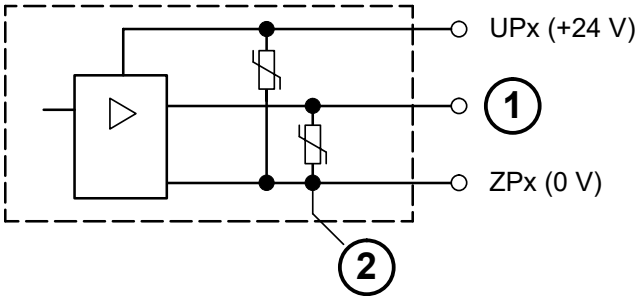


Fig. 218: Digital input/output (circuit diagram)

1	Digital output
2	Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter	Value
Number of channels per module	8 inputs/outputs (with transistors)
Distribution of the channels into groups	1 group for 8 channels
If the channels are used as inputs	
Channels DC0...DC07	Terminals 2.0...2.7
If the channels are used as outputs	
Channels DC0...DC07	Terminals 2.0...2.7
Indication of the input/output signals	1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation	From the CANopen network

Technical data of the digital inputs/outputs if used as inputs

Please refer to the Technical Data of the Digital Inputs *Chapter 1.6.3.7.2.3.10 “Technical data” on page 3102*. Deviation:

Terminals of the channels DC0 to DC7: Terminals 2.0 to 2.7

Due to the direct connection to the output, the demagnetizing varistor is also effective at the input. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Please refer to the Technical Data of the Digital Outputs *Chapter 1.6.3.7.2.3.10 “Technical data” on page 3102*. Deviation:

Terminals of the channels DC0 to DC7: Terminals 2.0 to 2.7

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

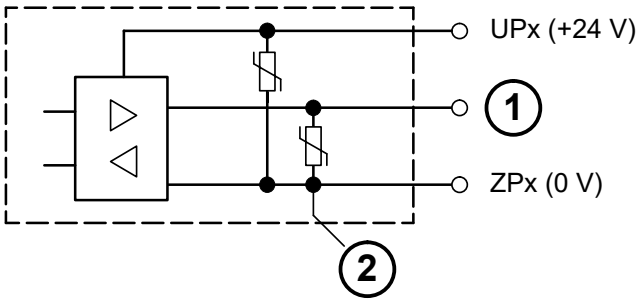


Fig. 219: Digital input/output (circuit diagram)

1	Digital input/output
2	For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 3.0 (DI8), 3.1 (DI9)
Used outputs	Terminal 4.0 (DO8)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	Fast Counter *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*
Operating modes	Operating modes *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 228 200 R0001	CI582-CN, CANopen communication interface module with 8 DI, 8 DO and 8 DC	Active
1SAP 428 200 R0001	CI582-CN-XC, CANopen communication interface module with 8 DI, 8 DO and 8 DC, XC version	Active

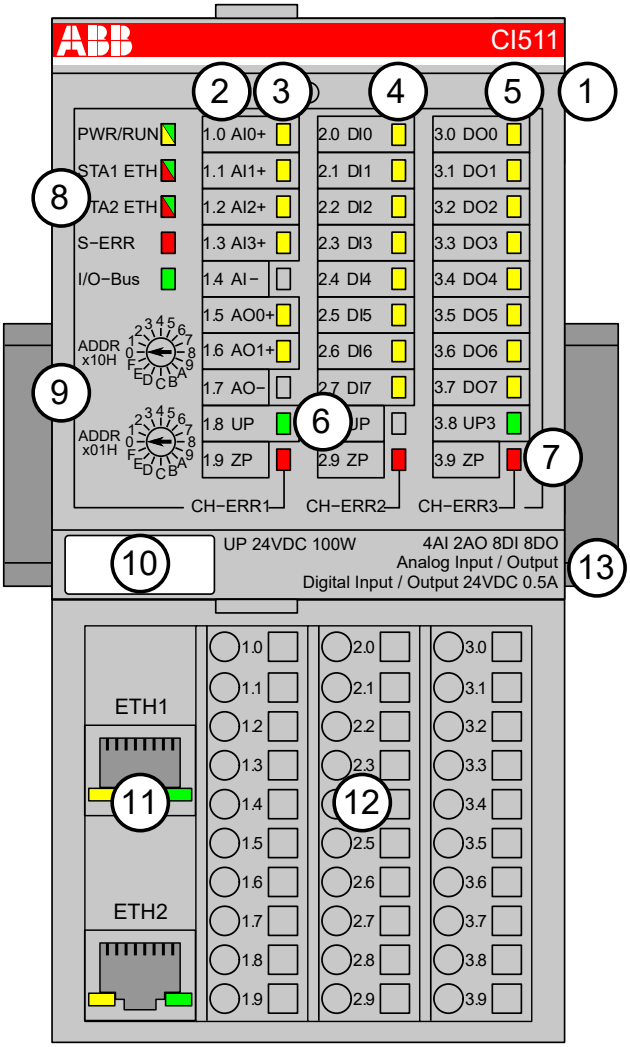
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.7.3 EtherCAT

CI511-ETHCAT

- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- Cam switch functionality (see also Extended Cam Switch Library)
- Extended Cam switch functionality *) (see also Extended Cam Switch Library)
- Module-wise galvanically isolated - Expandability with up to 10 S500 I/O Modules *)

*) Applicable for device index C0 and above.



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, NET, DC, S-ERR, I/O-Bus
9 2 rotary switches (reserved for future extensions)
10 Label
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail

Intended purpose

The EtherCAT communication interface module CI511-ETHCAT is used as decentralized I/O module in EtherCAT networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)

- 8 digital outputs 24 V DC in 1 group (3.0...3.7)
- Cam switch functionality

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	EtherCAT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	Not used; reserved for future extensions
Analog inputs	4 (configurable via software)
Analog outputs	2 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507 or TU508 ↬ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*

Connections

The Ethernet communication interface module CI511-ETHCAT is plugged on the I/O terminal unit TU507-ETH or TU508-ETH. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526).

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

The assignment of the other terminals:

Terminal	Signal	Description
1.0 to 1.3	AI0 to AI3	Positive pole of the 4 analog inputs
1.4	AI-	Negative pole of the analog inputs
1.5 to 1.6	AO0 to AO1	Positive pole of the 2 analog outputs
1.7	AO-	Negative pole of the analog outputs
2.0 to 2.7	DI0 to DI7	8 digital inputs
3.0 to 3.7	DO0 to DO7	8 digital outputs

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

CAUTION!

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

CAUTION!

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Analog signals are always laid in shielded cables. The cable shields are grounded at both ends of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

For simple applications (low disturbances, no high requirement on precision), the shielding can also be omitted.

The following figures show the connection of the Ethernet communication interface module CI511-ETHCAT.

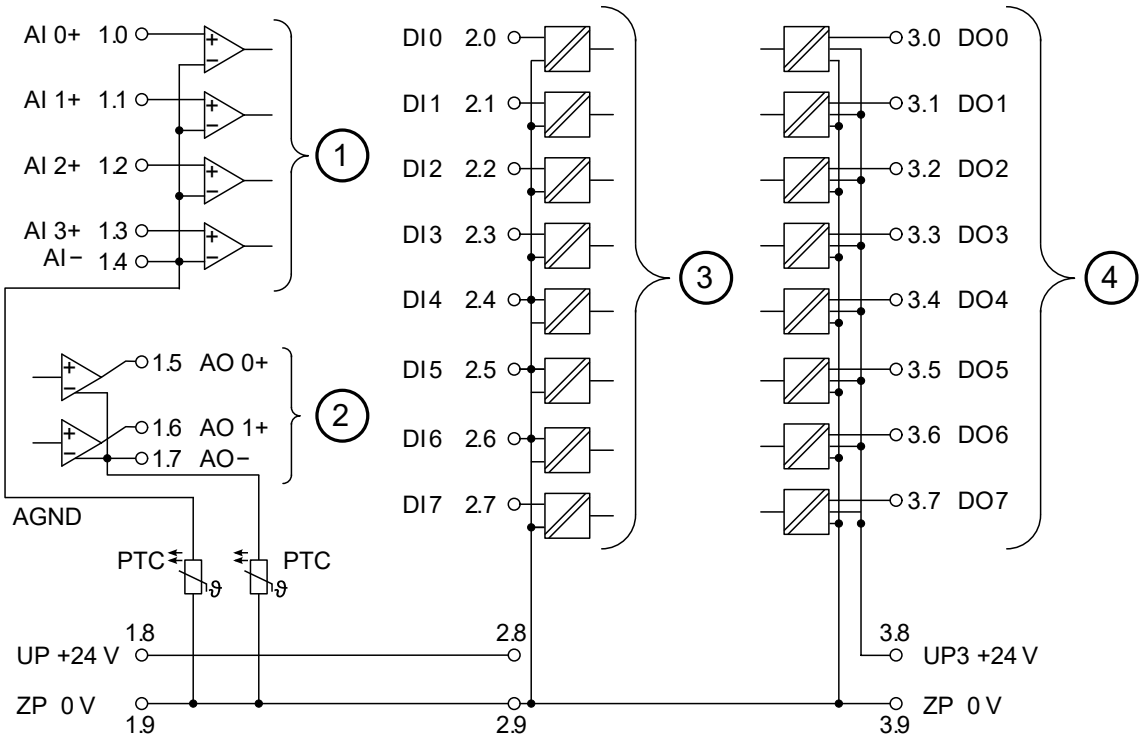


Fig. 220: Connection of the communication interface module CI511-ETHCAT

1 4 analog inputs, configurable for 0...10 V, -10...+10 V, 0/4...20 mA, Pt100/Pt1000, Ni1000 and digital signals
2 2 analog outputs, configurable for -10...+10 V, 0/4...20 mA
3 8 digital inputs 24 V DC
4 8 digital outputs 24 V DC, 0.5 A max.

In case of voltage feedback, 2 cases are distinguished:

1. The outputs are already active

The output group will be switched off. A diagnosis message will appear. After 5 seconds, the module tries automatic reactivation.

2. The outputs are not active

Only the output with voltage feedback will not be set to active. A diagnosis message will appear.

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

CAUTION!

The process supply voltage must be included within the grounding concept of the plant (e. g. grounding of the negative pole).

The module provide several diagnosis functions *Chapter 1.6.3.7.3.1.8 “Diagnosis” on page 3127*.

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

The function of the LEDs is described in the section State LEDs *Chapter 1.6.3.7.3.1.8 “Diagnosis” on page 3127*.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI511-ETHCAT provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration.

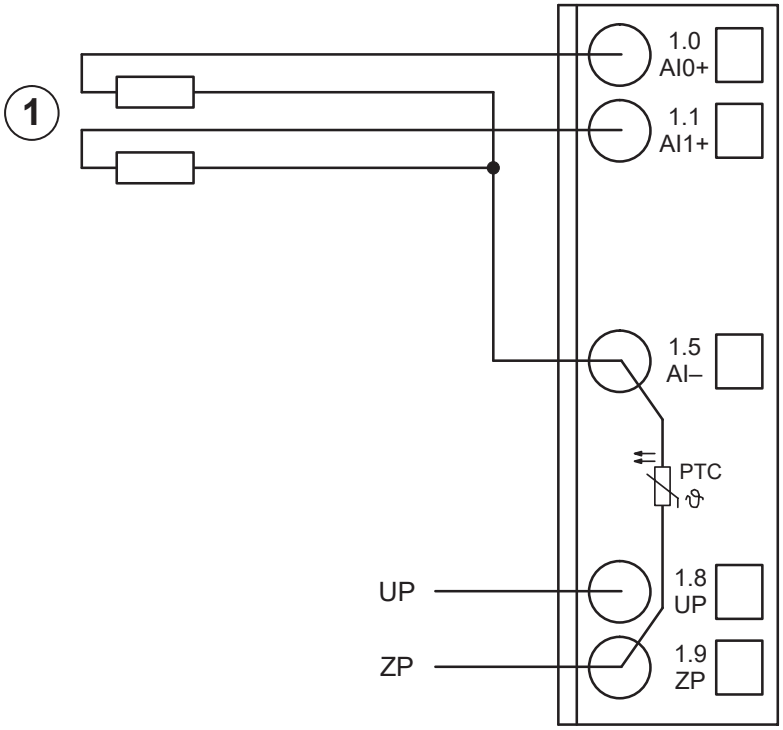


Fig. 221: Connection of resistance thermometers in 2-wire configuration

1 Pt100 (2-wire), Pt1000 (2-wire), Ni1000 (2-wire); 1 analog sensor requires 1 channel

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

The module CI511-ETHCAT performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI511-ETHCAT provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration.

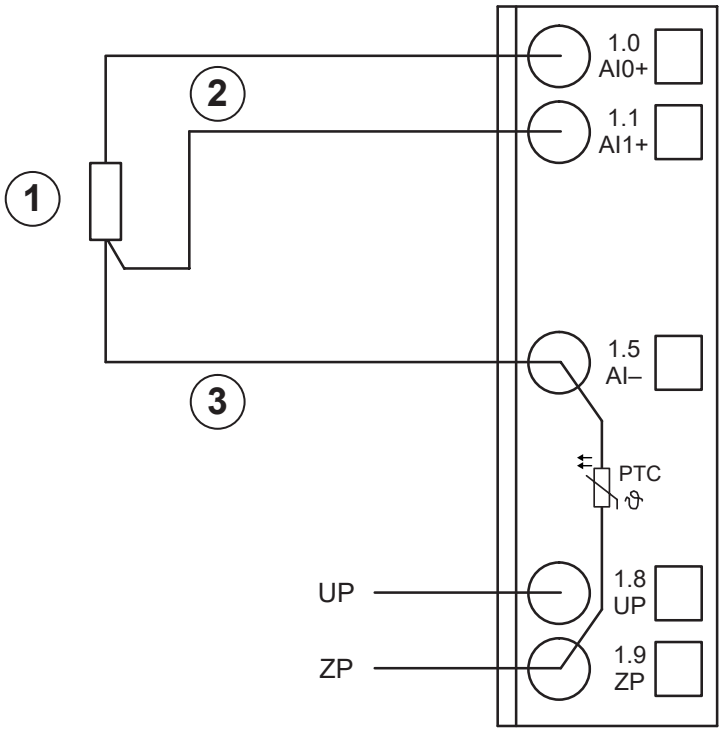


Fig. 222: Connection of resistance thermometers in 3-wire configuration

1 Pt100 (3-wire), Pt1000 (3-wire), Ni1000 (3-wire); 1 analog sensor requires 2 channels
2 Twisted pair within the cable
3 Return line: The return line is only needed once if measuring points are adjacent to each other. This saves wiring costs.

With 3-wire configuration, two adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary, to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

The module CI511-ETHCAT performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply

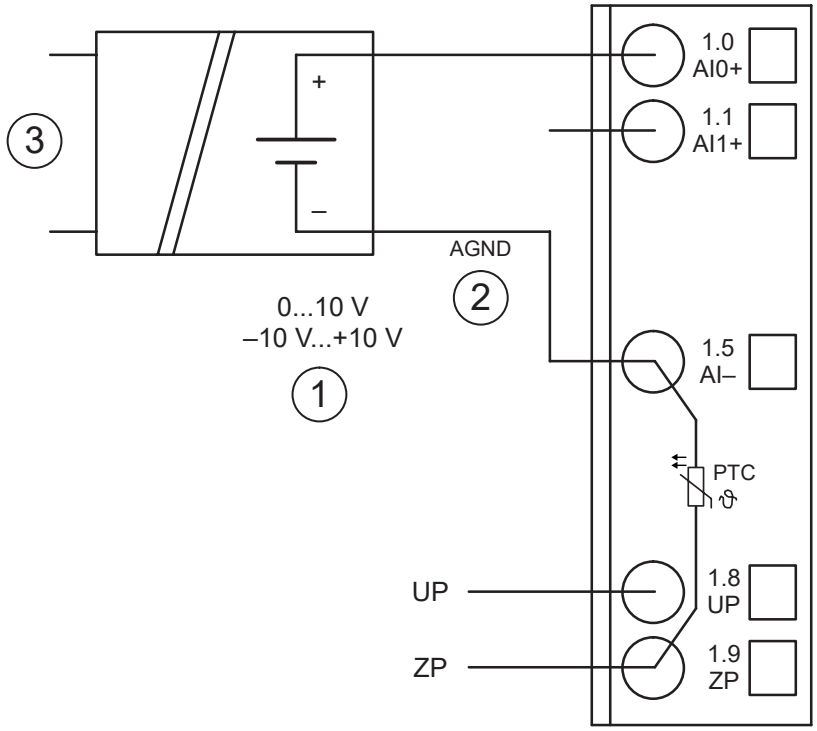


Fig. 223: Connection of active-type analog sensors (voltage) with galvanically isolated power supply

1 1 analog sensor requires 1 channel
2 By connecting to AI-, the galvanically isolated voltage source of the sensor is referred to ZP
3 Galvanically isolated power supply for the analog sensor

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply.

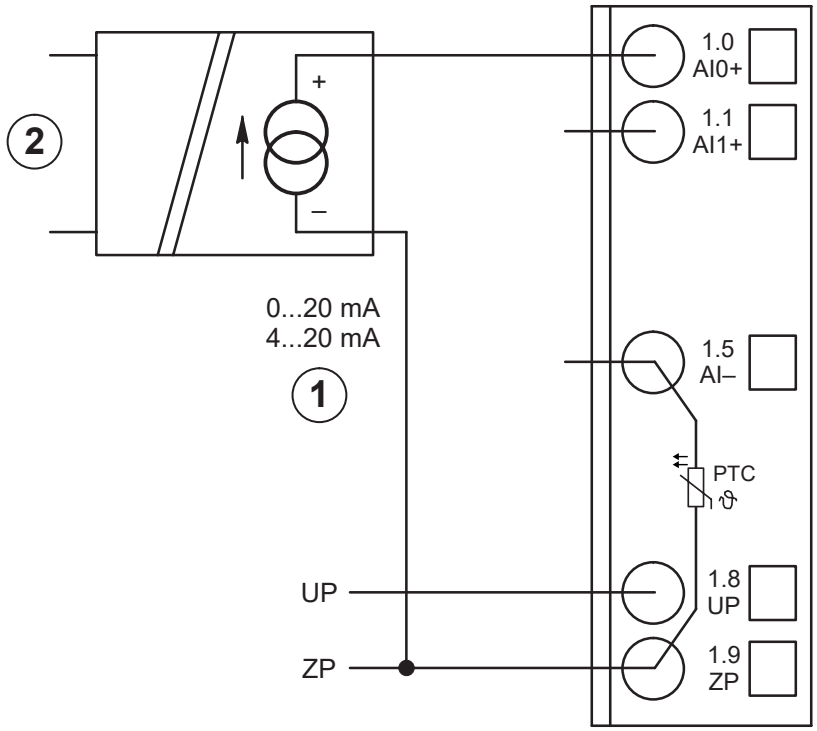


Fig. 224: Connection of active-type analog sensors (current) with galvanically isolated power supply

1 1 analog sensor requires 1 channel
2 Galvanically isolated power supply for the analog sensor

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

The following figure shows the connection of active-type sensors (voltage) with no galvanically isolated power supply.

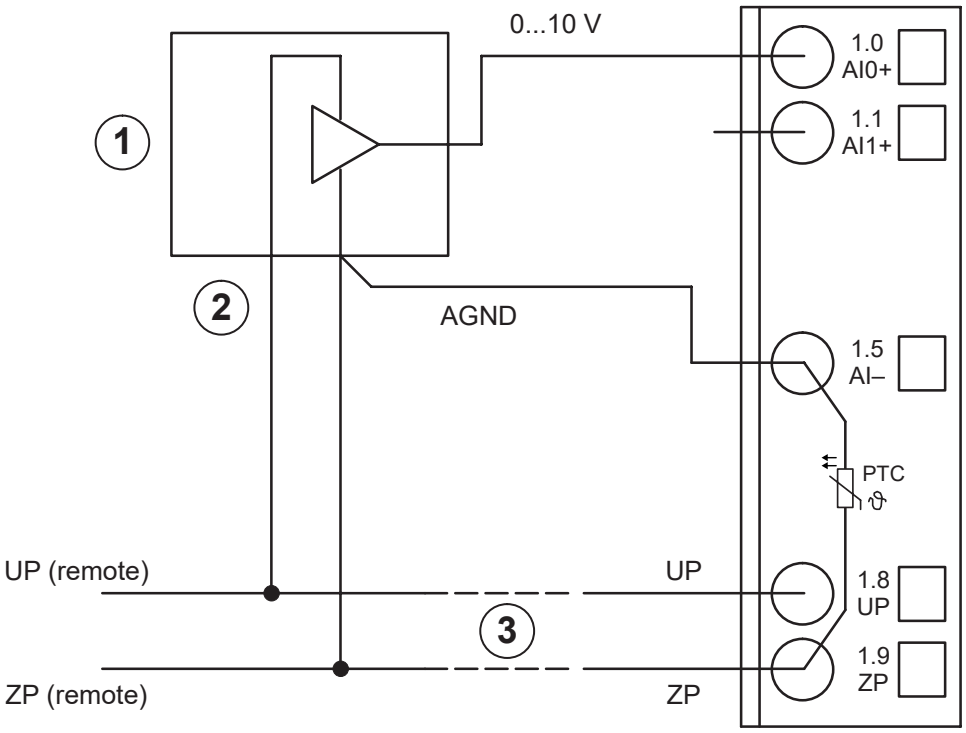


Fig. 225: Connection of active-type sensors (voltage) with no galvanically isolated power supply

1 1 analog sensor requires 1 channel
2 Power supply not galvanically isolated
3 The connection between the negative pole of the sensor and ZP has to be performed
4 Long cable

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used

*) if the sensor can provide this signal range

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

The following figure shows the connection of passive-type analog sensors (current).

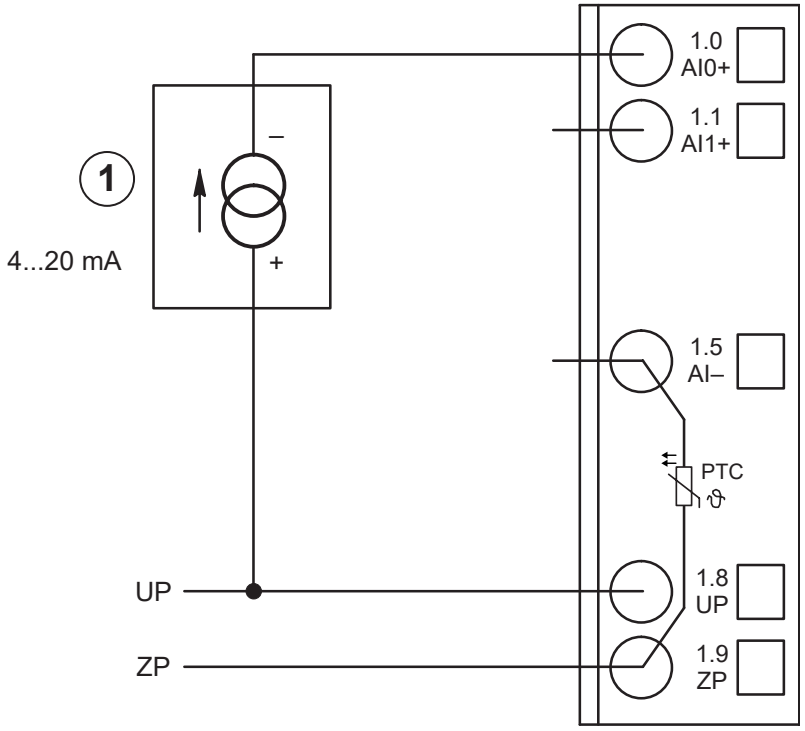


Fig. 226: Connection of passive-type analog sensors (current)

1 1 analog sensor requires 1 channel

Current	4...20 mA	1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second into an analog input, this input is switched off by the module (input protection). In such cases, it is recommended, to protect the analog input by a 10-volt zener diode (in parallel to I+ and I-). But, in general, it is a better solution to prefer sensors with fast initialization or without current peaks higher than 25 mA.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

Important: The ground potential at the sensors must not have a too big potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise problems can occur concerning the common-mode input voltages of the involved analog inputs

The following figure shows the connection of active-type analog sensors (voltage) to differential inputs.

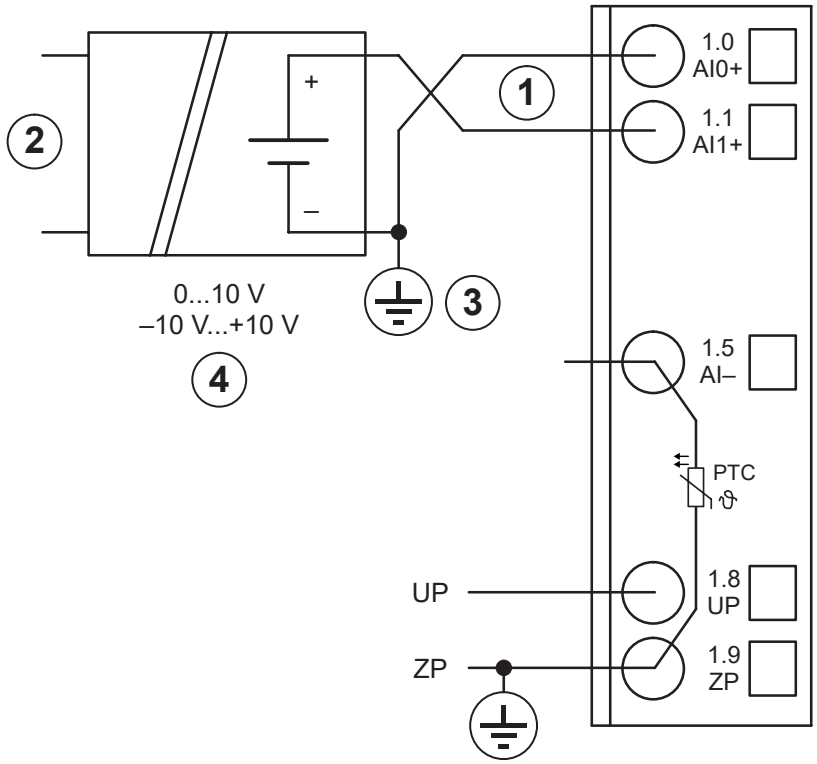


Fig. 227: Connection of active-type analog sensors (voltage) to differential inputs

1 1 analog sensor requires 2 channels
2 Galvanically isolated power supply for the analog sensor
3 Grounding at the sensor
4 0 V...10 V / -10 V...+10 V connected to differential inputs

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital input. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the use of analog inputs as digital inputs.

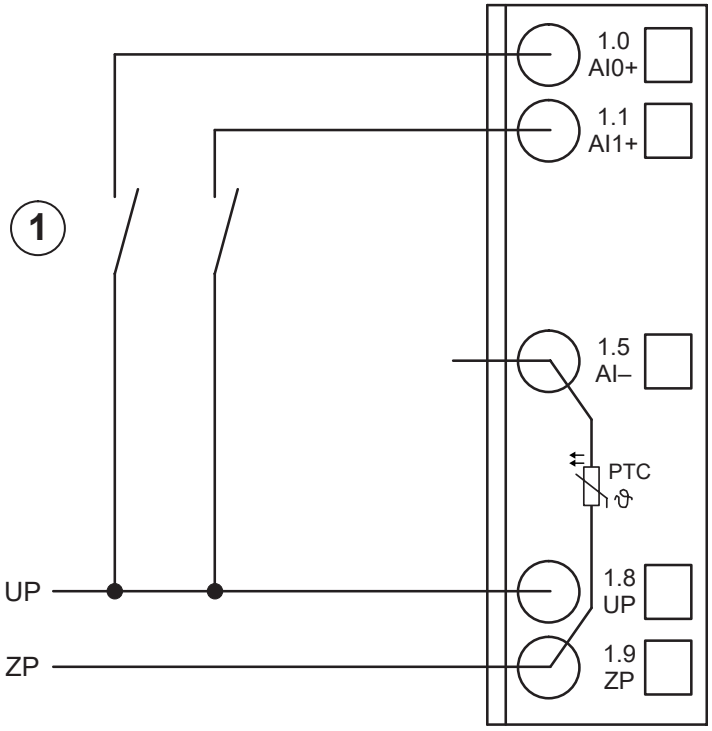


Fig. 228: Use of analog inputs as digital inputs

1 1 digital signal requires 1 channel

Digital input	24 V	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* ↳ *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

Connection of analog output loads (Voltage, current)

The following figure shows the connection of analog output loads (voltage, current).

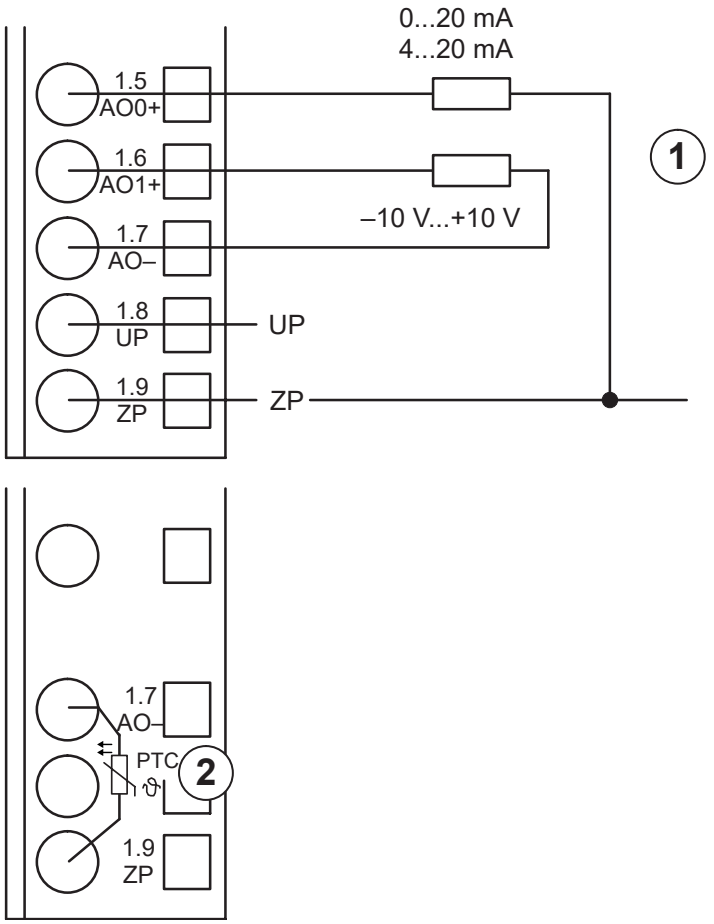


Fig. 229: Connection of analog output loads (voltage, current)

1 1 analog load requires 1 channel

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used
Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

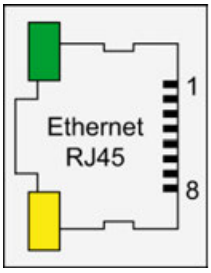
The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.3.7.3.1.7 “Parameterization” on page 3121* ↳ *Chapter 1.6.3.7.3.1.10 “Measuring ranges” on page 3130*.

Unused analog outputs can be left open-circuited.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment. The pin assignment is used for the EtherCAT master (communication module CM5xy-ETHCAT) as well.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.

The EtherCAT network differentiates between input-connectors (IN) and output-connectors (OUT):

At the EtherCAT slaves (communication interface modules), the ETH1-connector is IN and the ETH2-connector is OUT.

At the EtherCAT master (communication module), the ETHCAT1 connector has to be used. The ETHCAT2 connector is reserved for future extensions.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	1
Analog inputs (words)	4
Analog outputs (words)	2

Addressing

The Ethernet bus module CI511-ETHCAT does not consider the position of the rotary switches at the front side of the module. The function of the rotary switches is reserved for future expansions.

I/O configuration

In order to be able to use the CI51X-ETHCAT with device index C0 or above properly, please download the corresponding device description (.xml-)files from http://www.abb.com/plc and install them to the device repository of your Automation Builder. This will allow you to use up to 10 Expandable S500 I/O modules as well as the Extended Cam Switch Library with your CI51X-ETHCAT device.

The CI511-ETHCAT does not store configuration data itself.

The analog I/O channels are configured via software.

Parameterization

Module parameter

Name	Value	Internal value	Internal value, type	Default
Module ID	Internal	48155	WORD	48155
Parameter length	Internal	28	BYTE	28
Error LED / Fail-safe function [1])	On Off by E4 Off by E3 On + failsafe Off by E4 + failsafe Off by E3 + failsafe	0 1 3 16 17 19	BYTE	0
Check Supply	Off On	0 1	BYTE	1

Table 551: Error LED / Failsafe function [1])

Setting	Description
On	Error LED lights up at errors of all error classes, Failsafemode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafemode off
Off by E3	Error LED lights up at errors of error classes E1 and E2 auf, Failsafemode off
On + failsafe	Error LED lights up at errors of all error classes, Failsafemode on *)
Off by E4 + failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafemode on *)
Off by E3 + failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafemode on *)

*) The parameters behaviourAOatCommunicationFault and behaviourDOatCommunicationFault are only analyzed if the Failsafe-mode is ON.

Group parameters of the cam switch

Name	Value	Internal value	Internal value, type	Default
numOfUsed-Cams [1])	0 ... 32 128...160	0 ... 32 218...160	WORD	0
resolution [2])	0 ... 2 -1	0 ... 2 -1	DWORD	36000
zeroShift [3])	0 ... 2 -1	0 ... 2 -1	DWORD	0
EncoderBitReso-lution [4])	8 ... 32	8 ... 32	WORD	18
Reserve	-	-	WORD	-

[1]) The parameter numOfUsedCams defines the interrupt cycle time (Therefore, it takes effect to the accuracy of the track) and the behavior of the module if the DC information is lost.

Parameter setting for numOfUsed-Cams	Number of cams used	Interrupt cycle time	Behavior if DC information is lost
0	0	50 µs	Module changes to "safe-operational" state; the outputs are activated trough the user program
1...8	1...8	80 µs	
9...16	9...16	100 µs	
17...32	17...32	200 µs	
128	0	50 µs	Module keeps in "operational" state; the outputs are activated trough the user program
129...136	1...8	80 µs	Module keeps in "operational" state; the cam switch outputs are activated according to an interpolated timing information
137...144	9...16	100 µs	
145...170	17...32	200 µs	

[2]) The parameter resolution defines the angle resolution of the track. The value gives the number of increments related to 360°; e. g. the value 36,000 corresponds to an angle resolution of 0.01°.

[3]) The parameter zeroShift defines the zero shift. With it the encoder can be adjusted to the mounting position. The value of zeroShift is set in encoder-increments. It is not assigned to the parameter resolution of the cam switch.

[4]) The parameter EncoderBitResolution defines the resolution of the used encoder (in bits), e. g. with the default setting 18 bits the encoder has 196,608 divisions.

Channel parameters for the cam switch (max. 32x)

Name	Value	Internal value	Internal value, type	Default
camToTrack0 *)	Digital Output 0 ... 7, none	0 ... 7, FF	BYTE	FF
:	:	:	:	:
camToTrack31	Digital Output 0 .. .7, none	0 ... 7, FF	BYTE	FF

*) The value of the parameter camToTrack# defines which DO (digital output) is assigned to the track. camToTrack0 = 3 for example means that track 0 is assigned to the digital output 3. If the value FFh is set to a track, no digital output is assigned to it.

Name	Value	Referred FB from extended Cam Switch Library [2])	Internal value	Internal value, type	Default
cam-Type[0] [1]) ...	Common	MCX_CamSwitchSimple_c	0	BYTE	0
	Pulsed	MCX_CamSwitchSimple_dc			
	Timed	MCX_PulseSwitch_dc	1		
	Comfort	MCX_CamSwitchTimed_dc	2		
	Cam shift	MCX_CamSwitchCom-fort_dc	3		
	Binary shift		4		
	Multiturn cam	MCX_CamShift_dc	5		
	Time timed	MCX_BinaryShift_dc	6		
	Reference	MCX_CamSwitchMulti_dc	7		
	Multiturn timed	MCX_SwitchTimeTimed_dc	8		
		MCX_BinaryReference_dc	9		
		MCX_CamSwitchMulti-Timed_dc			

[1]) camType additionally to camToTrack identifies the type of each cam switch and enables the use of a specific function block from the Extended Cam Switch Library.

[2]) camType parameters and the Extended Camswitch Library are only available for CI511-ETHCAT and CI512-ETHCAT with device index C0 and above.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard	0	BYTE	0
Behaviour AO at comm. error *)	Off	0	BYTE	0
	Last value	1		
	Last value 5 s	6		
	Last value 10 s	11		
	Substitute value	2		
	Substitute value 5 s	7		
	Substitute value 10 s	12		

*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, channel configuration	see [1])	see [1])	BYTE	0
Input 0, check channel	see [2])	see [2])	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, channel configuration	see [1])	see [1])	BYTE	0
Input 3, channel configuration	see [2])	see [2])	BYTE	0

Channel configuration [1])

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)

Internal value	Operating modes of the analog inputs, individually configurable
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 552: Channel monitoring [2])

Internal Value	Check channel
0	Plausib(ility), cut wire, short circuit
3	not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, channel configuration	see [3])	see [3])	BYTE	0
Output 0, check channel	see [4])	see [4])	BYTE	0
Output 0, substitute value	see [5])	see [5])	WORD	0
Output 1, channel configuration	see [3])	see [3])	BYTE	0
Output 1, check channel	see [4])	see [4])	BYTE	0
Output 1, substitute value	see [5])	see [5])	WORD	0

Table 553: Channel configuration [3])

Internal value	Operating modes of the analog outputs, individually configurable
0	Not used (default)
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 554: Channel monitoring [4])

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 555: Substitute value [5])

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s	Last value 5 s	0
Last value for 10 s	Last value 10 s	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s	Substitute value 5 s	Depending on configuration
Substitute value for 10 s	Substitute value 10 s	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.01 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.01 ms 0x00
Detect short circuits at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error *)	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute 5 sec Substitute 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 255	00h ... FFh	BYTE	0 0x0000

*) The parameter behaviourDOatCommunicationFault is only analyzed if the Failsafe-mode is ON.

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	20	Slave-to-Slave mal-function		Check configu-ration
3	-	31	31	31	41	Distributed Clock malfunction		Check configu-ration
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage UP
4	-	31	31	31	45	Process voltage UP3 too low		Check process voltage
4	-	31	31	31	34	No response during initialization of the I/O module		Replace I/O module

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])	Check terminals
Channel error digital							
4	-	31	2	0..7	46	Voltage feedback on deactivated digital output [5])	Check terminals
4	-	31	2	0..7	47	Short circuit at digital output	Check terminals
Channel error analog							
4	-	31	1	0..3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0..3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0..3	47	Short circuit at an analog input	Check terminals
4	-	31	3	0..1	48	Analog value overflow at an analog output	Check output value
4	-	31	3	0..1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI511-ETHCAT diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself or ADR = Hardware address (e. g. of the DC551)

[3])	With "Module" the following allocation applies dependent of the master: 31 = Module itself (Module error) or Module type (1=AI, 2=DO, 3=AO; channel error)
[4])	Diagnosis message appears for the whole output group and not per channel. The message occurs if the output channel is already active.
[5])	Diagnosis message appears per channel. The message occurs if the output channel is not active.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, NET, DC, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 556: States of the 5 system LEDs

LED	Color	Off	On	Flashing	1x Flash	2x Flash
PWR/RUN	Green	Error in the internal supply voltage or process voltage missing	Internal supply voltage OK	Module is not configured	--	--
	Yellow	--	--	--	--	--
NET	Green	Init	Operational	Pre-operational	Safe-operational	--
	Red	No error	PDI Watchdog Timeout	Invalid Configuration	Unsolicited State Change	Application time out
DC *)	Green	Distributed Clock not active	Distributed Clock active	--	--	--
	Red	--	--	--	--	--
S-ERR	Red	No error	Internal error	--	--	--
I/O-Bus	Green	No communication interface modules connected or communication error	---	---	--	--
ETH1	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--

LED	Color	Off	On	Flashing	1x Flash	2x Flash
ETH2	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--

*) The state of this LED is only significant if the cam switch functionality is enabled

Table 557: States of the 27 process LEDs

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
			800 : 701	0320 : 02BD
Normal range	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Bus connection	2 x RJ45
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)

Parameter	Value
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability (S500 I/O modules)	Up to 10 S500 I/O modules (Index C0 and above), not available (Index below C0)
Indicators	5 LEDs for state indication
Adjusting elements	2 rotary switches (used for future topology extensions)
Quantity of input/output data	CI512-ETHCAT: 10 bytes input and 14 bytes output CI511-ETHCAT: 18 bytes input and 18 bytes output
Limit of data for input and output	144 byte
Acyclic services	SDO (1500 bytes max.) Emergency ECAT_SLV_DIAG
Protective functions (according to CODESYS)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation to network

Technical data of the module

Parameter		Value
Process supply voltage UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of analog inputs		4
Number of analog outputs		2

Parameter	Value
Reference potential for all digital inputs and outputs	Negative pole of the supply voltage, signal name ZP
Diagnosis	See Diagnosis and Displays *Chapter 1.6.3.7.3.1.8 “Diagnosis” on page 3127*
Operation and error displays	32 LEDs (totally)
Weight (without terminal unit)	ca. 125 g
Mounting position	Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA

Parameter		Value
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

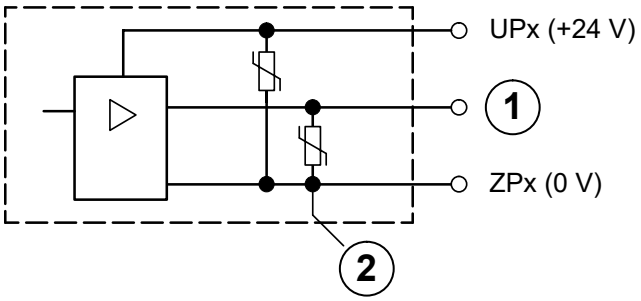


Fig. 230: Digital input/output (circuit diagram)

1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for AI0+ to AI3+		Terminal 1.4 (AI-) for voltage and RTD measurement Terminals 1.9, 2.9 and 3.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Galvanic isolation		Against Ethernet network
Configurability		0 V...10 V, -10 V...+10 V, 0/4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %

Parameter	Value
Relationship between input signal and hex code	Tables Input Ranges Voltage, Current and Digital Input ↳ *Chapter 1.6.3.7.3.1.10.1 "Input ranges voltage, current and digital input" on page 3130* and Input range resistance temperature detector ↳ *Chapter 1.6.3.7.3.1.10.2 "Input ranges resistance temperature detector" on page 3131*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for the inputs		Terminals 1.9, 2.9 and 3.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 1.5...1.6
Reference potential for AO0+ to AO1+		Terminal 1.7 (AO-) for voltage outputTerminals 1.9, 2.9 and 3.9 (ZP) for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against Ethernet network
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)

Parameter	Value
Output resistance (load), as current output	0 ... 500 Ω
Output loadability, as voltage output	± 10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Table Output Ranges Voltage and Current ↯ *Chapter 1.6.3.7.3.1.10.3 “Output ranges voltage and current” on page 3132*
Unused outputs	Are configured as unused (default value) and can be left open-circuited

Ordering data

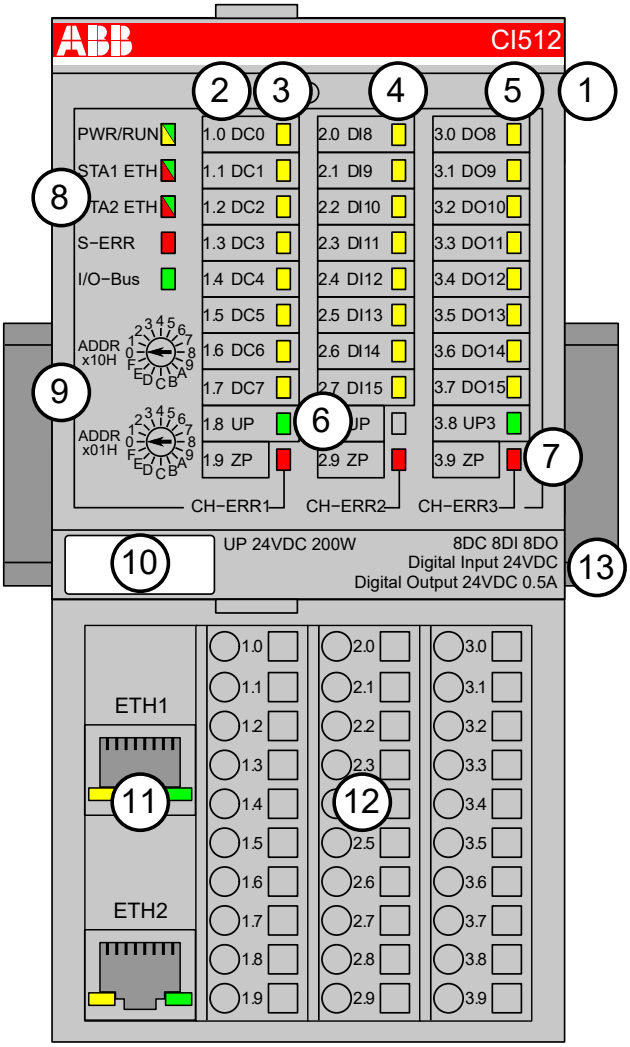
Part no.	Description	Product life cycle phase *)
1SAP 220 900 R0001	CI511-ETHCAT, EtherCAT communication interface module, 8 DI, 8 DO, 4 AI and 2 AO	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI512-ETHCAT

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Cam switch functionality (see also Extended Cam Switch Library)
- Extended Cam switch functionality *)
  (see also Extended Cam Switch Library)
- Module-wise galvanically isolated
- Expandability with up to 10 S500 I/O modules *)

*) Applicable for device index C0 and above.



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital configurable inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 System LEDs: PWR/RUN, NET, DC, S-ERR, I/O-Bus
9 2 rotary switches (reserved for future extensions)
10 Label
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail

Intended purpose

The EtherCAT communication interface module CI512-ETHCAT is used as decentralized I/O module in EtherCAT networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)
- Cam switch functionality

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the configurable digital inputs/outputs is performed by software.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	EtherCAT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	Not used; reserved for future extensions
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507 or TU508 *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*

Connections

The Ethernet communication interface module CI512-ETHCAT is plugged on the I/O terminal unit TU507-ETH or TU508-ETH. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter Chapter 1.6.4.5 “AC500-eCo” on page 3352.

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	DC0 to DC7	8 digital inputs/outputs (configurable via software)
2.0 to 2.7	DI0 to DI7	8 digital inputs (delay time configurable via software)
3.0 to 3.7	DO0 to DO7	8 digital outputs

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figures show the connection of the Ethernet communication interface module CI512-ETHCAT.

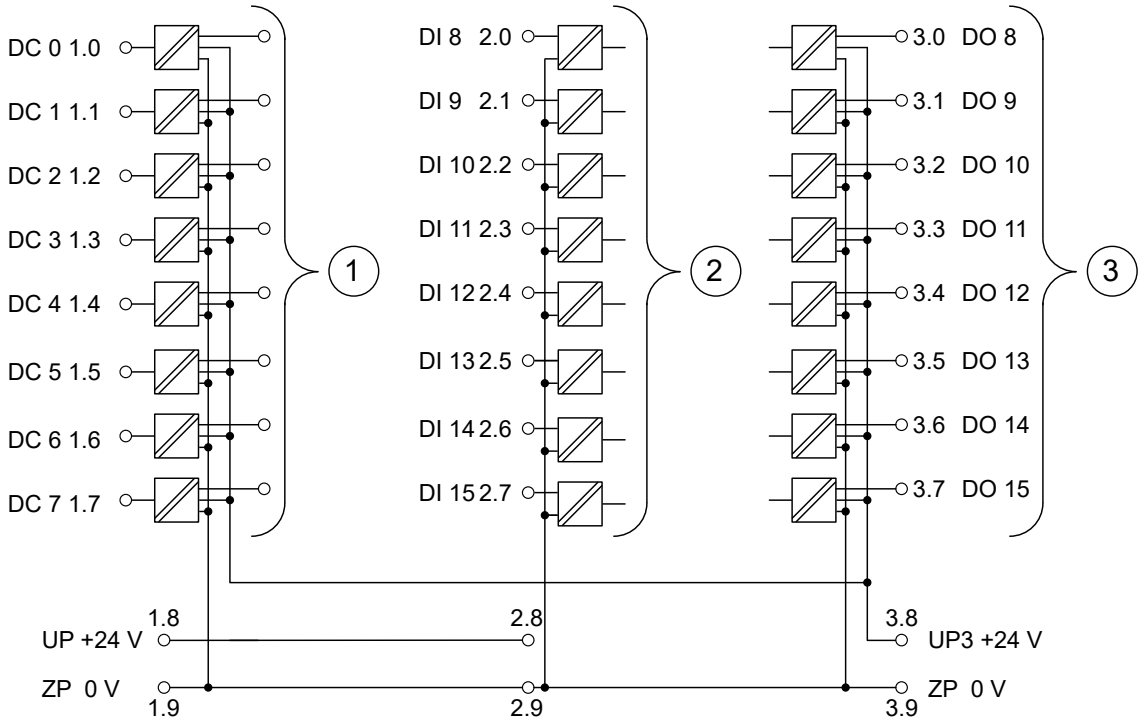


Fig. 231: Connection of the communication interface module CI512-ETHCAT

1 8 digital configurable inputs/outputs 24 V DC
2 8 digital inputs 24 V DC
3 8 digital outputs 24 V DC

In case of voltage feedback, 2 cases are distinguished:

1. The outputs are already active

The output group will be switched off. A diagnosis message will appear. After 5 seconds, the module tries automatic reactivation.

2. The outputs are not active

Only the output with voltage feedback will not be set to active. A diagnosis message will appear.

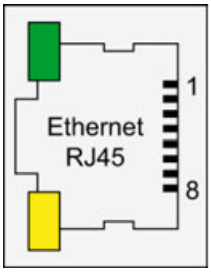
CAUTION!
The process supply voltage must be included within the grounding concept of the plant (e. g. grounding of the negative pole).

The module provides several diagnosis functions *Chapter 1.6.3.7.3.2.9 “Diagnosis” on page 3147*.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment. The pin assignment is used for the EtherCAT master (communication module CM5xy-ETHCAT) as well.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.

The EtherCAT network differentiates between input-connectors (IN) and output-connectors (OUT):

At the EtherCAT slaves (communication interface modules), the ETH1-connector is IN and the ETH2-connector is OUT.

At the EtherCAT master (communication module), the ETHCAT1 connector has to be used. The ETHCAT2 connector is reserved for future extensions.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	1
Configurable digital inputs/outputs (bytes)	1 + 1

Addressing

The Ethernet communication interface module CI512-ETHCAT does not consider the position of the rotary switches at the front side of the module. The function of the rotary switches is reserved for future expansions.

I/O configuration

> *In order to be able to use the CI51X-ETHCAT with device index C0 or above properly, please download the corresponding device description (.xml-)files from http://www.abb.com/plc and install them to the device repository of your Automation Builder. This will allow you to use up to 10 Expandable S500 I/O modules as well as the Extended Cam Switch Library with your CI51X-ETHCAT device.*

The CI512-ETHCAT does not store configuration data itself.

The analog I/O channels are configured via software.

Parameterization

Module parameter

Name	Value	Internal value	Internal value, type	Default
Module ID	Internal	49435	WORD	49435
Parameter length	Internal	10	BYTE	10
Error LED / Fail-safe function [1])	On Off by E4 Off by E3 On + failsafe Off by E4 + failsafe Off by E3 + failsafe	0 1 3 16 17 19	BYTE	0
Check Supply	Off On	0 1	BYTE	1

Table 558: Error LED / Failsafe function [1])

Setting	Description
On	Error LED lights up at errors of all error classes, Failsafe mode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
Off by E3	Error LED lights up at errors of error classes E1 and E2 auf, Failsafe mode off
On + failsafe	Error LED lights up at errors of all error classes, Failsafe mode on *)
Off by E4 + failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
Off by E3 + failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

*) The parameter behaviourDOatCommunicationFault is only analyzed if the Failsafe-mode is ON.

Group parameters of the cam switch

Name	Value	Internal value	Internal value, type	Default
numOfUsed-Cams [1])	0 ... 32 128...160	0 ... 32 218...160	WORD	0
resolution [2])	0 ... 2 -1	0 ... 2 -1	DWORD	36000
zeroShift [3])	0 ... 2 -1	0 ... 2 -1	DWORD	0
EncoderBitReso-lution [4])	8 ... 32	8 ... 32	WORD	18
Reserve	-	-	WORD	-

Remarks:

[1]) The parameter numOfUsedCams defines the interrupt cycle time (Therefore, it takes effect to the accuracy of the track) and the behavior of the module if the DC information is lost.

Parameter setting for numOfUsed-Cams	Number of cams used	Interrupt cycle time	Behavior if DC information is lost
0	0	50 µs	Module changes to "safe-operational" state; the outputs are activated trough the user program
1...8	1...8	80 µs	
9...16	9...16	100 µs	
17...32	17...32	200 µs	
128	0	50 µs	Module keeps in "operational" state; the outputs are activated trough the user program
129...136	1...8	80 µs	Module keeps in "operational" state; the cam switch outputs are activated according to an interpolated timing information
137...144	9...16	100 µs	
145...170	17...32	200 µs	

[2]) The parameter resolution defines the angle resolution of the track. The value gives the number of increments related to 360°; e. g. the value 36,000 corresponds to an angle resolution of 0.01°.

[3]) The parameter zeroShift defines the zero shift. With it the encoder can be adjusted to the mounting position. The value of zeroShift is set in encoder-increments. It is not assigned to the parameter resolution of the cam switch.

[4]) The parameter EncoderBitResolution defines the resolution of the used encoder (in bits), e. g. with the default setting 18 bits the encoder has 196,608 divisions.

Channel parameters for the cam switch (max. 32x)

Name	Value	Internal value	Internal value, type	Default
camToTrack0 [1])	Digital Output 0 ... 15, none	0 ... 15, FF	BYTE	FF
:	:	:	:	:
camToTrack31	Digital Output 0 ... 15, none	0 ... 15, FF	BYTE	FF

[1]) The value of the parameter camToTrack# defines which DO (digital output) is assigned to the track. camToTrack0 = 3 for example means that track 0 is assigned to the digital output 3. If the value FFh is set to a track, no digital output is assigned to it.

Name	Value	Referred FB from extended Cam Switch Library [2])	Internal value	Internal value, type	Default
cam-Type[0] [1]) ...	Common	MCX_CamSwitchSimple_c	0	BYTE	0
	Pulsed	MCX_CamSwitchSimple_dc			
	Timed	MCX_PulseSwitch_dc	1		
	Comfort	MCX_CamSwitchTimed_dc	2		
	Cam shift	MCX_CamSwitchComfort_dc	3		
	Binary shift	MCX_CamShift_dc	4		
	Multiturn cam	MCX_BinaryShift_dc	5		
	Time timed	MCX_CamSwitchMulti_dc	6		
	Reference	MCX_SwitchTimeTimed_dc	7		
	Multiturn timed	MCX_BinaryReference_dc	8		
		MCX_CamSwitchMulti-Timed_dc	9		

[1]) camType additionally to camToTrack identifies the type of each cam switch and enables the use of a specific function block from the Extended Cam Switch Library.

[2]) camType parameters and the Extended Camswitch Library are only available for CI511-ETHCAT and CI512-ETHCAT with device index C0 and above.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.01 ms	0	BYTE	0.01 ms
	1 ms	1		0x00
	8 ms	2		
	32 ms	3		
Detect short circuit at outputs	Off	0	BYTE	On
	On	1		0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour DO at comm. error *)	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute values DO	0 ... 65535	0000h ... FFFFh	WORD	0 0x0000
*) The parameter behaviourDOatCommunicationFault is only analyzed if the Failsafe-mode is ON.				

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000..063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module error								
3	-	31	31	31	43	Internal error in the module		Replace I/O module
3	-	31	31	31	20	Slave-to-Slave malfunction		Check configuration
3	-	31	31	31	41	Distributed Clock malfunction		Check configuration
3	-	31	31	31	26	Parameter error		Check master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000..063**	**AC500-Display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage
4	-	31	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])	Check terminals
Channel error digital							
4	-	31	2	0..15	46	Voltage feedback on deactivated digital output [5])	Check terminals
4	-	31	4	0..7	47	Short circuit at digital output	Check terminals
4	-	31	2	8..15	47	Short circuit at digital output	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module; 14 = I/O bus; 31 = Module itself The identifier is not contained in the CI512-ETHCAT diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself or ADR = Hardware address (e. g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: 31 = Module itself (Module error) or Module type (1=AI, 2=DO, 3=AO; channel error)
[4])	Diagnosis message appears for the whole output group and not per channel. The message occurs if the output channel is already active.
[5]	Diagnosis message appears per channel. The message occurs if the output channel is not active.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, NET, DC, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 559: States of the 5 system LEDs

LED	Color	Off	On	Flashing	1x flash	2x flash
PWR/RUN	Green	Error in the internal supply voltage or process voltage missing	Internal supply voltage OK	Module is not config-ured	--	--
	Yellow	--	--	--	--	--
NET	Green	Init	Operational	Pre-opera-tional	Safe-opera-tional	--
	Red	No error	PDI Watchdog Timeout	Invalid Con-figuration	Unsolicited State Change	Application time out
DC *)	Green	Distributed Clock not active	Distributed Clock active	--	--	--
	Red	--	--	--	--	--
S-ERR	Red	No error	Internal error	--	--	--
I/O-Bus	Green	No commu-nication interface modules connected or commu-nication error	---	---	--	--
ETH1	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--
ETH2	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--
*) The state of this LED is only significant if the camswitch functionality is enabled						

Table 560: States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Bus connection	2 x RJ45
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability (S500 I/O modules)	Up to 10 S500 I/O modules (Index C0 and above), not available (Index below C0)
Indicators	5 LEDs for state indication
Adjusting elements	2 rotary switches (used for future topology extensions)
Quantity of input/output data	CI512-ETHCAT: 10 bytes input and 14 bytes output CI511-ETHCAT: 18 bytes input and 18 bytes output
Limit of data for input and output	144 byte

Parameter	Value
Acyclic services	SDO (1500 bytes max.) Emergency ECAT_SLV_DIAG
Protective functions (according to CODESYS)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation to network

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of configurable digital inputs/outputs		8
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Diagnosis		See Diagnosis and Displays *Chapter 1.6.3.7.3.2.9 “Diagnosis” on page 3147*
Operation and error displays		34 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

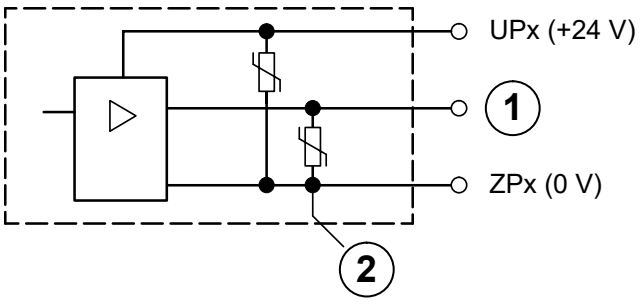


Fig. 232: Digital input/output (circuit diagram)

1 Digital Output
2 Varistors for demagnetization when inductive loads are turned off

Figure:

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC07	Terminals 1.0...1.7
If the channels are used as outputs		
	Channels DC0...DC07	Terminals 1.0...1.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the Ethernet network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V *)
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

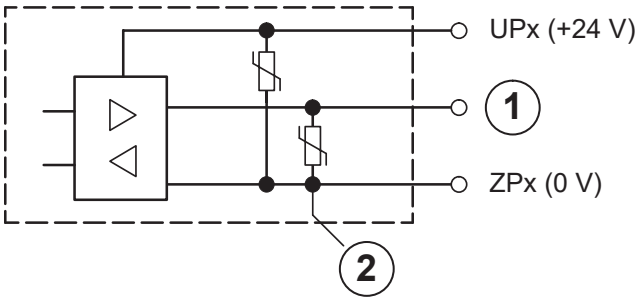


Fig. 233: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Ordering data

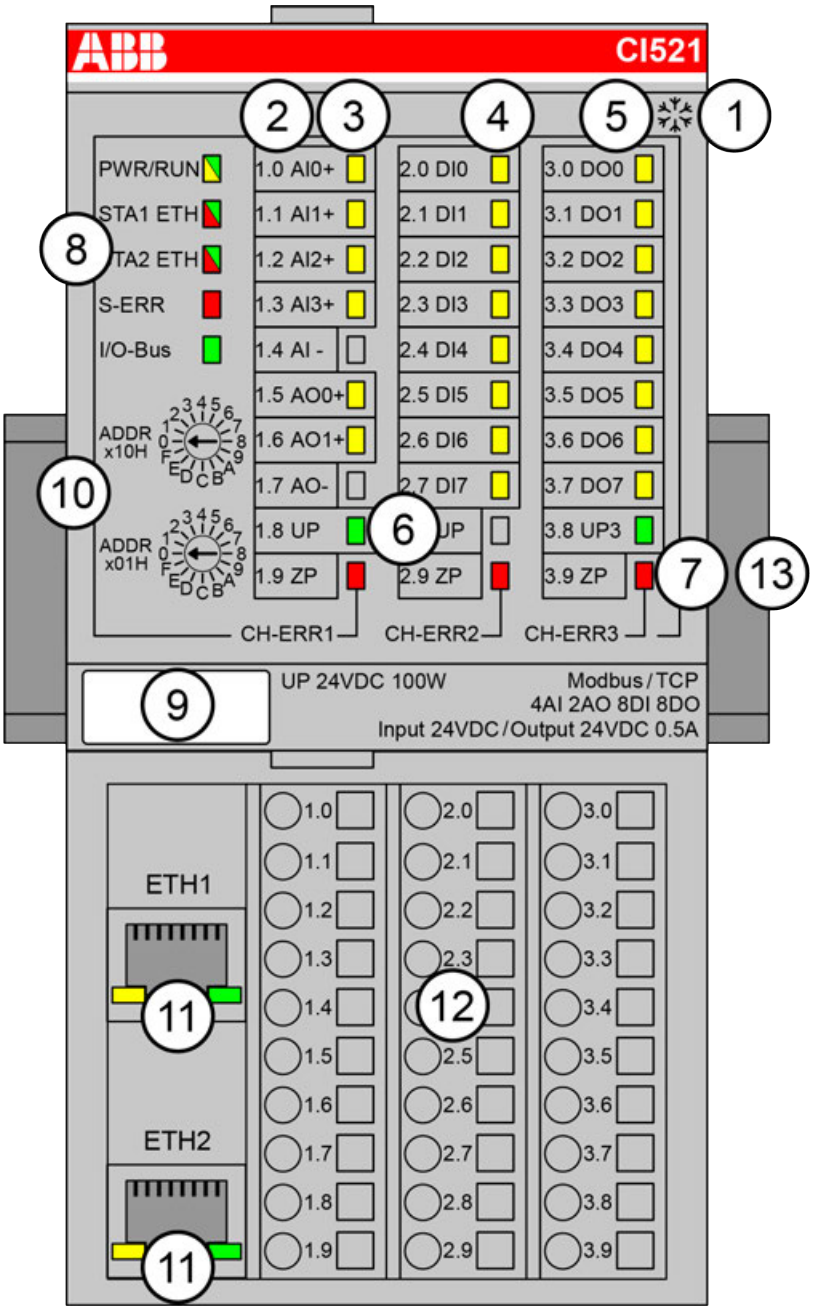
Part no.	Description	Product life cycle phase *)
1SAP 221 000 R0001	CI512-ETHCAT, EtherCAT communication interface module, 8 DI, 8 DO and 8 DC	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.7.4 Modbus

CI521-MODTCP

- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the IP address
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The Modbus TCP communication interface module CI521-MODTCP is used as decentralized I/O module in Modbus TCP networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	Modbus TCP
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	for setting the last BYTE of the IP (00h to FFh)
Analog inputs	4 (configurable via software)
Analog outputs	2 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Required terminal unit	TU507 or TU508 ↬ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*

Connections

The Ethernet communication interface module CI521-MODTCP is plugged on the I/O terminal unit TU507-ETH or TU508-ETH ↬ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Conditions for undisturbed operating with older I/O expansion modules

All I/O expansion modules that are attached to the CI52x-MODTCP must be powered up together with the CI52x-MODTCP if the firmware version of these I/O expansion modules is V1.9 or lower.

The firmware version is related to the index. The index is printed on the module type label on the right side.

Modules as of index listed in the following table can be powered up independently.

S500 I/O module type	First index with firmware version above 1.9
AI523	D0
AI523-XC	D0
AI531	A3
AI531-XC	A0
AO523	D0
AO523-XC	D0
AX521	D0
AX521-XC	D0
AX522	D0
AX522-XC	D0
CD522	A2
CD522-XC	A0
DA501	A2
DA501-XC	A0
DA502	A1
DA502-XC	A1
DC522	D0
DC522-XC	D0
DC523	D0

S500 I/O module type	First index with firmware version above 1.9
DC523-XC	D0
DC532	D0
DC532-XC	D0
DI524	D0
DI524-XC	D0
DO524	A2
DO524-XC	A2
DX522	D0
DX522-XC	D0
DX531	D0
AC522	D0
PD501	D0

Do not connect any voltages externally to digital outputs!

Reason: Externally voltages at an output or several outputs may cause that other outputs are supplied through that voltage instead of voltage UP3 (reverse voltage). This ist not intended usage.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO0..DO7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	AI0+	Positive pole of analog input signal 0
1.1	AI1+	Positive pole of analog input signal 1
1.2	AI2+	Positive pole of analog input signal 2
1.3	AI3+	Positive pole of analog input signal 3
1.4	AI-	Negative pole of analog input signals 0 to 3
1.5	AO0+	Positive pole of analog output signal 0
1.6	AO1+	Positive pole of analog output signal 1
1.7	AI-	Negative pole of analog output signals 0 and 1
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI0	Signal of the digital input DI0
2.1	DI1	Signal of the digital input DI1
2.2	DI2	Signal of the digital input DI2
2.3	DI3	Signal of the digital input DI3
2.4	DI4	Signal of the digital input DI4

Terminal	Signal	Description
2.5	DI5	Signal of the digital input DI5
2.6	DI6	Signal of the digital input DI6
2.7	DI7	Signal of the digital input DI7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO0	Signal of the digital output DO0
3.1	DO1	Signal of the digital output DO1
3.2	DO2	Signal of the digital output DO2
3.3	DO3	Signal of the digital output DO3
3.4	DO4	Signal of the digital output DO4
3.5	DO5	Signal of the digital output DO5
3.6	DO6	Signal of the digital output DO6
3.7	DO7	Signal of the digital output DO7
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figures show the connection of the Ethernet communication interface module CI521-MODTCP.

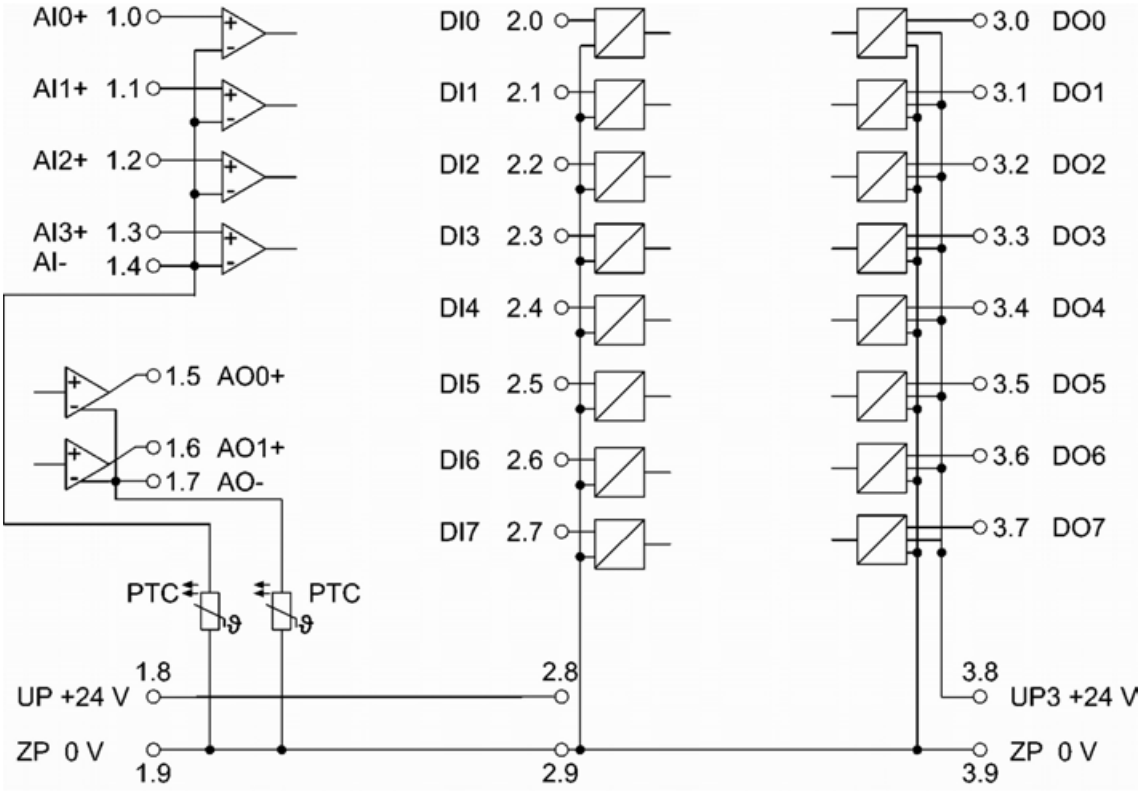


Fig. 234: Connection of the communication interface module CI521-MODTCP

Further information is provided in the System Technology chapter *Chapter 1.6.5.3.1 “Modbus communication interface module” on page 3603*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.

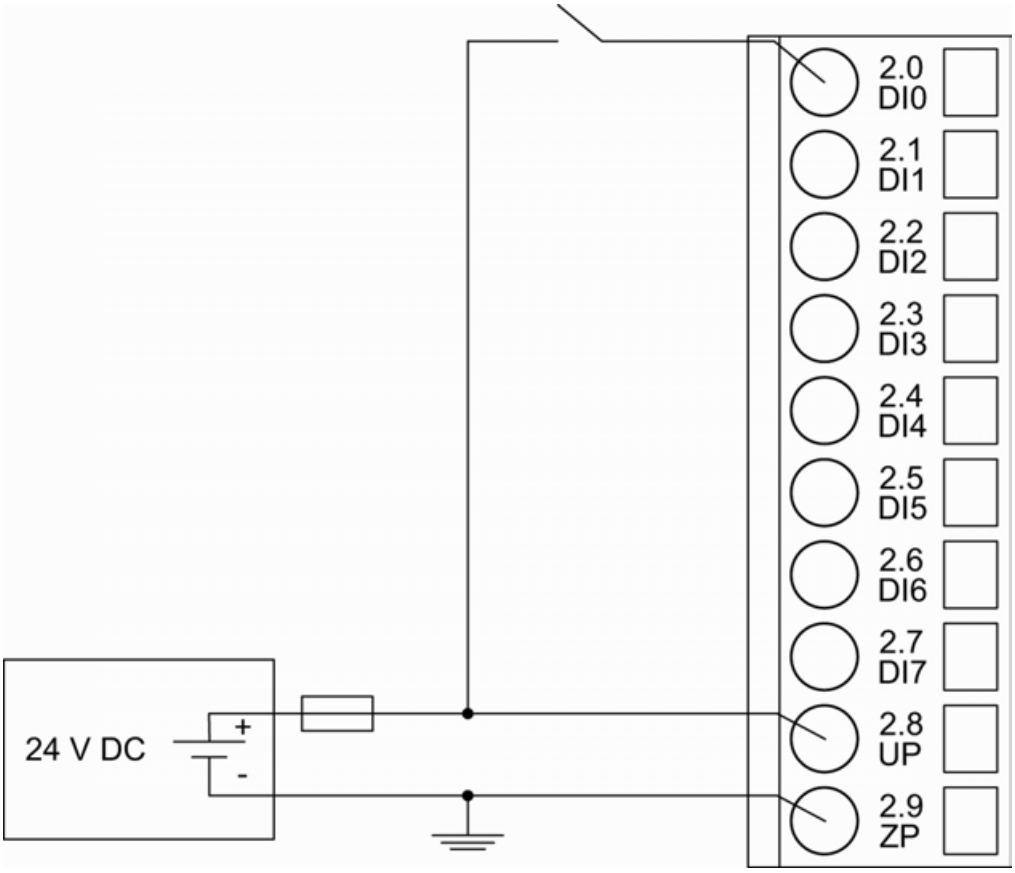


Fig. 235: Connection of the digital inputs to the module CI521-MODTCP

The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.4.1.8.2 “State LEDs” on page 3187*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

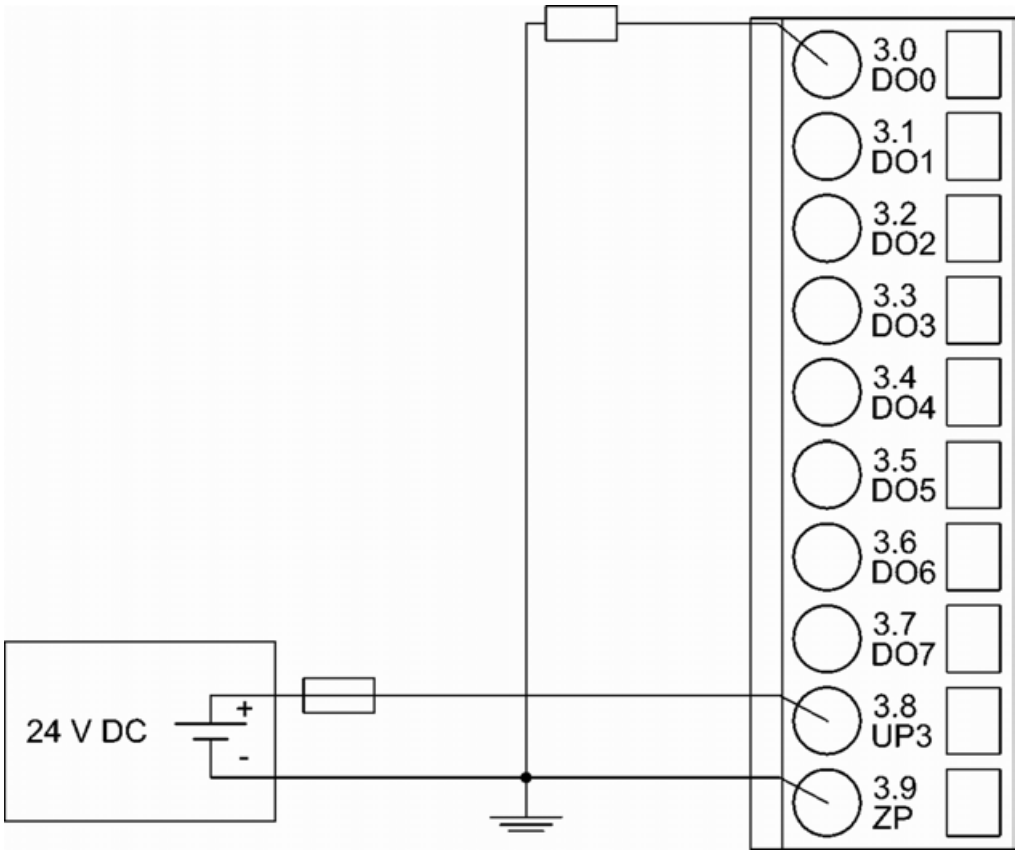


Fig. 236: Connection of configurable digital inputs/outputs to the module CI521-MODTCP

The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.4.1.8.2 “State LEDs” on page 3187*.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI521-MODTCP provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

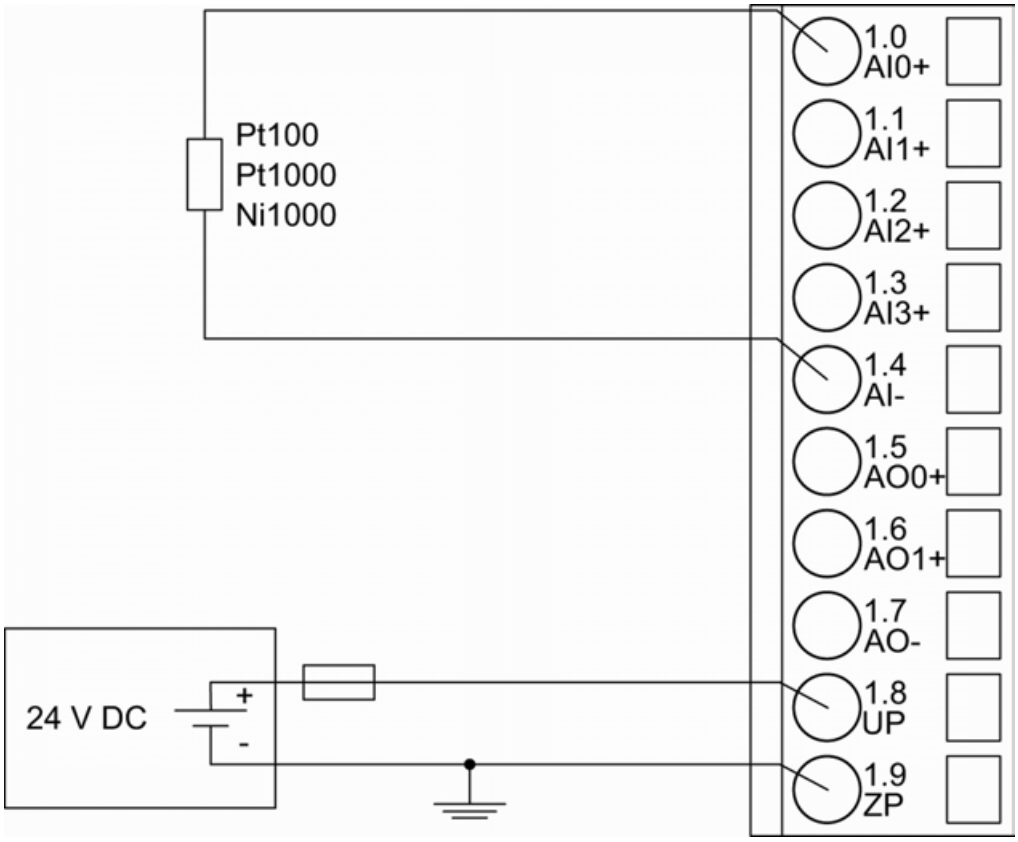


Fig. 237: Connection of resistance thermometers in 2-wire configuration to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Pt100	-50 °C...+70 °C	2-wire configuration, 1 channel used
Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

The module CI521-MODTCP performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI521-MODTCP provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

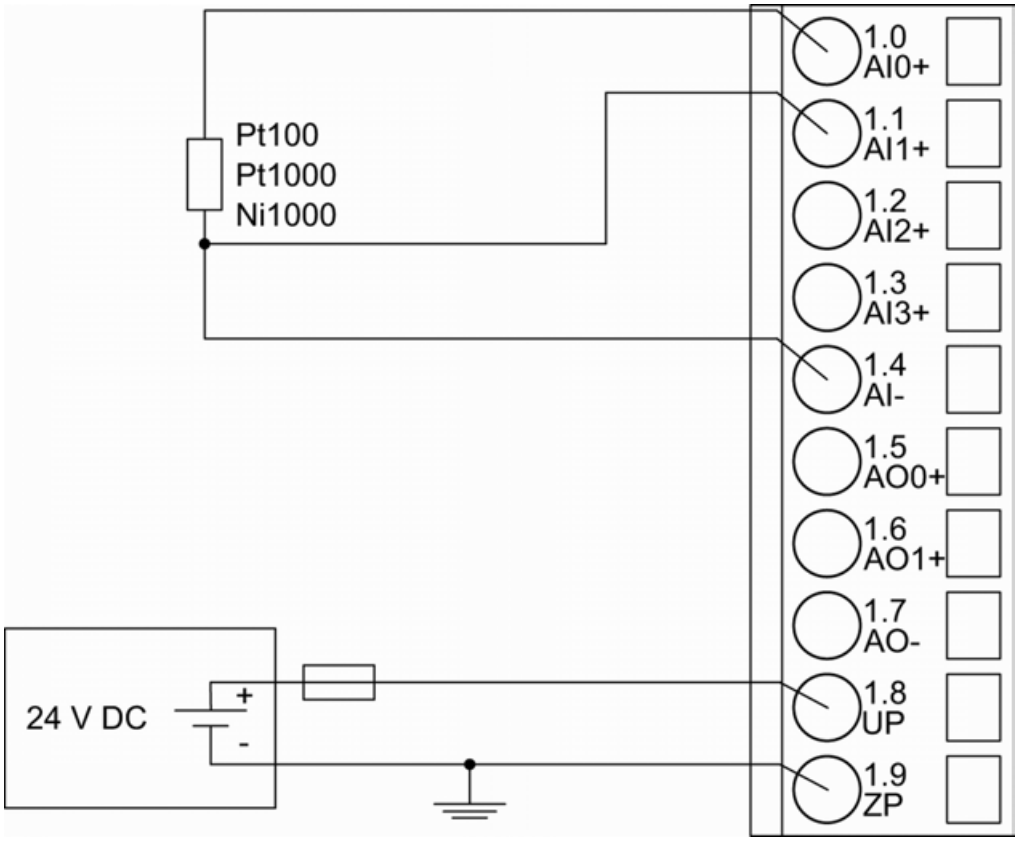


Fig. 238: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Pt100	-50 °C...+70 °C	3-wire configuration, 2 channels used
Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

The module CI521-MODTCP performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

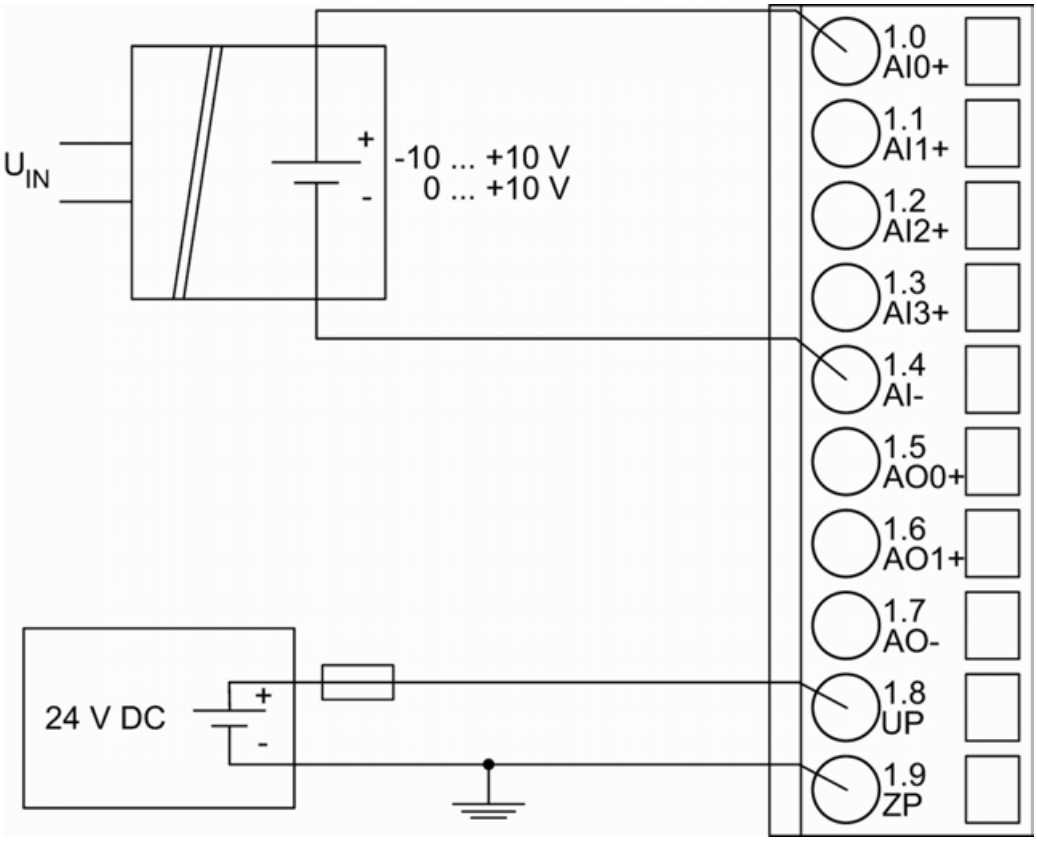


Fig. 239: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

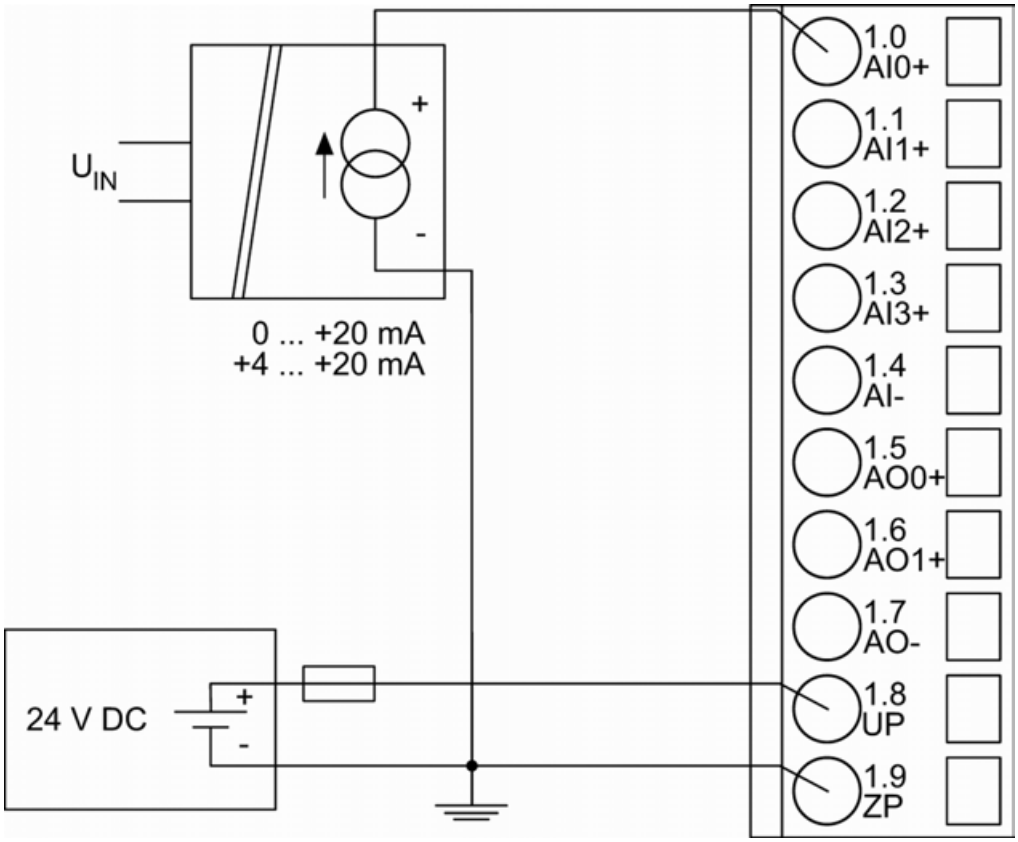


Fig. 240: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

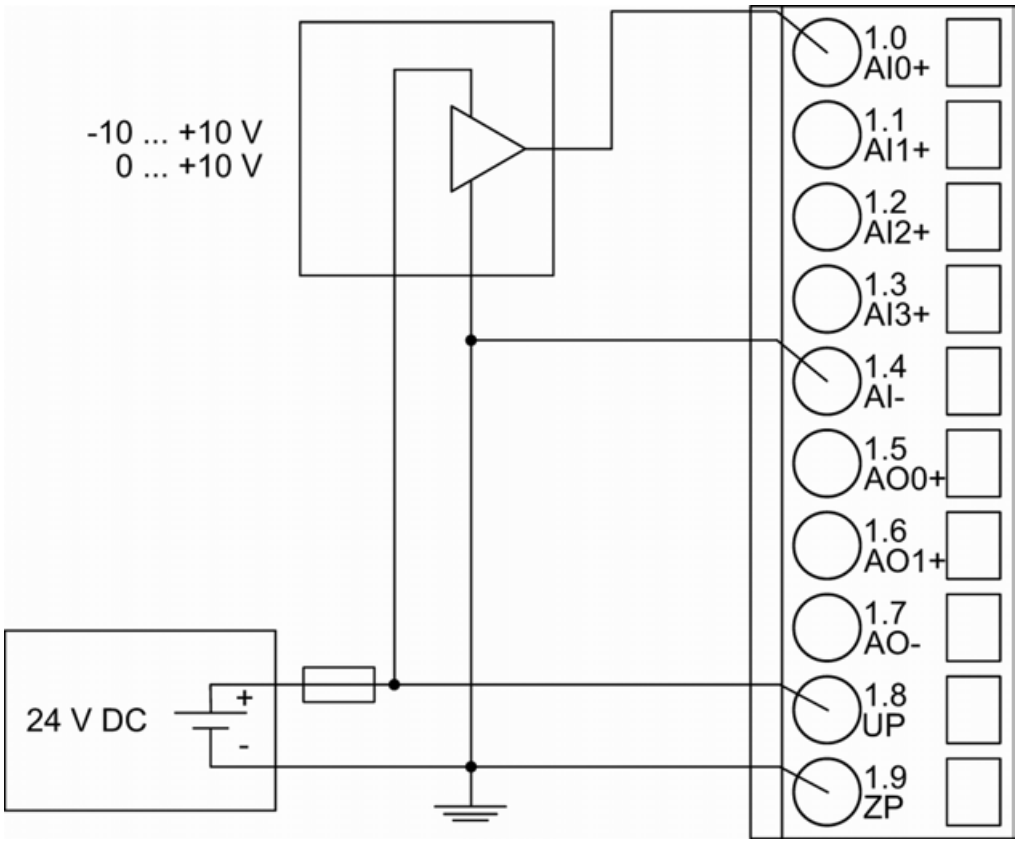


Fig. 241: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ± 1 V).

Make sure that the potential difference never exceeds ± 1 V (also not with long cable lengths).

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

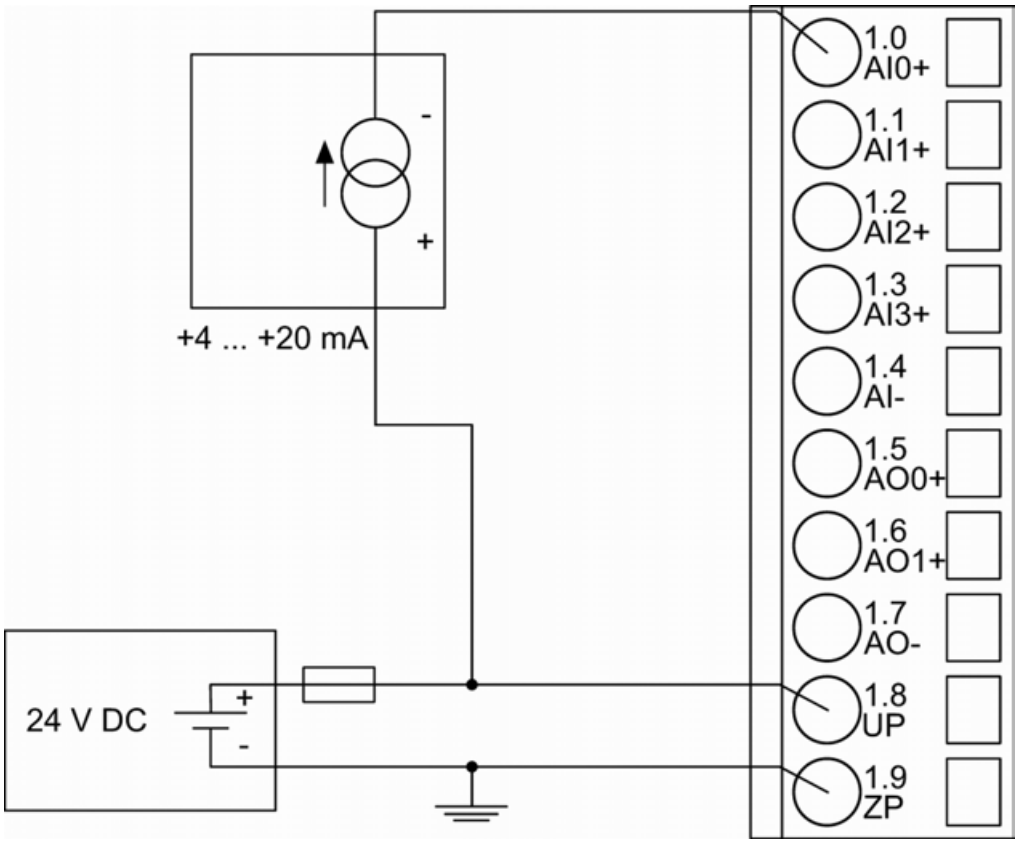


Fig. 242: Connection of passive-type analog sensors (current) to the analog inputs

The following measuring ranges can be configured ↬ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↬ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Current	4...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↬ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt zener diode in parallel to AIx+ and ZP.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

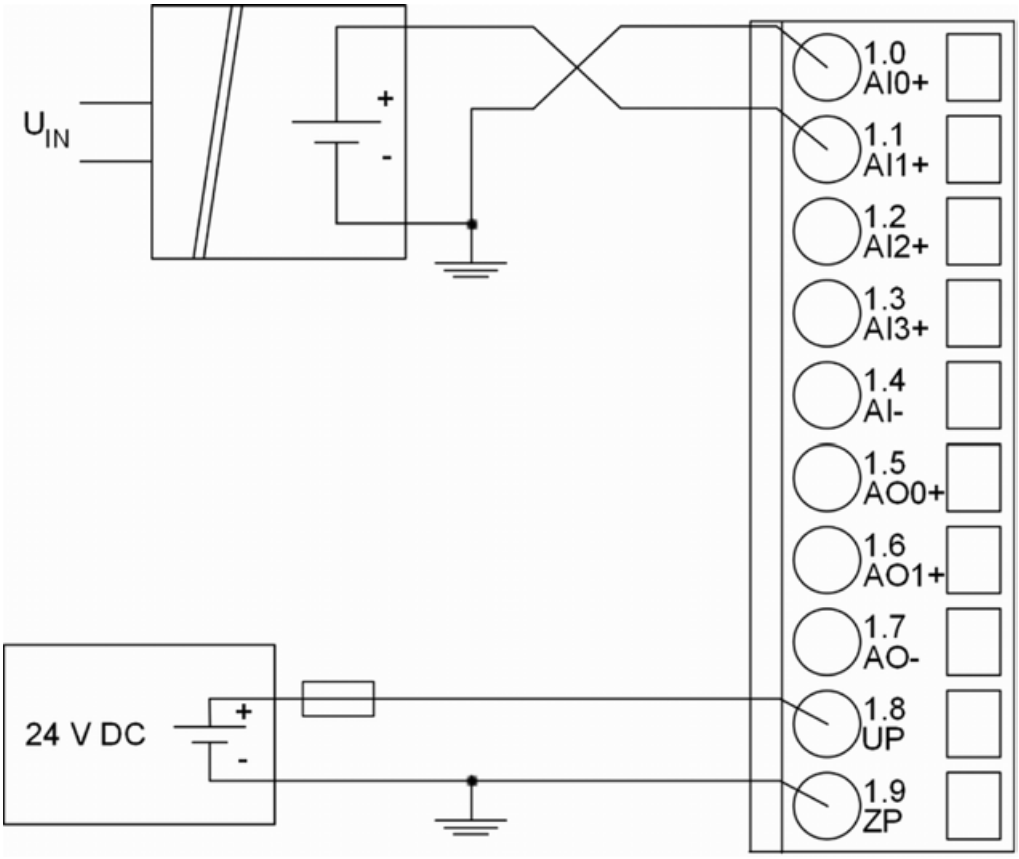


Fig. 243: Connection of active-type analog sensors (voltage) to differential analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Voltage	0...10 V	With differential inputs, 2 channels used
Voltage	-10 V...+10 V	With differential inputs, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs ↳ *Chapter 1.6.3.7.4.1.10.5 “Technical data of the analog inputs if used as digital inputs” on page 3194*. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

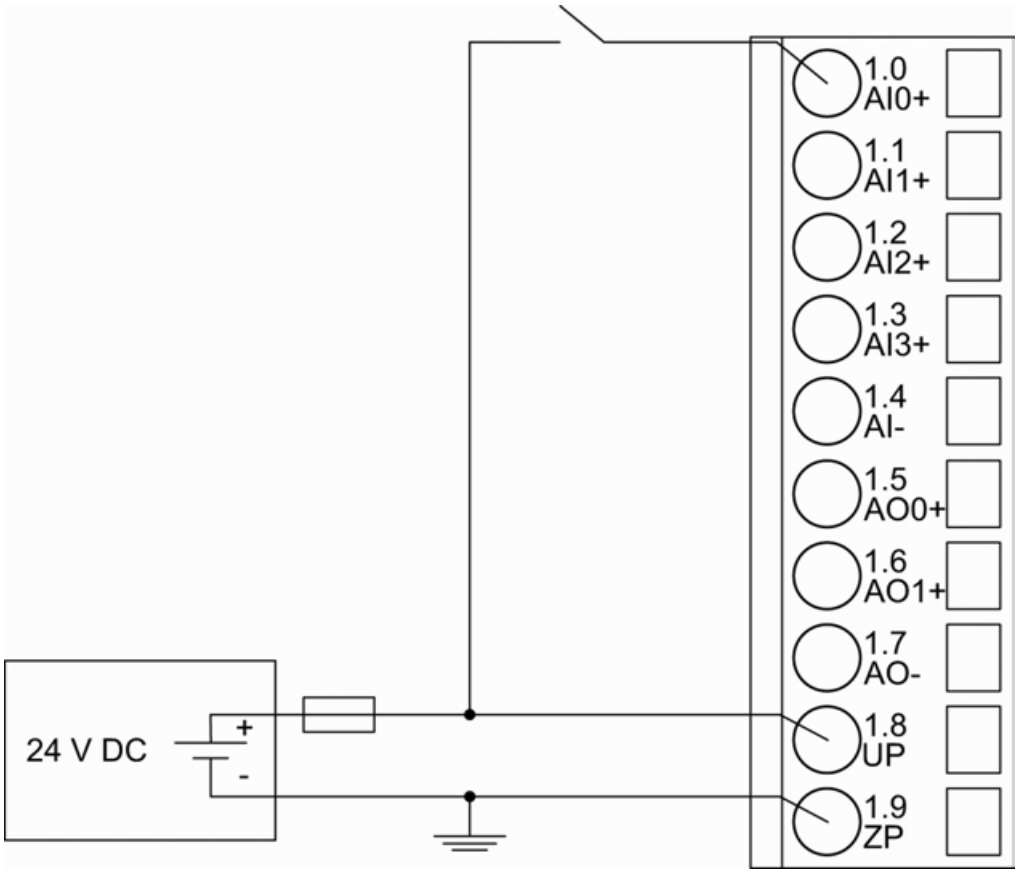


Fig. 244: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188* :

Digital input	24 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

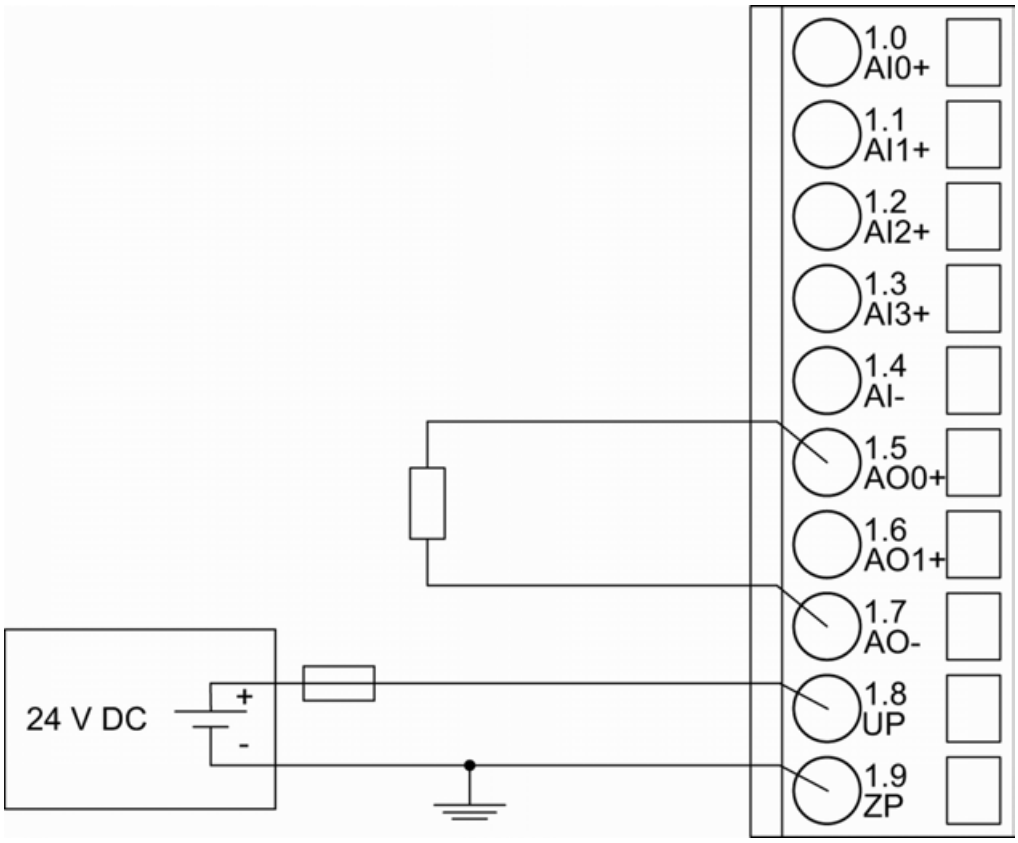


Fig. 245: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

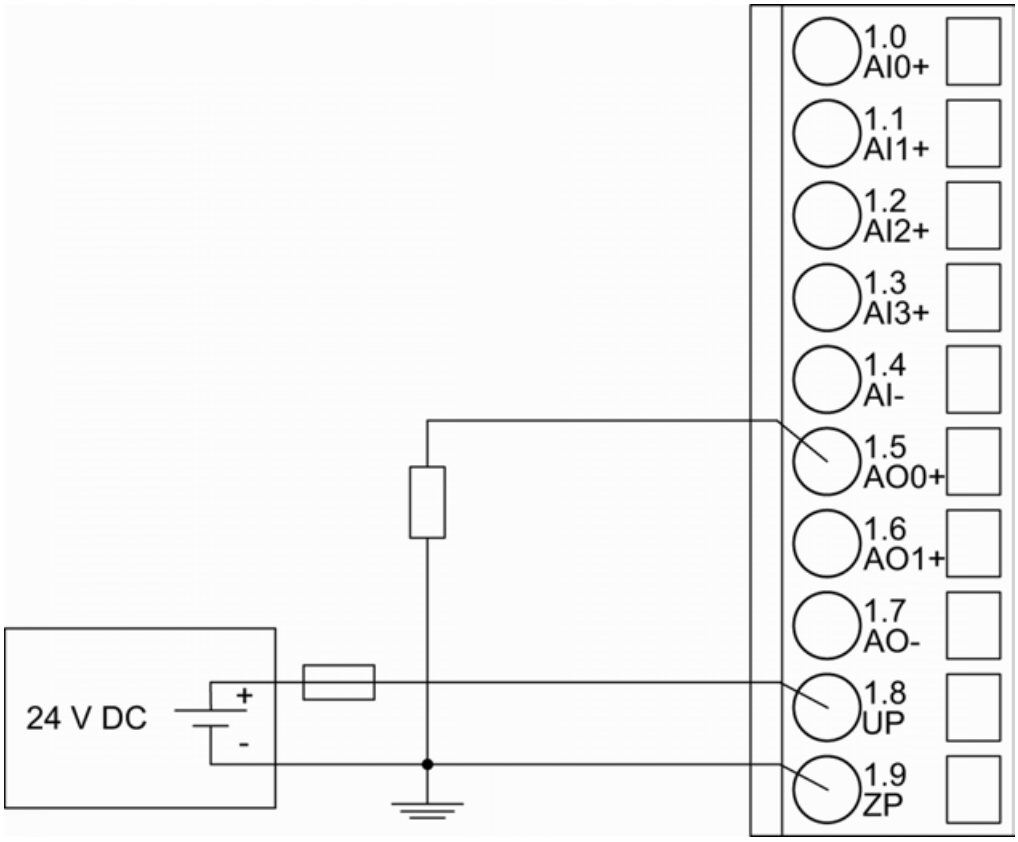


Fig. 246: Connection of analog output loads (current)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176* and ↳ *Chapter 1.6.3.7.4.1.9 “Measuring ranges” on page 3188*:

Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

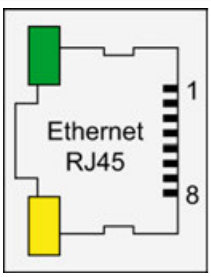
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*.

Unused analog outputs can be left open-circuited.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected

Interface	PIN	Signal	Description
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.*

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

The IP address of the CI521-MODTCP Module can be set with the "ABB IP Configuration Tool". *Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675*

If the last byte of the IP is set to 0, the address switch will be used instead.

Address switch position 255 is mapped to fixed IP 192.168.0.254 independent of other stored settings. This is a backup so the module can always get a valid IP address and can be configured by the “ABB IP Configuration Tool”.

Address switch position 0 is mapped to last byte equal 1 and DHCP enabled.

The factory setting for the IP is 192.168.0.x (last byte is address switch).

I/O configuration

The CI521-MODTCP stores configuration parameters (IP address configuration, module parameters).

The analog/digital I/O channels are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.3.7.4.1.7 “Parameterization” on page 3176*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7400	WORD	7000
Ignore Module	Internal	0	BYTE	0
Parameter length	Internal	63	BYTE	63
Error LED / Failsafe function see table Error LED / Failsafe function *Table 561 “Error LED / Failsafe function” on page 3177*	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0

Name	Value	Internal value	Internal value, type	Default
Timeout for Bus supervision	No supervision 10 ms timeout 20 ms timeout	0 1 2	BYTE	No supervision
IO Mapping Structure [3])	Fixed Mapping Dynamic Mapping	0 1	BYTE	0
Reserved	Internal	0	ARRAY[0..2] OF BYTE	0,0,0
Check supply	off on	0 1	BYTE	1
Fast counter	0 : 10 [3])	0 : 10	BYTE	0

[1]) With a faulty ID, the Modules reports a "parameter error" and does not perform cyclic process data transmission.

[2]) Counter operating modes, see description of the *Chapter 1.6.5.1.12 “Fast counters” on page 3570*.

[3]) Fixed Mapping means each module has its own Modbus registers for data transfer independent of the IO bus constellation. For details see *Chapter 1.6.5.3.1.2 “Modbus TCP registers” on page 3604*.

Dynamic mapping means the structure of the IO Date is dependent on the I/O bus constellation. Each I/O bus expansion module starts directly after the module before on the next Word adress.

[4]) If none of the parameters is set all masters / clients in the network have read and write rights on the CI52x-MODTCP device and its connected expansion modules.

If at least one parameter is set only the configured masters / clients have write rights on the CI52x-MODTCP device, all other masters / clients still have read access to the CI52x-MODTCP device.

Table 561: Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameters Behaviour AO at comm. error and Behaviour DO at comm. error are only analyzed if the Failsafe-mode is ON.	

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard Reserved	0 255	BYTE	0
Behaviour AO at comm. error *)	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	0
*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Table Operating modes of the analog inputs ↳ *Table 562 “Channel configuration” on page 3179*	Table Operating modes of the analog inputs ↳ *Table 562 “Channel configuration” on page 3179*	BYTE	0
Input 0, Check channel	Table Channel montoring ↳ *Table 563 “Channel monitoring” on page 3179*	Table Channel montoring ↳ *Table 563 “Channel monitoring” on page 3179*	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, Channel configuration	Table Operating modes of the analog inputs ↳ *Table 562 “Channel configuration” on page 3179*	Table Operating modes of the analog inputs ↳ *Table 562 “Channel configuration” on page 3179*	BYTE	0
Input 3, Check channel	Table Channel montoring ↳ *Table 563 “Channel monitoring” on page 3179*	Table Channel montoring ↳ *Table 563 “Channel monitoring” on page 3179*	BYTE	0

Table 562: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Table 563: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Table Operating modes of the analog outputs *Table 564 “Channel configuration” on page 3180*	Table Operating modes of the analog outputs *Table 564 “Channel configuration” on page 3180*	BYTE	0
Output 0, Check channel	Table Channel monitoring *Table 565 “Channel monitoring” on page 3180*	Table Channel monitoring *Table 565 “Channel monitoring” on page 3180*	BYTE	0

Name	Value	Internal value	Internal value, type	Default
Output 0, Substitute value	Table Substitute value ↳ *Table 566 “Substitute value” on page 3180*	Table Substitute value ↳ *Table 566 “Substitute value” on page 3180*	WORD	0
Output 1, Channel configuration	Table Operating modes of the analog outputs ↳ *Table 564 “Channel configuration” on page 3180*	Table Operating modes of the analog outputs ↳ *Table 564 “Channel configuration” on page 3180*	BYTE	0
Output 1, Check channel	Table Channel monitoring ↳ *Table 565 “Channel monitoring” on page 3180*	Table Channel monitoring ↳ *Table 565 “Channel monitoring” on page 3180*	BYTE	0
Output 1, Substitute value	Table Substitute value ↳ *Table 566 “Substitute value” on page 3180*	Table Substitute value ↳ *Table 566 “Substitute value” on page 3180*	WORD	0

Table 564: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 565: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 566: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 255	00h ... FFh	BYTE	0 0x0000
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	On 0x01

[1]) The parameters Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.

[2]) The state "externally voltage detected" appears, if the output of a channel DC0..DC7 should be switched on while an externally voltage is connected ↳ *Chapter 1.6.3.7.4.1.3 “Connections” on page 3158*. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".

Diagnosis and state LEDs

Structure of the diagnosis block

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI521-MODTCP (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O Module ... 10 = 10th connected S500 I/O Module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

For diagnosis firmware version ≥ 3.2.6 is required.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	No process voltage UP		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket		Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**					
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap terminal unit [9])		Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])		Remove wrong I/O module and plug projected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])		Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])		Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found		Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])		Restart, if error persists replace terminal unit
4	-	31	31	31	46	Voltage feedback on activated digital outputs DO0...DO7 on UP3 [4])		Check terminals

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	No process voltage UP3	Check process supply voltage
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Externally voltage detected at digital output DO0...DO7 [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at digital output [7])	Check terminals
Channel error analog							
4	-	31	1	0..3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0..3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0..3	47	Short circuit at an analog input	Check terminals

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	31	3	0..1	4	Analog value over-flow at an analog output	Check output value
4	-	31	3	0..1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI521-MODTCP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself; 1..10 = Expansion module
[3])	With "Module" the following allocation applies: 31 = Module itself Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DO0...DO7 cause that other digital outputs are supplied through that voltage ↳ *Chapter 1.6.3.7.4.1.3 “Connections” on page 3158*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0...DO7 has overrun the process supply voltage UP3 ↳ *Chapter 1.6.3.7.4.1.3 “Connections” on page 3158*. Diag-nosis message appears for the whole module.
[6])	This message appears, if the output of a channel DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 100ms. Then a new start up will be executed. This diagnosis message appears per channel.

[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 567: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with IO Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	Device configured, acyclic data exchange running
	Red	---	Communication error (timeout) appeared	IP address error
STA2 ETH (System LED "SF")	Green	Device has valid parameters	Device is running parameterization sequenze	Device has no parameters
	Red	---	---	Device has invalid parameters
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 568: States of the 27 process LEDs

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
Normal range or measured value too low	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
Normal range	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output

Parameter		Value
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of analog inputs		4
Number of analog outputs		2
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Ethernet		10/100 base-TX, internal switch, 2 x RJ45 socket
Setting of the IP address		With ABB IP config tool and 2 rotary switches at the front side of the module
Diagnose		See Diagnosis and Displays ↬ *Chapter 1.6.3.7.4.1.8 “Diagnosis and state LEDs” on page 3182*
Operation and error displays		32 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Extended ambient temperature (XC version)		> 60 °C on request
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 2.0 to 2.7
Reference potential for all inputs	Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC

Parameter		Value
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload

Parameter		Value
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

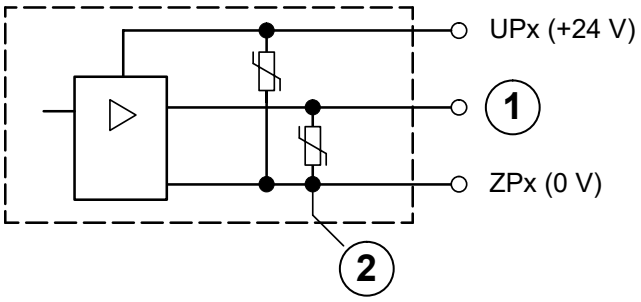


Fig. 247: Digital input/output (circuit diagram)

1 Digital Output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 1.0 to1.3
Reference potential for AI0+ to AI3+		Terminal 1.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9 and 3.9 for current measurement
Input type		
	Unipolar	Voltage 0 ... 10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 ... +10 V
Galvanic isolation		Against Ethernet network
Configurability		0...10 V, -10...+10 V, 0/4...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s

Parameter	Value
Resolution	Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Tables Input ranges voltage, current and digital input ↬ *Chapter 1.6.3.7.4.1.9.1 “Input ranges voltage, current and digital input” on page 3188* Input range resistance temperature detector ↬ *Chapter 1.6.3.7.4.1.9.2 “Input ranges resistance temperature detector” on page 3189*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for the inputs		Terminals 1.9, 2.9 and 3.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter	Value
Number of channels per module	2
Distribution of channels into groups	1 group for 2 channels
Connection of the channels AO0+...AO1+	Terminals 1.5...1.6

Parameter		Value
Reference potential for AO0+ to AO1+		Terminal 1.7 (AO-) for voltage outputTerminal 1.9, 2.9 and 3.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10...+10 V, 0...20 mA, 4...20 mA (each output can be configured individually)
Output resistance (load), as current output		0...500 Ω
Output loadability, as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		Table Output ranges voltage and current ↳ *Chapter 1.6.3.7.4.1.9.3 “Output ranges voltage and current” on page 3190*
Unused outputs		Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI0), 2.1 (DI1)
Used outputs	Terminal 3.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	See ↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

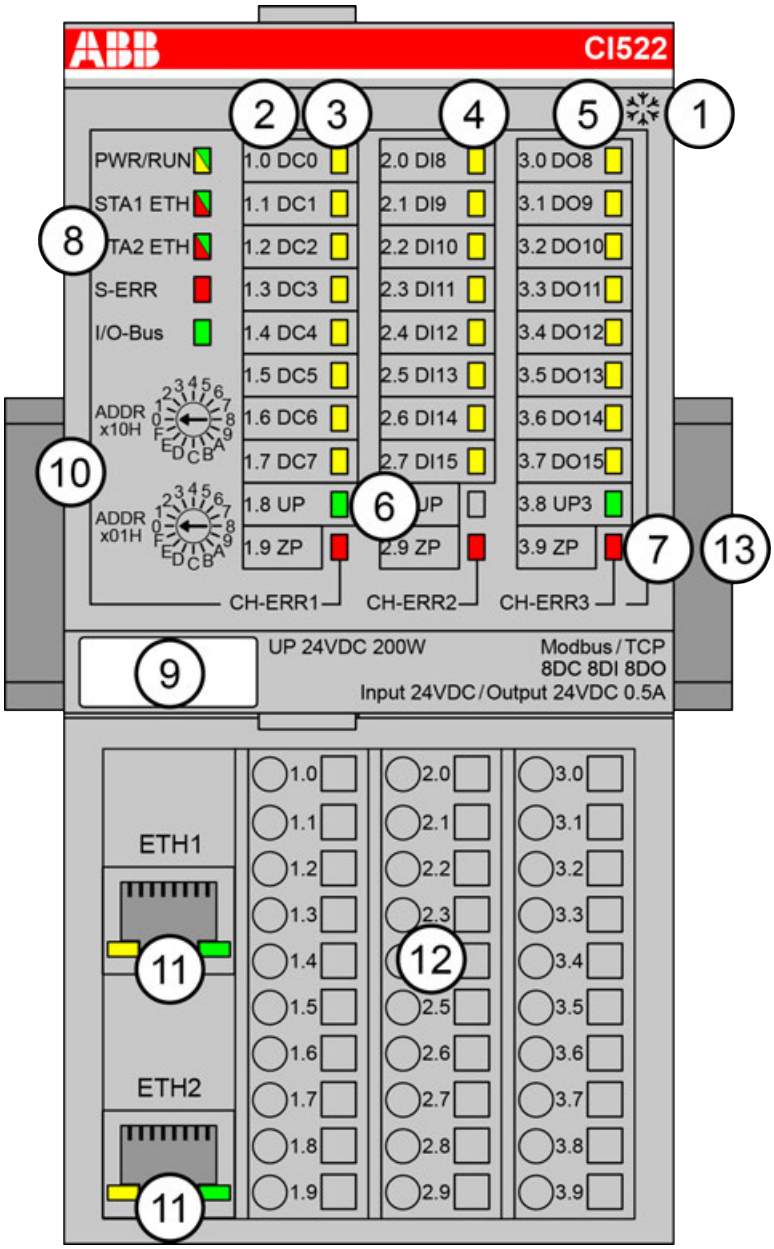
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 222 100 R0001	CI521-MODTCP, Modbus TCP communication interface module, 4 AI, 2 AO, 8 DI and 8 DO	Active
1SAP 422 100 R0001	CI521-MODTCP-XC, Modbus TCP communication interface module, 4 AI, 2 AO, 8 DI and 8 DO, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CI522-MODTCP

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital configurable inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the IP address
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

Modbus TCP communication interface module CI522-MODTCP is used as decentralized I/O module in Modbus TCP networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the configurable digital inputs/outputs is performed by software.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Interface	Ethernet
Protocol	Modbus TCP
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	for setting the last BYTE of the IP ADDRESS (00h to FFh)
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Required terminal unit	TU507 or TU508 ↳ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*

Connections

The Ethernet bus module CI522-MODTCP is plugged on the I/O terminal unit TU507-ETH ↳ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549* or TU508-ETH ↳ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Conditions for undisturbed operating with older I/O expansion modules

All I/O expansion modules that are attached to the CI52x-MODTCP must be powered up together with the CI52x-MODTCP if the firmware version of these I/O expansion modules is V1.9 or lower.

The firmware version is related to the index. The index is printed on the module type label on the right side.

Modules as of index listed in the following table can be powered up independently.

S500 I/O module type	First index with firmware version above 1.9
AI523	D0
AI523-XC	D0
AI531	A3
AI531-XC	A0
AO523	D0
AO523-XC	D0
AX521	D0
AX521-XC	D0
AX522	D0
AX522-XC	D0
CD522	A2
CD522-XC	A0
DA501	A2
DA501-XC	A0
DA502	A1
DA502-XC	A1
DC522	D0
DC522-XC	D0
DC523	D0
DC523-XC	D0
DC532	D0
DC532-XC	D0
DI524	D0

S500 I/O module type	First index with firmware version above 1.9
DI524-XC	D0
DO524	A2
DO524-XC	A2
DX522	D0
DX522-XC	D0
DX531	D0
AC522	D0
PD501	D0

Do not connect any voltages externally to digital outputs!

This ist not intended usage.

Reason: Externally voltages at one or more terminals DC0...DC7 or DO8...DO15 may cause that other digital outputs are supplied through that voltage instead of voltage UP3 (reverse voltage).

This is also possible, if DC channels are used as inputs. For this, the source for the input signals should be the impressed UP3 of the device.

This limitation does not apply for the input channels DI0..DI7.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO8...DO15 and DC0...DC7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	DC0	Signal of the configurable digital input/output DC0
1.1	DC1	Signal of the configurable digital input/output DC1
1.2	DC2	Signal of the configurable digital input/output DC2
1.3	DC3	Signal of the configurable digital input/output DC3
1.4	DC4	Signal of the configurable digital input/output DC4
1.5	DC5	Signal of the configurable digital input/output DC5
1.6	DC6	Signal of the configurable digital input/output DC6
1.7	DC7	Signal of the configurable digital input/output DC7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)

Terminal	Signal	Description
2.0	DI8	Signal of the digital input DI8
2.1	DI9	Signal of the digital input DI9
2.2	DI10	Signal of the digital input DI10
2.3	DI11	Signal of the digital input DI11
2.4	DI12	Signal of the digital input DI12
2.5	DI13	Signal of the digital input DI13
2.6	DI14	Signal of the digital input DI14
2.7	DI15	Signal of the digital input DI15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO8	Signal of the digital output DO8
3.1	DO9	Signal of the digital output DO9
3.2	DO10	Signal of the digital output DO10
3.3	DO11	Signal of the digital output DO11
3.4	DO12	Signal of the digital output DO12
3.5	DO13	Signal of the digital output DO13
3.6	DO14	Signal of the digital output DO14
3.7	DO15	Signal of the digital output DO15
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the Ethernet bus module CI522-MODTCP.

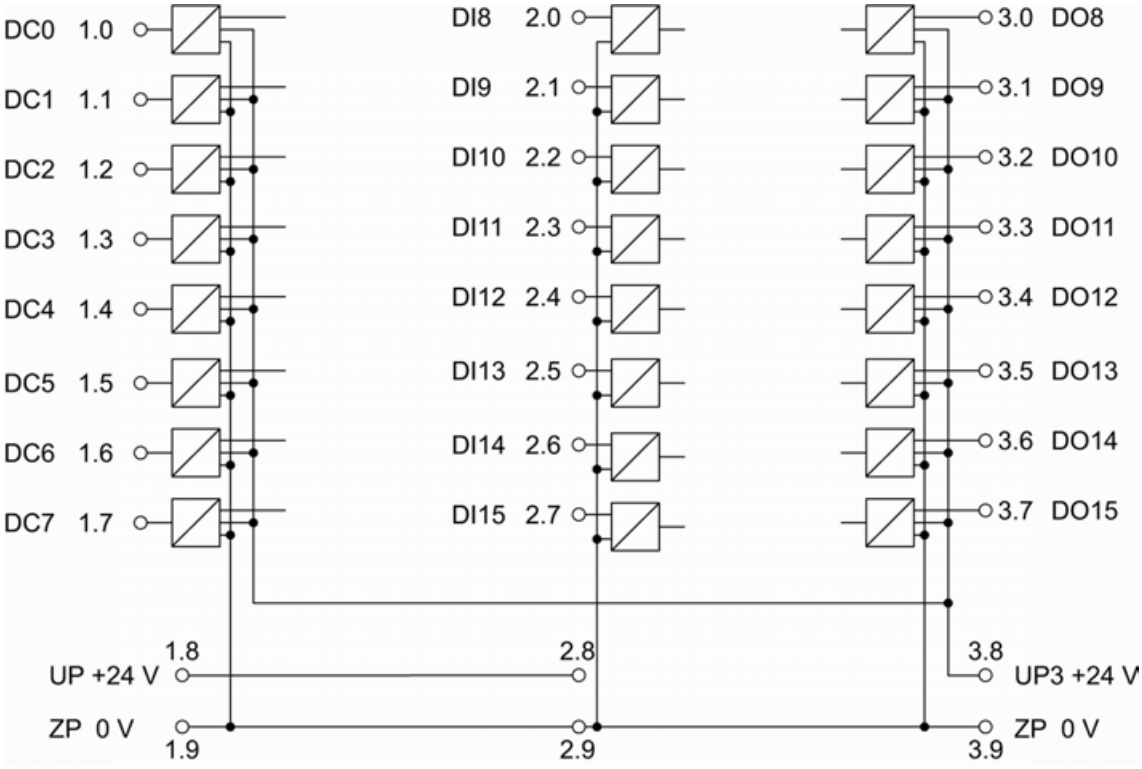


Fig. 248: Connection of the communication interface module CI522-MODTCP

Further information is provided in the System Technology chapter *Chapter 1.6.5.3.1 “Modbus communication interface module” on page 3603*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.

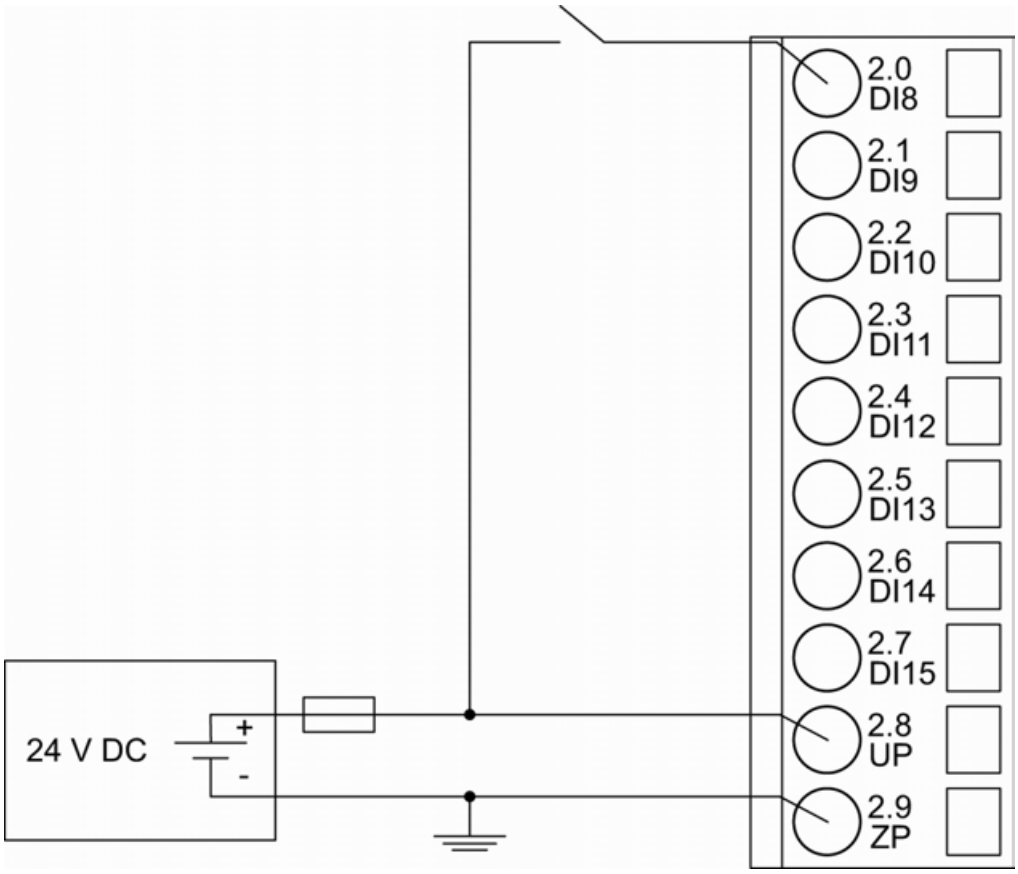
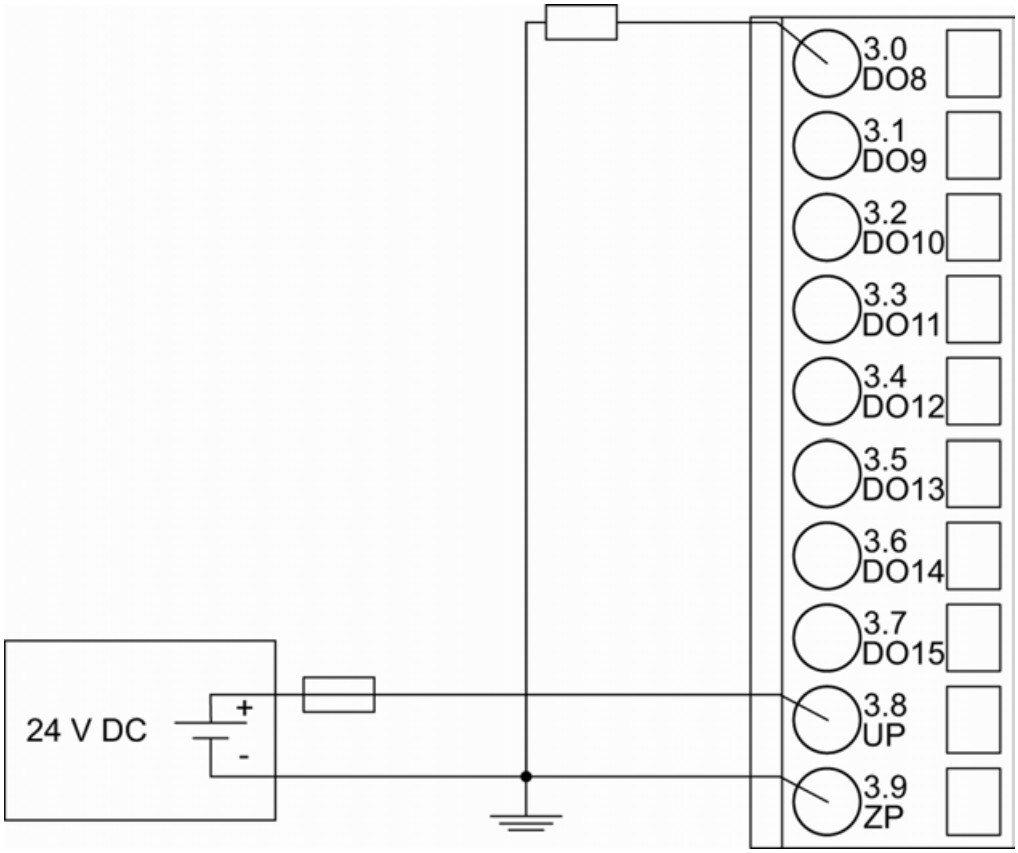


Fig. 249: Connection of the digital inputs to the module CI522-MODTCP

The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.4.2.8.1 “State LEDs” on page 3214*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

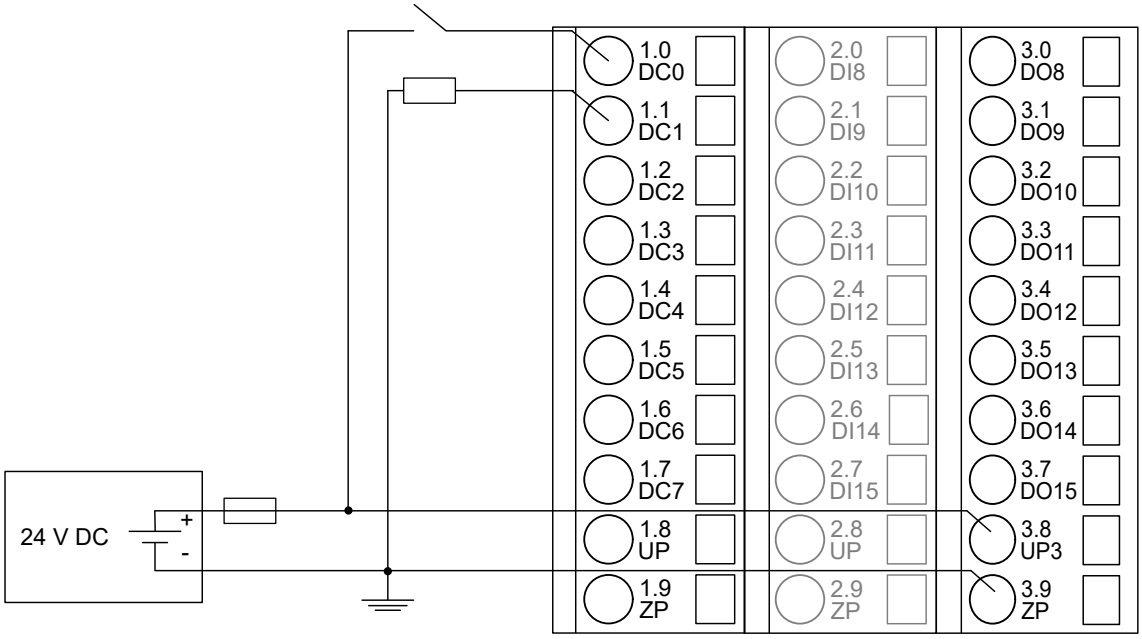


The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.4.2.8.1 “State LEDs” on page 3214*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

CAUTION!
If a DC channel is used as input, the source for the input signals should be the impressed UP3 of the device *Chapter 1.6.3.7.4.2.3 “Connections” on page 3198*.

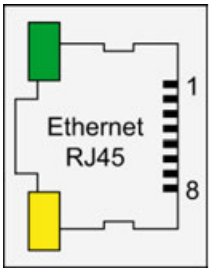


The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.4.2.8.1 “State LEDs” on page 3214*.

Assignment of the Ethernet ports

The terminal unit for the Communication Interface Module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.*

Internal data exchange

Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

The IP address of the CI5221-MODTCP Module can be set with the “ABB IP Configuration Tool”. *Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675*.

If the last byte of the IP is set to 0, the address switch will be used instead.

Address switch position 255 is mapped to fixed IP 192.168.0.254 independent of other stored settings. This is a backup so the module can always get a valid IP address and can be configured by the “ABB IP Configuration Tool”.

Address switch position 0 is mapped to last byte equal 1 and DHCP enabled.

The factory setting for the IP is 192.168.0.x (last byte is address switch).

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI522-MODTCP stores configuration parameters (IP address configuration, module parameters).

The digital I/O channels are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.3.7.4.2.7 “Parameterization” on page 3206*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7405	WORD	7405
Ignore Module	Internal	0	BYTE	0
Parameter length	Internal	47	BYTE	47
Error LED / Failsafe function (Table Error LED / Failsafe function *Table 569 “Table Error LED / Failsafe function” on page 3208*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		

Name	Value	Internal value	Internal value, type	Default
	Off by E3 + fail-safe	19		
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Timeout for Bus supervision	No supervision 10 ms timeout 20 ms timeout	0 1 2	BYTE	No supervision
IO Mapping Structure [3])	Fixed Mapping Dynamic Mapping	0 1	BYTE	0
Reserved	Internal	0	ARRAY[0..2] OF BYTE	0,0,0
Check supply	off on	0 1	BYTE	1
Fast counter	0 : 10 [2])	0 : 10	BYTE	0

Remarks:

[1])	With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.
[2])	Counter operating modes *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*

[3])	Fixed Mapping means each module has its own Modbus registers for data transfer independent of the I/O bus constellation description. For details see *Chapter 1.6.5.3.1.2 “Modbus TCP registers” on page 3604*. Dynamic mapping means the structure of the IO Date is dependent on the I/O bus constellation. Each I/O bus expansion module starts directly after the module before on the next Word adress.
[4])	If none of the parameters is set all masters / clients in the network have read and write rights on the CI52x-MODTCP device and its connected expansion modules. If at least one parameter is set only the configured masters / clients have write rights on the CI52x-MODTCP device, all other masters / clients still have read access to the CI52x-MODTCP device.

Table 569: Table Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameter Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.	

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 65535	0000h ... FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behaviour DO at comm. error is apply to DC and DO channels and only analyzed if the Failsafe-mode is ON.
[2])	The state "externally voltage detected" appears, if the output of a channel DC0...DC7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears, if externally voltage at digital outputs DC0...DC7 and accordingly DO8...DO15 has exceeded the process supply voltage UP3 ↳ *Chapter 1.6.3.7.4.2.3 “Connections” on page 3198* (see description in section). The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters on "OFF". This parameter should only be disabled in exceptional cases for voltage overflow may produce reverse voltage.

Diagnosis

Structure of the Diagnosis Block

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI502-PNIO (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O Module ... 10 = 10th connected S500 I/O Module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error
6	Reserved	0

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

For diagnosis firmware version ≥ 3.2.6 is required.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket		Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**					
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap terminal unit [9])		Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])		Remove wrong I/O module and plug projected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])		Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])		Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found		Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])		Restart, if error persists replace terminal unit
4	1...6	255	2	0	45	The connected Communication Module has no connection to the network		Check cabeling

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage
4	-	31	31	31	46	Reverse voltage from digital out-puts DO8...DO15 to UP3 [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow at outputs (above UP3 level) [5])	Check termi-nals/ check process supply voltage
Channel error digital							
4	-	31	2	8..15	46	Externally voltage detected at digital output DO8...DO15 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0...DC7 [6])	Check terminals
4	-	31	4	0...7	47	Short circuit at digital output DC0...DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO8...DO15[7])	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI502-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1..10 = Expansion module
[3])	With "Module" the following allocation applies dependent of the master: Module error: 31 = Module itself Channel error: Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DC0...DC7 oder DO8...DO15 cause that other digital outputs are supplied through that voltage (voltage feedback, see description in 'Connections' ↳ *Chapter 1.6.3.7.4.2.3 "Connections" on page 3198*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage at digital outputs DC0...DC7 and accordingly DO8...DO15 has exceeded the process supply voltage UP3 ↳ *Chapter 1.6.3.7.4.2.3 "Connections" on page 3198*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DC0...DC7 or DO8...DO15 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 2000ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 570: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---

LED	Color	OFF	ON	Flashing
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	Device configured, acyclic data exchange running
	Red	---	Communication error (timeout) appeared	IP address error
STA2 ETH (System LED "SF")	Green	Device has valid parameters	Device is running parameterization sequenze	Device has no parameters
	Red	---	---	Device has invalid parameters
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 571: States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of configurable digital inputs/outputs		8
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Ethernet		10/100 base-TX, internal switch, 2 x RJ45 socket
Setting of the I/O device identifier		With 2 rotary switches at the front side of the module
Diagnosis		See Diagnosis and Displays *Chapter 1.6.3.7.4.2.8 “Diagnosis” on page 3209*
Operation and error displays		34 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40°C per group)
Extended ambient temperature (XC version)		> 60 °C on request
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI8 to DI15		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO8 to DO15		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

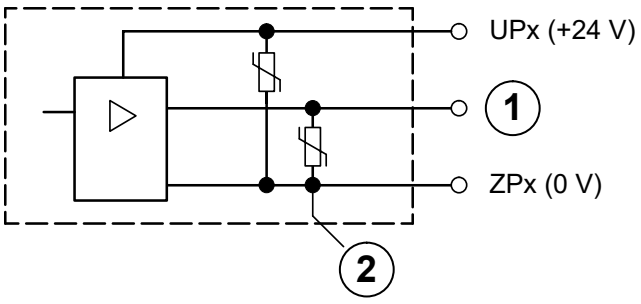


Fig. 250: Digital input/output (circuit diagram)

1 Digital Output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC7	Terminals 1.0...1.7
If the channels are used as outputs		
	Channels DC0...DC7	Terminals 1.0...1.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the Ethernet network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0,8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

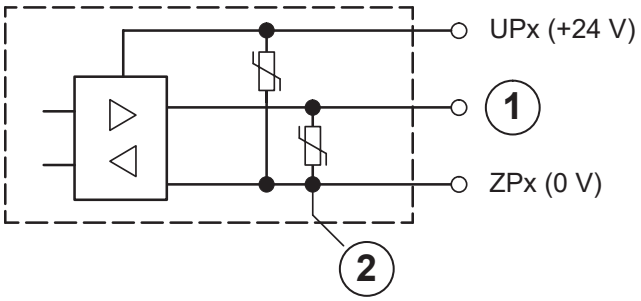


Fig. 251: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI8),Terminal 2.1 (DI9)
Used outputs	Terminal 3.0 (DO8)
Counting frequency	Depending on operation mode: Mode 1- 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	See *Chapter 1.6.5.1.12 "Fast counters" on page 3570*

Ordering data

Ordering No.	Scope of delivery	Product life cycle phase *)
1SAP 222 200 R0001	CI522-MODTCP, Modbus TCP communication interface module, 8 DC, 8 DI and 8 DO	Active
1SAP 422 200 R0001	CI522-MODTCP-XC, Modbus TCP communication interface module, 8 DC, 8 DI and 8 DO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.7.5 PROFINET

Comparison of the CI5xx-PNIO modules

The PROFINET IO devices combine the advantages of decentralized I/O modules with the reaction time of AC500 mounted central I/O modules. The devices for PROFINET provide the extension -PNIO in the device name.

The communication module CM579-PNIO acts as I/O controller in a PROFINET network. It is connected to the processor module via an internal communication bus. Depending on the terminal base, several communication modules can be used for one processor module.

The communication interface modules CI5xx-PNIO act as I/O devices in a PROFINET network.

Additionally the communication module CM589-PNIO(-4) can be used to setup a AC500 PLC to act as I/O module in a PROFINET network.

The difference of the CI5xx-PNIO devices can be found in their input and output characteristics ↳ *Chapter 1.6.3.7.5.1.1.1 “Characteristics of CI50x-PNIO” on page 3222*.

PROFINET IO devices CI50x-PNIO

Characteristics of CI50x-PNIO

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Input/Output characteristics of CI501-PNIO

The PROFINET communication interface module CI501-PNIO is used as decentralized I/O module in PROFINET networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (1.0...1.3), configurable as:
  - -10 ... +10 V
  - 0 ... +10 V
  - -10 ... +10 V (differential voltage)
  - 0 ... 20 mA
  - 4 ... 20 mA
  - Pt100 , Pt1000, Ni1000 (for each 2-wire and 3-wire)
  - 24 V digital input function
- 2 analog outputs (1.5...1.6), configurable as:
  - -10 ... +10 V
  - 0 ... 20 mA
  - 4 ... 20 mA
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital transistor outputs 24 V DC (0.5 A max.) in 1 group (3.0...3.7)
- Resolution of the analog channels: 12 bits

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Input/Output characteristics of CI502-PNIO

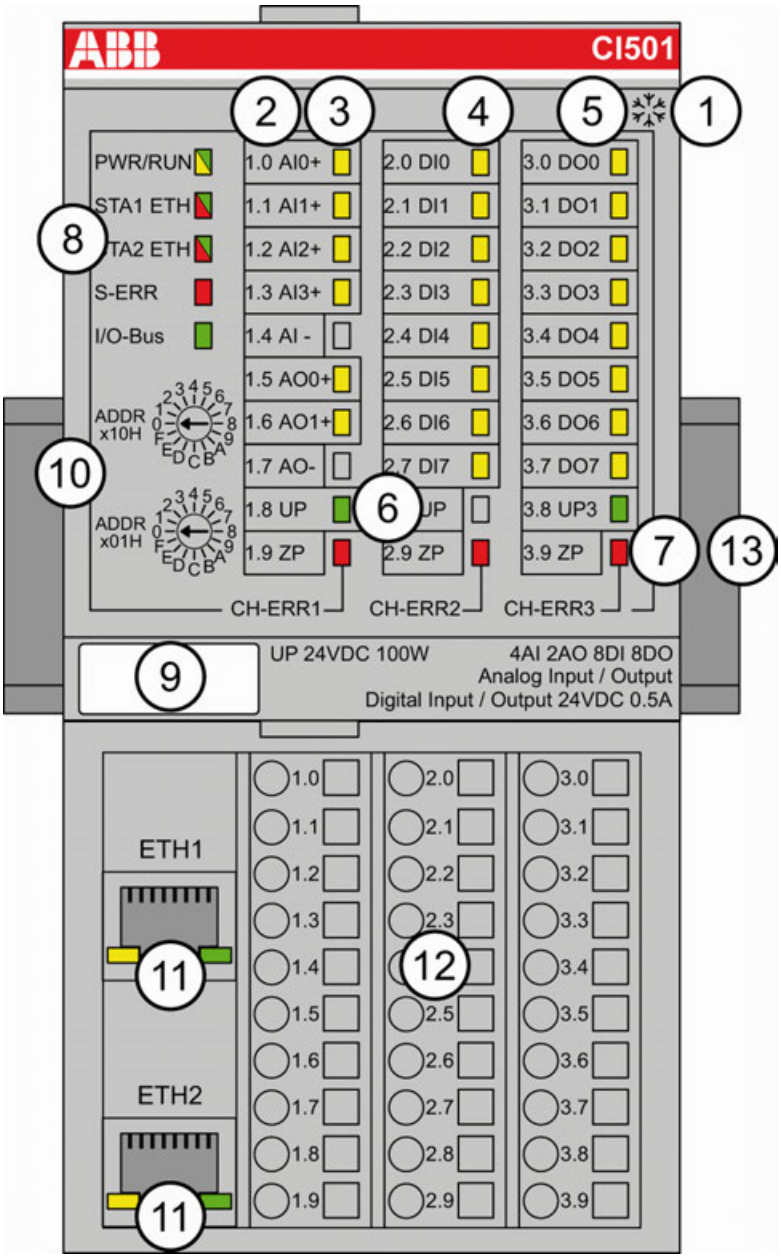
- 8 digital inputs 24 V DC
- 8 digital transistor outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- XC version for usage in extreme ambient conditions available

Technical data of the serial interfaces of CI504-PNIO

Parameter	Value
Number of serial interfaces	3
Connectors for serial interfaces	X11 for COM1 X12 for COM2 X13 for COM3
Supported physical layers	RS-232 RS-422 RS-485
Supported protocols	ASCII
Transmission rate	Configurable from 300 bit/s to 115.200 bit/s

CI501-PNIO

- 4 analog inputs, 2 analog outputs, 8 digital inputs, 8 digital outputs
- Resolution 12 bits plus sign
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the I/O device identifier
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit

12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The PROFINET communication interface modules CI501-PNIO and CI502-PNIO are used as communication interface modules in PROFINET networks. The network connection is performed by Ethernet cables which are inserted in the RJ45 connectors in the terminal unit. An Ethernet switch in the communication interface module allows daisy chaining of the network.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

The communication interface module contains 22 I/O channels with the following properties:

- 4 configurable analog inputs (2-wire / single-ended) or 2 configurable analog inputs (3-wire / differential) (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC, 0.5 A max. in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the PROFINET network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

Parameter	Value
Interface	Ethernet
Protocol	PROFINET IO RT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the I/O device identifier for configuration purposes (00h to FFh)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507 or TU508 ↬ *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*

Connections

The Ethernet communication interface module CI501-PNIO is plugged on the I/O terminal unit TU507-ETH or TU508-ETH *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to digital outputs!

Reason: External voltages at an output or several outputs may cause that other outputs are supplied through that voltage instead of voltage UP3 (reverse voltage). This is unintended usage.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is connected at the outputs DO0...DO7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	AI0+	Positive pole of analog input signal 0
1.1	AI1+	Positive pole of analog input signal 1
1.2	AI2+	Positive pole of analog input signal 2
1.3	AI3+	Positive pole of analog input signal 3
1.4	AI-	Negative pole of analog input signals 0 to 3
1.5	AO0+	Positive pole of analog output signal 0
1.6	AO1+	Positive pole of analog output signal 1
1.7	AI-	Negative pole of analog output signals 0 and 1
1.8	UP	Process voltage UP (24 V DC)

Terminal	Signal	Description
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI0	Signal of the digital input DI0
2.1	DI1	Signal of the digital input DI1
2.2	DI2	Signal of the digital input DI2
2.3	DI3	Signal of the digital input DI3
2.4	DI4	Signal of the digital input DI4
2.5	DI5	Signal of the digital input DI5
2.6	DI6	Signal of the digital input DI6
2.7	DI7	Signal of the digital input DI7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO0	Signal of the digital output DO0
3.1	DO1	Signal of the digital output DO1
3.2	DO2	Signal of the digital output DO2
3.3	DO3	Signal of the digital output DO3
3.4	DO4	Signal of the digital output DO4
3.5	DO5	Signal of the digital output DO5
3.6	DO6	Signal of the digital output DO6
3.7	DO7	Signal of the digital output DO7
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

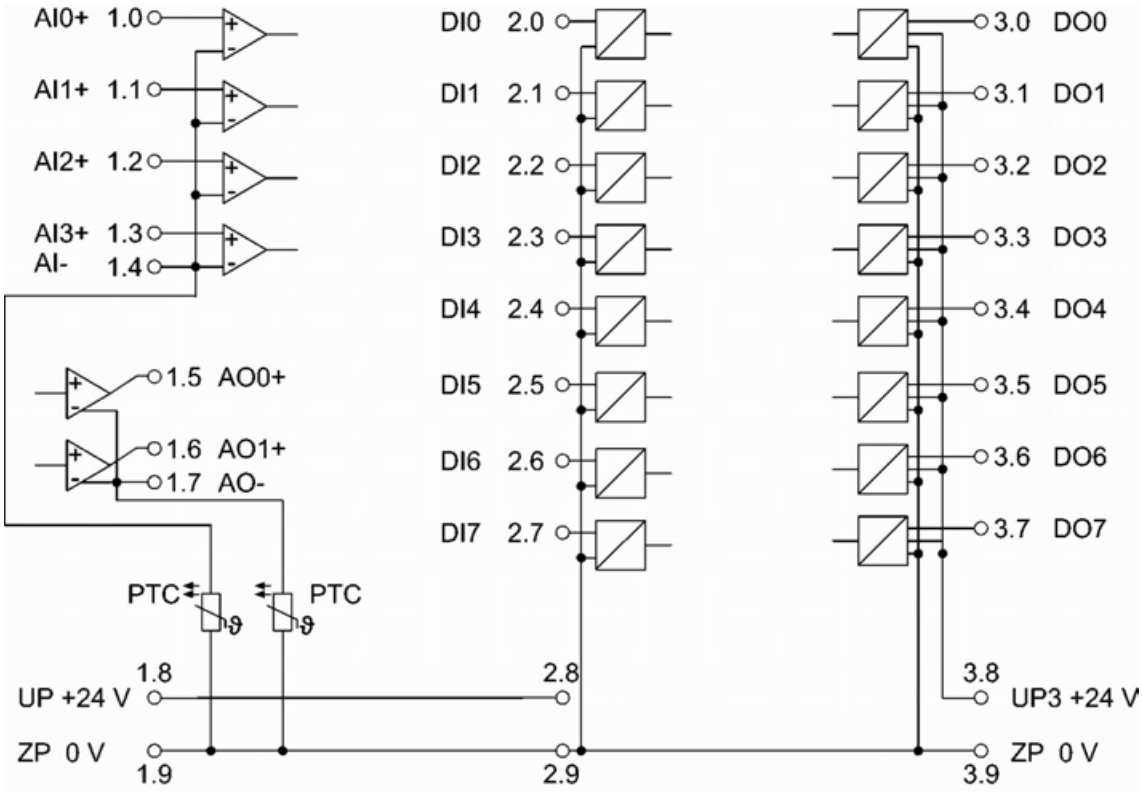
– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

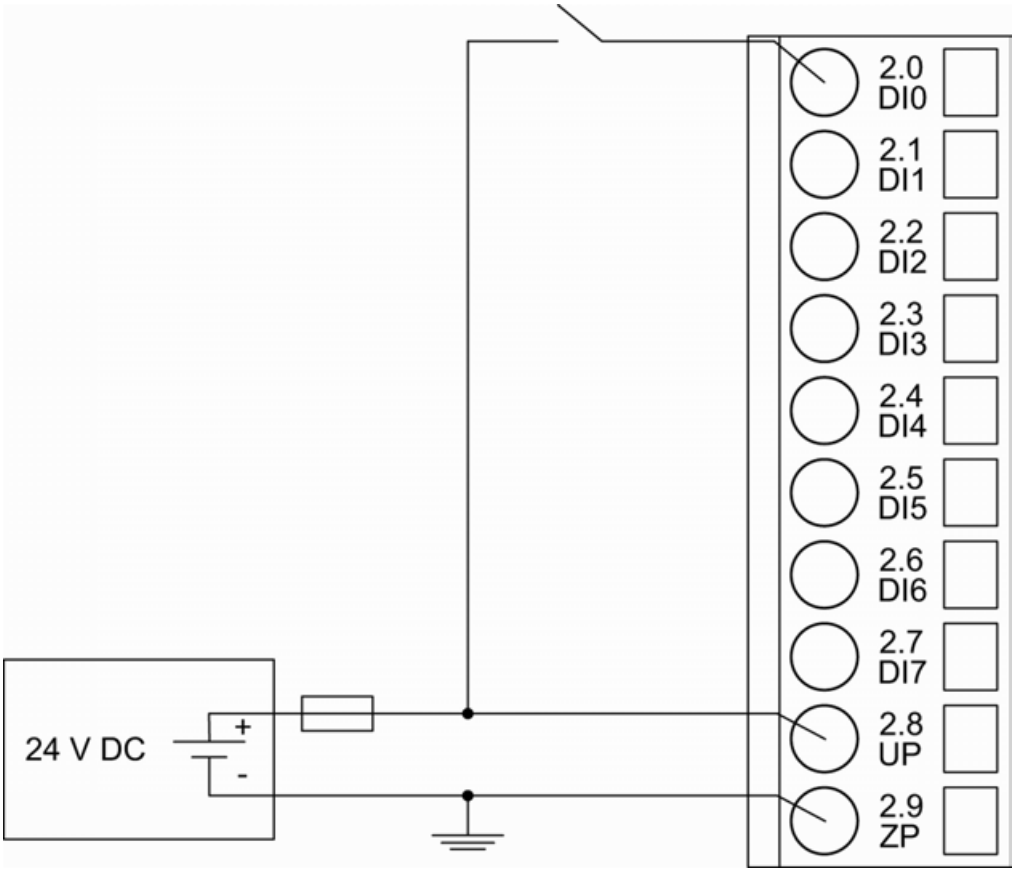
The following figures show the connection of the Ethernet bus module CI501-PNIO.



Further information is provided in the System Technology chapter *Chapter 1.6.5.3.2 “PROFINET communication interface module” on page 3629*.

Connection of the digital inputs

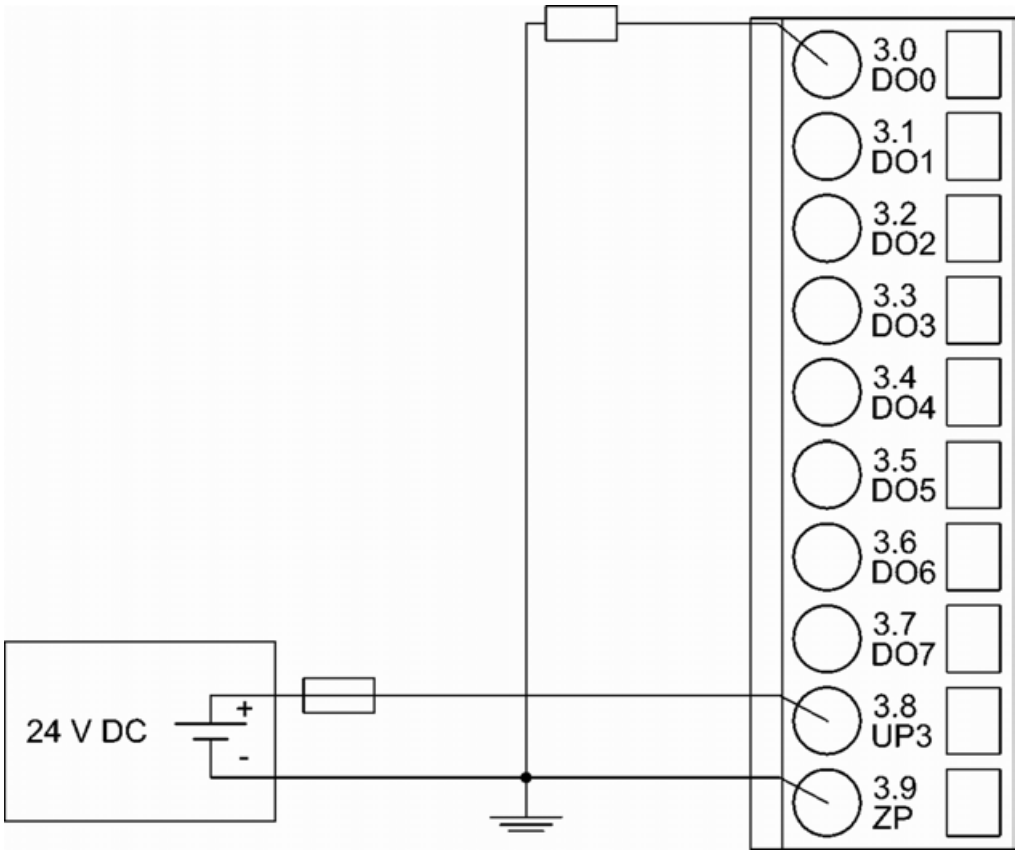
The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.



The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.5.2.8.2 “State LEDs” on page 3253*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

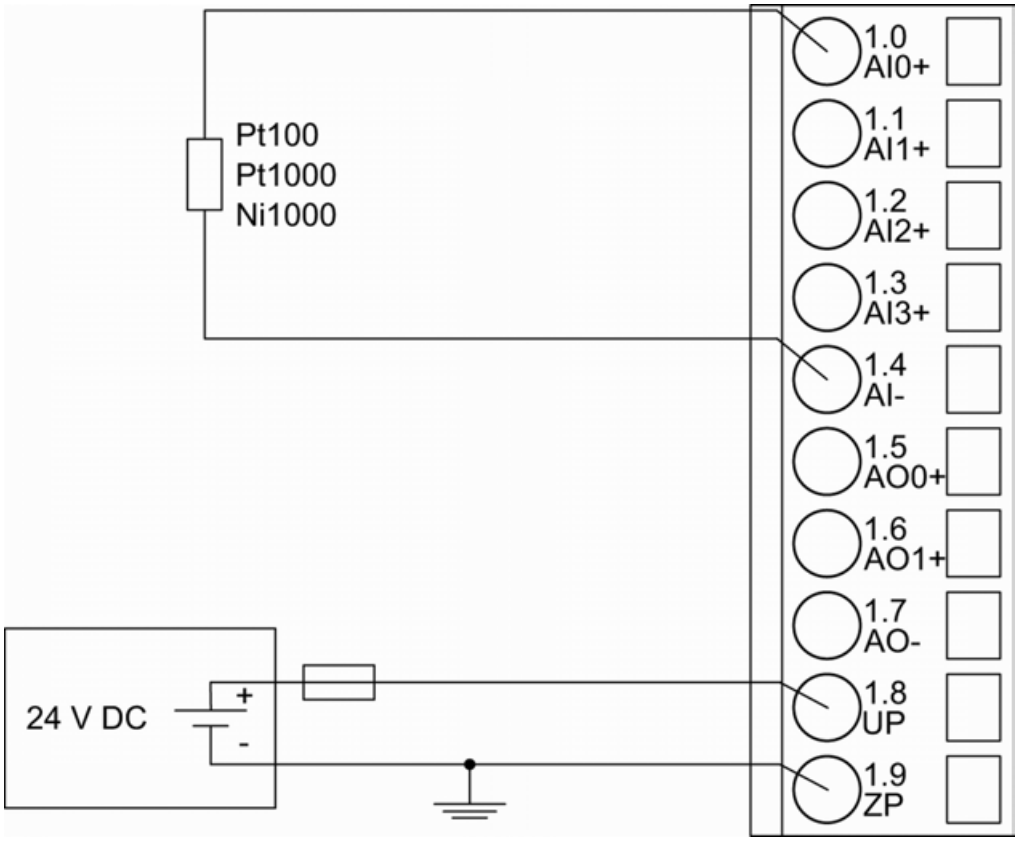


The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.5.2.8.2 “State LEDs” on page 3253*.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI501-PNIO provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

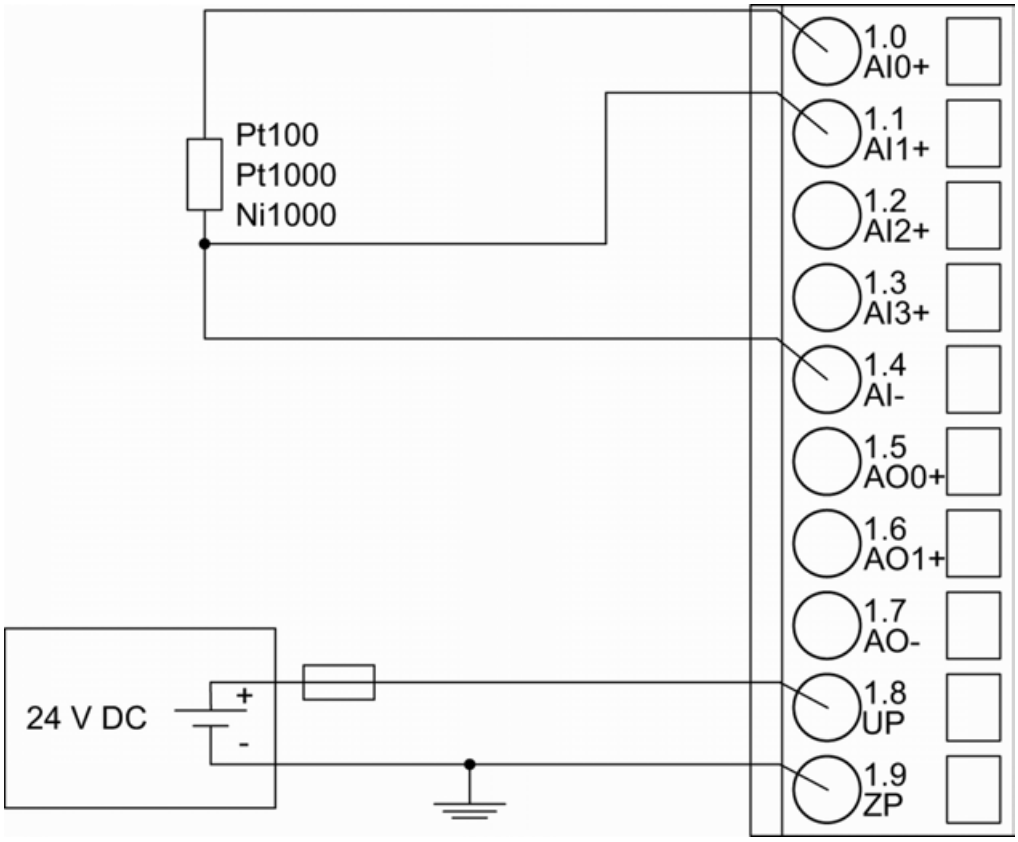
The module CI501-PNIO performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI501-PNIO provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.



With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

Pt100	-50 °C...+70 °C	3-wire configuration, 2 channels used
Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

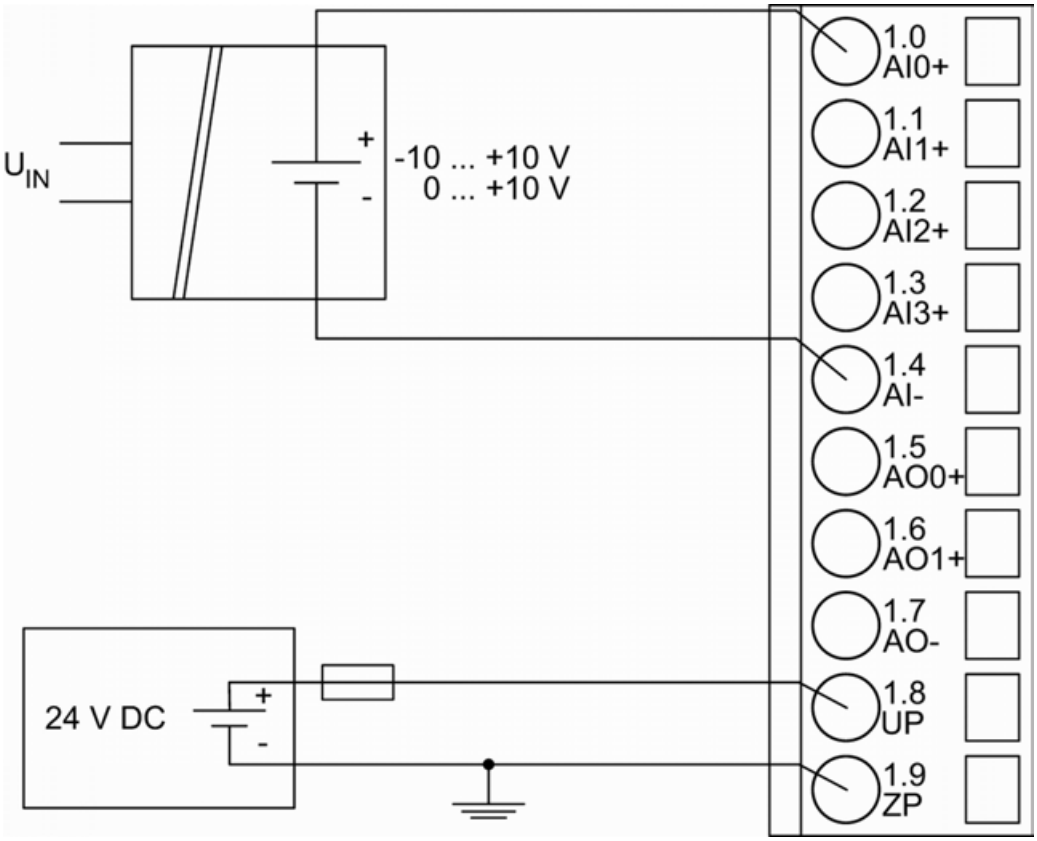
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

The module CI501-PNIO performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

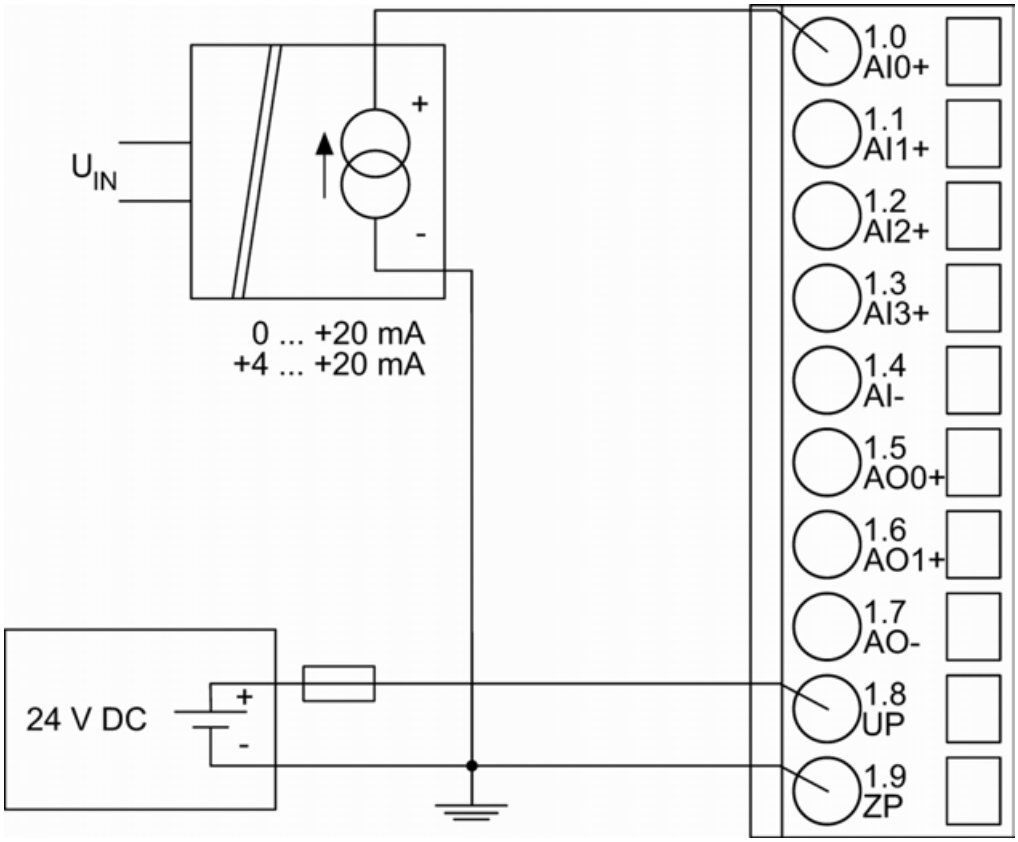
Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

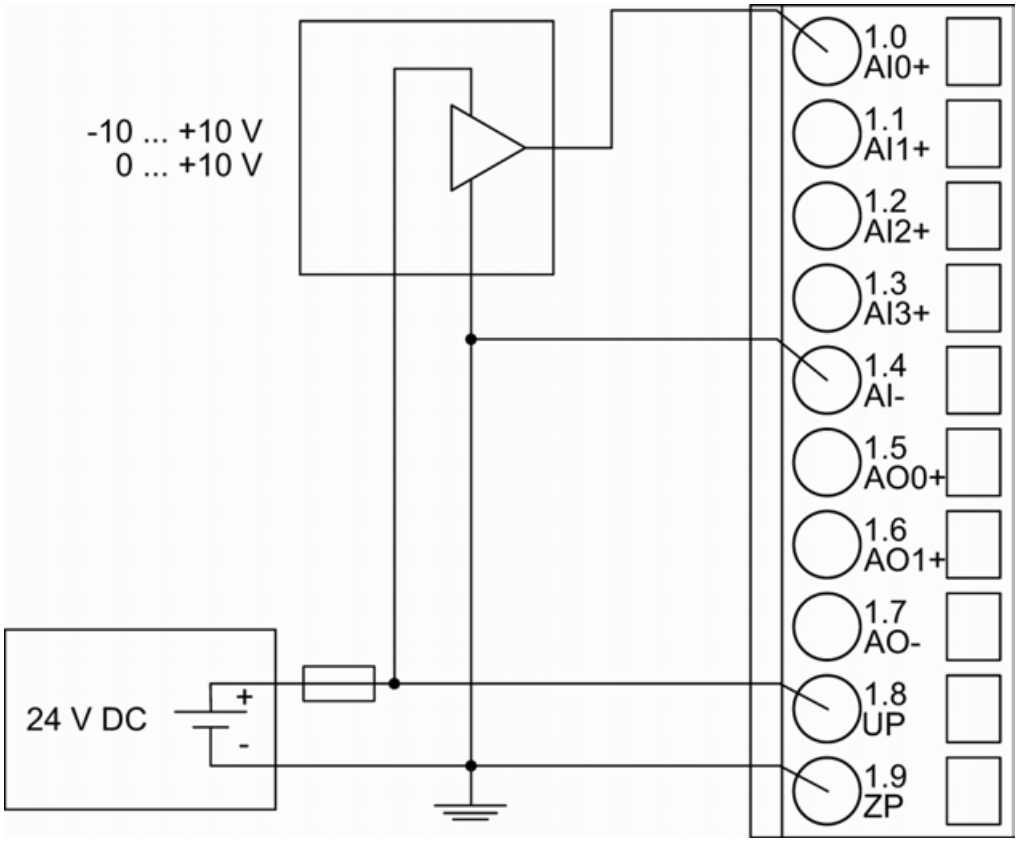
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4 mA...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V (also not with long cable lengths).

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242*:

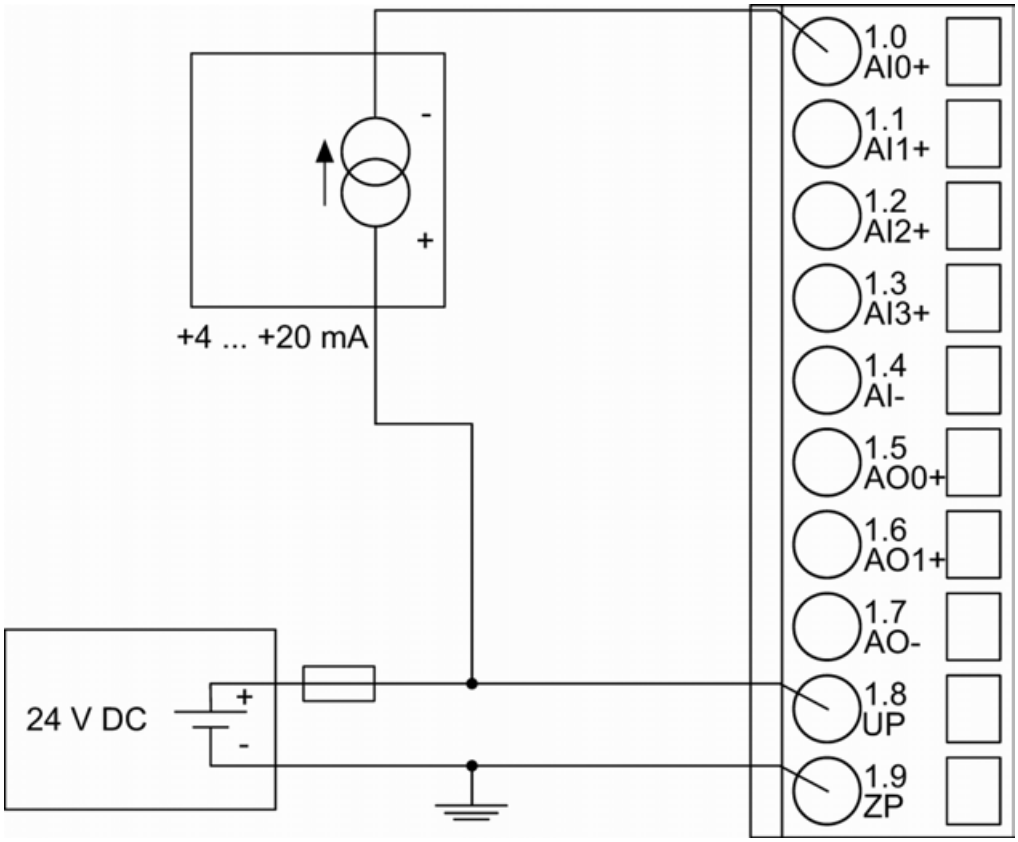
Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt zener diode in parallel to AIx+ and ZP.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4 mA...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

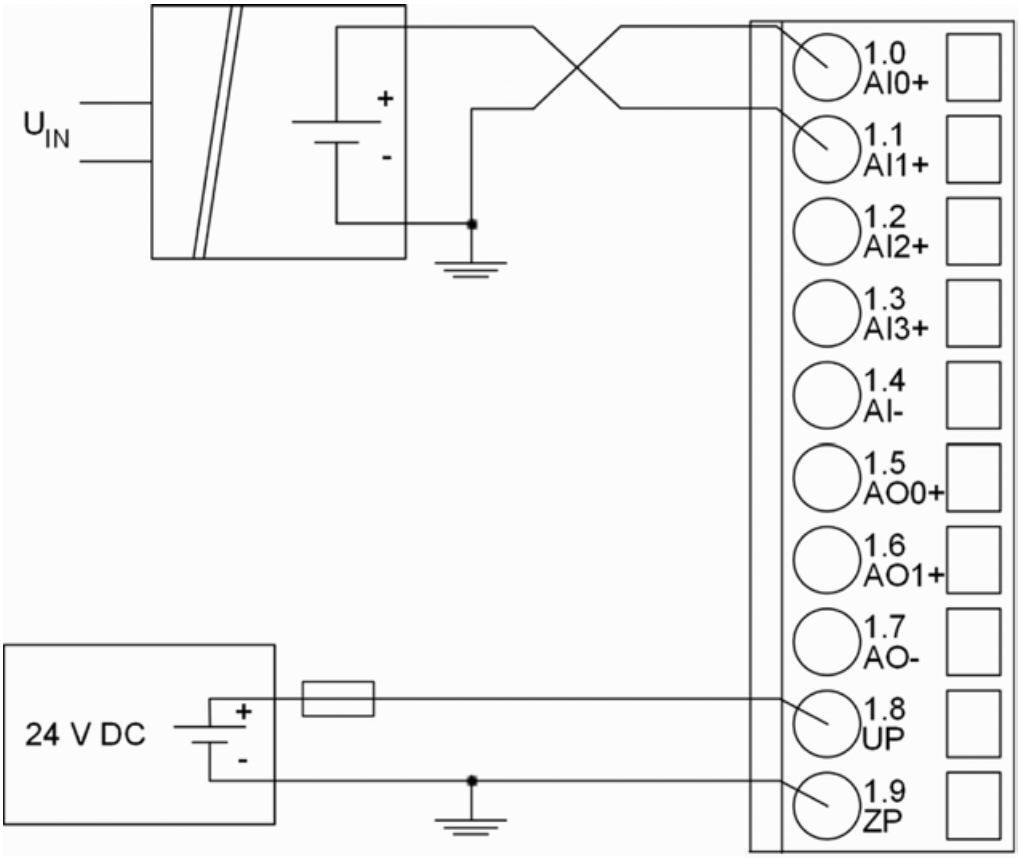
CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

Voltage	0 V...10 V	With differential inputs, 2 channels used
Voltage	-10 V...+10 V	With differential inputs, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

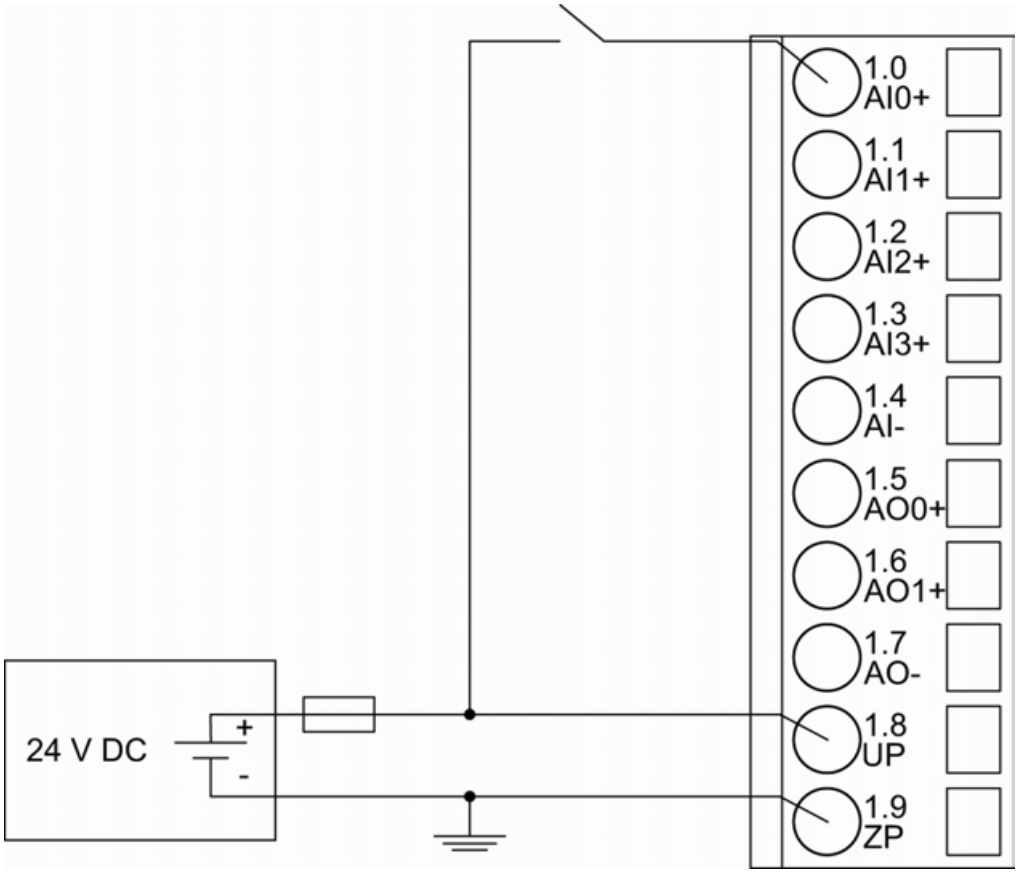


Fig. 252: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255* :

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

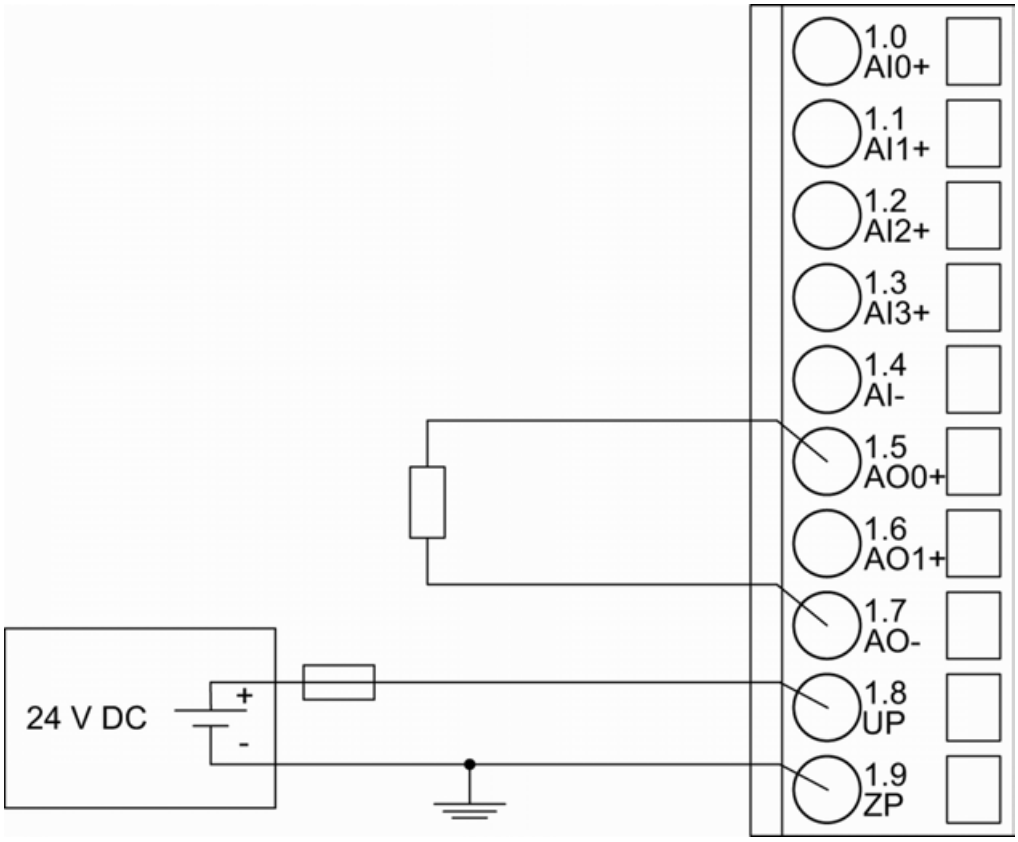


Fig. 253: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

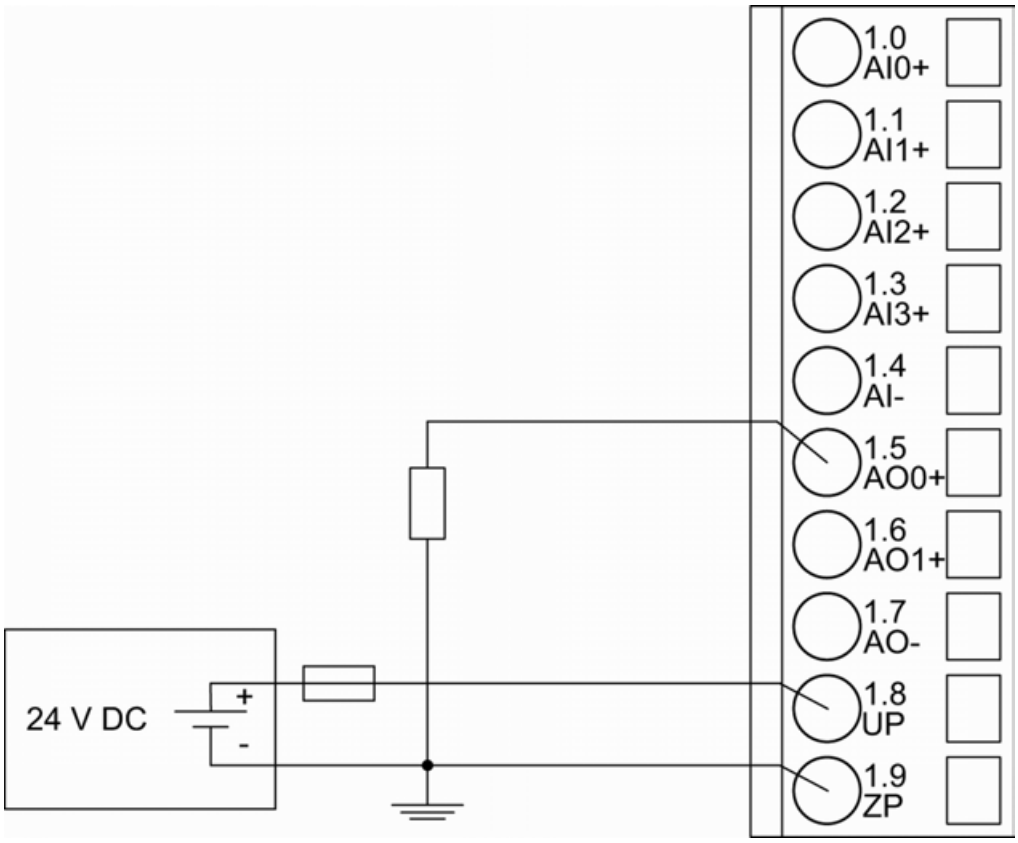


Fig. 254: Connection of analog output loads (current)

The following measuring ranges can be configured ↳ *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242* ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*:

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

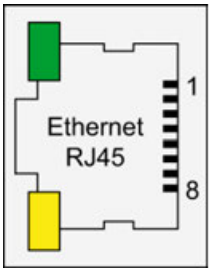
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*.

Unused analog outputs can be left open-circuited.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected

Interface	PIN	Signal	Description
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.*

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI501-PNIO stores some PROFINET configuration parameters (I/O device identifier, I/O device type and IP address configuration). No more configuration data is stored.

The analog/digital I/O channels are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.3.7.5.2.7 “Parameterization” on page 3242*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7000	WORD	7000
Parameter length	Internal	25	BYTE	25
Error LED / Failsafe function see table Error LED / Failsafe function *Table 572 "Error LED / Failsafe function" on page 3243*	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
Process cycle time [2])	1 ms process cycle time	1	BYTE	1 ms
	2 ms process cycle time	2		
	3 ms process cycle time	3		
	4 ms process cycle time	4		
	5 ms process cycle time	5		
	6 ms process cycle time	6		
	7 ms process cycle time	7		
	8 ms process cycle time	8		
	9 ms process cycle time	9		
	10 ms process cycle time	10		
	11 ms process cycle time	11		
	12 ms process cycle time	12		
	13 ms process cycle time	13		
	14 ms process cycle time	14		
	15 ms process cycle time	15		
	16 ms process cycle time	16		
Check supply	off	0	BYTE	1
	on	1		

Name	Value	Internal value	Internal value, type	Default
Input delay	8 ms	8 ms	BYTE	8 ms
Fast counter	0 : 10 [3])	0 : 10	BYTE	0
Detect short circuit at outputs	On	1	BYTE	On
Behavior digital outputs at comm. error	Off	0	BYTE	Off
Substitute value digital outputs	0	0..255	BYTE	0
Overvoltage behavior on output	Off	0	BYTE	Off
Behavior analog outputs atcomm. error	Off	0	BYTE	Off
I/O-Bus reset	Off	0	BYTE	Off
	On	1	BYTE	Off

Remarks:

[1])	With a faulty ID, the modules reports a "parameter error" and does not perform cyclic process data transmission.
[2])	As for device index C0 the parameter is no longer evaluated.
[3])	Counter operating modes, see description of the Fast counter ↬ *Chapter 1.6.3.6.1.2.9 “Fast counter” on page 2776*.

Table 572: Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameters Behaviour AO at comm. error and Behaviour DO at comm. error are only analyzed if the Failsafe-mode is ON.	

IO-BUS reset after PROFINET reconnection

IO-BUS reset after PROFINET reconnection controls the behavior of PROFINET CI modules in relation to connected I/O modules (both safety and non-safety I/O modules).

- IO-BUS reset after PROFINET reconnection = *"On"* resets and, thus, re-parameterizes all attached I/O modules. All internal I/O modules states are reset, including the related diagnosis information.

  Note that if the parameter is set to *"On"* then:
  - The bumpless re-start of non-safety I/O modules will not be supported. It means, for example, that non-safety output channels will go from fail-safe values to *"0"* values during the re-connection and re-parameterization time and after that go to new output values.
  - Safety I/O modules will be re-parameterized and re-started as newly started modules, which may not require their PROFIsafe reintegration, depending on safety CPU state, in the safety application.
- IO-BUS reset after PROFINET reconnection = *"Off"* will not reset all attached I/O modules. It will re-parameterize I/O modules only if parameter change is detected during the reconnection. All internal I/O modules states are not reset, including the related diagnosis information.

  Note that if the parameter is set to *"Off"* then:
  - The bumpless re-start of non-safety I/O modules is supported (if no parameters are changed). It means, for example, that non-safety output channels will not go from fail-safe values to *"0"* values during the re-connection and re-parameterization time, but directly from fail-safe values to new output values.
  - Safety I/O modules will not be re-parameterized (if no parameters are changed). Thus, they may continue their operation, which may require their PROFIsafe reintegration in the safety application on the safety CPU, e.g., if PROFIsafe watchdog time for this safety I/O module has expired. Any reintegration of such safety I/O modules will be not only application specific but also PROFIsafe specific and depend on the safety I/O handling in the safety application.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard	0	BYTE	0
	Reserved	255		
Behaviour AO at comm. error *)	Off	0	BYTE	0
	Last value	1		
	Last value 5 s	6		
	Last value 10 s	11		
	Substitute value	2		
	Substitute value 5 s	7		
	Substitute value 10 s	12		
*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Table Operating modes of the analog inputs ↳ *Table 573 “Channel configuration” on page 3245*	Table Operating modes of the analog inputs ↳ *Table 573 “Channel configuration” on page 3245*	BYTE	0
Input 0, Check channel	Table Channel montoring ↳ *Table 574 “Channel monitoring” on page 3246*	Table Channel montoring ↳ *Table 574 “Channel monitoring” on page 3246*	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, Channel configuration	Table Operating modes of the analog inputs ↳ *Table 573 “Channel configuration” on page 3245*	Table Operating modes of the analog inputs ↳ *Table 573 “Channel configuration” on page 3245*	BYTE	0
Input 3, Check channel	Table Channel montoring ↳ *Table 574 “Channel monitoring” on page 3246*	Table Channel montoring ↳ *Table 574 “Channel monitoring” on page 3246*	BYTE	0

Table 573: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0 V...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

Internal value	Operating modes of the analog inputs, individually configurable
19	3-wire Ni1000 -50 °C...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Table 574: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Table Operating modes of the analog outputs *Further information on page 3247*	Table Operating modes of the analog outputs *Further information on page 3247*	BYTE	0
Output 0, Check channel	Table Channel monitoring *Table 576 “Channel monitoring” on page 3247*	Table Channel monitoring *Table 576 “Channel monitoring” on page 3247*	BYTE	0
Output 0, Substitute value	Table Substitute value *Table 577 “Substitute value” on page 3247*	Table Substitute value *Table 577 “Substitute value” on page 3247*	WORD	0
Output 1, Channel configuration	Table Operating modes of the analog outputs *Further information on page 3247*	Table Operating modes of the analog outputs *Further information on page 3247*	BYTE	0
Output 1, Check channel	Table Channel monitoring *Table 576 “Channel monitoring” on page 3247*	Table Channel monitoring *Table 576 “Channel monitoring” on page 3247*	BYTE	0
Output 1, Substitute value	Table Substitute value *Table 577 “Substitute value” on page 3247*	Table Substitute value *Table 577 “Substitute value” on page 3247*	WORD	0

Table 575: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Table 576: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 577: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	On 0x01
[1]) The parameters Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON. [2]) The state "externally voltage detected" appears, if the output of a channel DC0...DC7 should be switched on while an externally voltage is connected ↯ *Chapter 1.6.3.7.5.2.3 “Connections” on page 3226*. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".				

Diagnosis and state LEDs

Structure of the diagnosis block via PNIO_DEV_ALARM function block

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI501-PNIO (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master

Byte Number	Description	Possible Values
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check master

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	No process voltage UP	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket	Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap ter-minal unit [9])	Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug pro-jected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])		Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found		Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])		Restart, if error persists replace terminal unit
4	-	31	31	31	46	Voltage feedback on activated digital outputs DO0...DO7 on UP3 [4])		Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module		Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low		Check process supply voltage
4	1...6	255	2	0	45	The connected Communication Module has no connection to the network		Check cabeling
4	-	31	31	31	45	No process voltage UP3		Check process supply voltage

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Externally voltage detected at digital output DO0...DO7 [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at digital output [7])	Check terminals
Channel error analog							
4	-	31	1	0...3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0...3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0...3	47	Short circuit at an analog input	Check terminals
4	-	31	3	0...1	4	Analog value overflow at an analog output	Check output value
4	-	31	3	0...1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the communication module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI501-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself; 1...10 = Expansion module
[3])	With "Module" the following allocation applies: 31 = Module itself Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DO0...DO7 cause that other digital outputs are supplied through that voltage ↳ *Chapter 1.6.3.7.5.2.3 "Connections" on page 3226*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0...DO7 has overrun the process supply voltage UP3 ↳ *Chapter 1.6.3.7.5.2.3 "Connections" on page 3226*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 100 ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 578: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	---

LED	Color	OFF	ON	Flashing
	Red	---	---	Device is not configured
STA2 ETH (System LED "SF")	Green	---	---	Got identification request from I/O controller
	Red	No system error	System error (collective error)	---
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 579: States of the 27 process LEDs

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 to DO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Under-flow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...+70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	80.0 °C	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
Normal range		400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	800 : 701	0320 : 02BD

Range	Pt100 / Pt1000 -50...+70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
		0.0 °C	0.0 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
		-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	0	0000
Measured value too low	< -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-1 : -500	FFFF : FE0C
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-501 : -600	FE0B : FDA8

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	> 11.7589 V	> 23.5178 mA	> 22.8142 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	< -11.7589 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of analog inputs		4
Number of analog outputs		2
Input data length		2 bytes
Output data length		2 bytes
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the I/O device identifier		With 2 rotary switches at the front side of the module
Diagnose		See Diagnosis and Displays *Chapter 1.6.3.7.5.2.8 “Diagnosis and state LEDs” on page 3248*
Operation and error displays		32 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)

Parameter	Value
Extended ambient temperature (XC version)	>60 °C on request
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms

Parameter	Value
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Technical data of the digital inputs

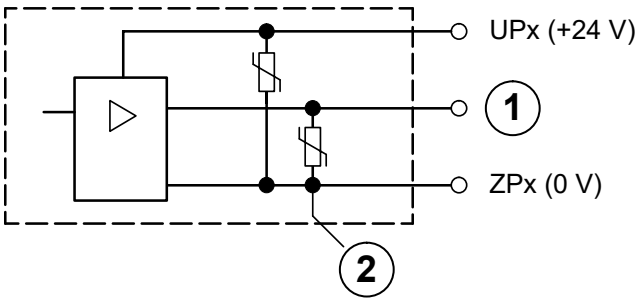
Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels

Parameter		Value
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 1.0 to1.3
Reference potential for AI0+ to AI3+		Terminal 1.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9 and 3.9 for current measurement
Input type		
	Unipolar	Voltage 0 V... 10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V... +10 V
Galvanic isolation		Against Ethernet network
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		Tables Input ranges voltage, current and digital input and Input range resistance temperature detector ↳ *Chapter 1.6.3.7.5.2.9.1 “Input ranges voltage, current and digital input” on page 3255*
Unused inputs		Are configured as "unused" (default value)
Overvoltage protection		Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for the inputs		Terminals 1.9, 2.9 and 3.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 1.5...1.6
Reference potential for AO0+ to AO1+		Terminal 1.7 (AO-) for voltage output terminal 1.9, 2.9 and 3.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %

Parameter	Value
Relationship between input signal and hex code	Table Output ranges voltage and current ↳ *Chapter 1.6.3.7.5.2.9.3 “Output ranges voltage and current” on page 3256*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI0), 2.1 (DI1)
Used outputs	Terminal 3.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

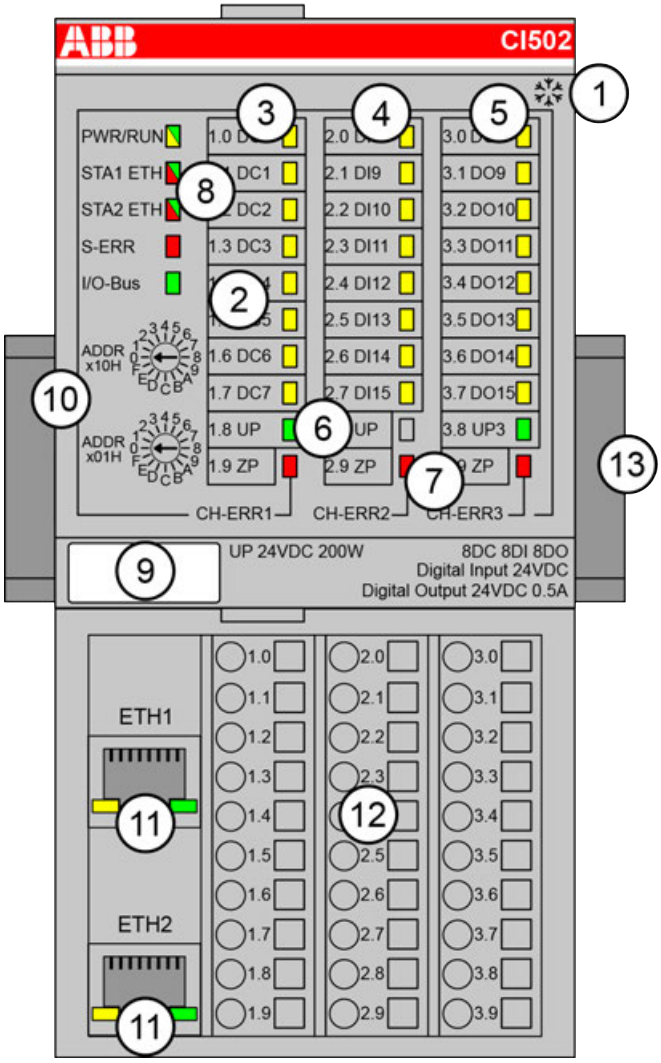
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 220 600 R0001	CI501-PNIO (V3), PROFINET communication interface module, 8 DI, 8 DO, 4 AI and 2 AO	Active
1SAP 420 600 R0001	CI501-PNIO-XC (V3), PROFINET communication interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI502-PNIO

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital configurable inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the I/O device identifier
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The PROFINET communication interface module CI502-PNIO is used as communication interface module in PROFINET networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

The CI502 communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs
- 8 digital inputs: 24 V DC
- 8 digital outputs: 24 V DC, 0.5 A max.

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

Parameter	Value
Interface	Ethernet
Protocol	PROFINET IO RT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the IO device identifier for configuration purposes (00h to FFh)
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507-ETH or TU508-ETH *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*

Connections

The Ethernet communication interface module CI502-PNIO is plugged on the I/O terminal unit TU507-ETH *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549* or TU508-ETH *Chapter 1.6.3.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 2549*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.3.8.2.6 “TA526 - Wall mounting accessory” on page 3329*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.4.6 “AC500 (Standard)” on page 3398.*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V.

The assignment of the other terminals:

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to digital outputs!

This ist not intended usage.

Reason: Externally voltages at one or more terminals DC0..DC7 or DO0..DO7 may cause that other digital outputs are supplied through that voltage instead of voltage UP3 (reverse voltage).

This is also possible, if DC channels are used as inputs. For this, the source for the input signals should be the impressed UP3 of the device.

This limitation does not apply for the input channels DI0..DI7.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO0...DO7 and DC0...DC7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	DC0	Signal of the configurable digital input/output DC0
1.1	DC1	Signal of the configurable digital input/output DC1
1.2	DC2	Signal of the configurable digital input/output DC2
1.3	DC3	Signal of the configurable digital input/output DC3
1.4	DC4	Signal of the configurable digital input/output DC4
1.5	DC5	Signal of the configurable digital input/output DC5
1.6	DC6	Signal of the configurable digital input/output DC6
1.7	DC7	Signal of the configurable digital input/output DC7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)

Terminal	Signal	Description
2.0	DI8	Signal of the digital input DI8
2.1	DI9	Signal of the digital input DI9
2.2	DI10	Signal of the digital input DI10
2.3	DI11	Signal of the digital input DI11
2.4	DI12	Signal of the digital input DI12
2.5	DI13	Signal of the digital input DI13
2.6	DI14	Signal of the digital input DI14
2.7	DI15	Signal of the digital input DI15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO8	Signal of the digital output DO8
3.1	DO9	Signal of the digital output DO9
3.2	DO10	Signal of the digital output DO10
3.3	DO11	Signal of the digital output DO11
3.4	DO12	Signal of the digital output DO12
3.5	DO13	Signal of the digital output DO13
3.6	DO14	Signal of the digital output DO14
3.7	DO15	Signal of the digital output DO15
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

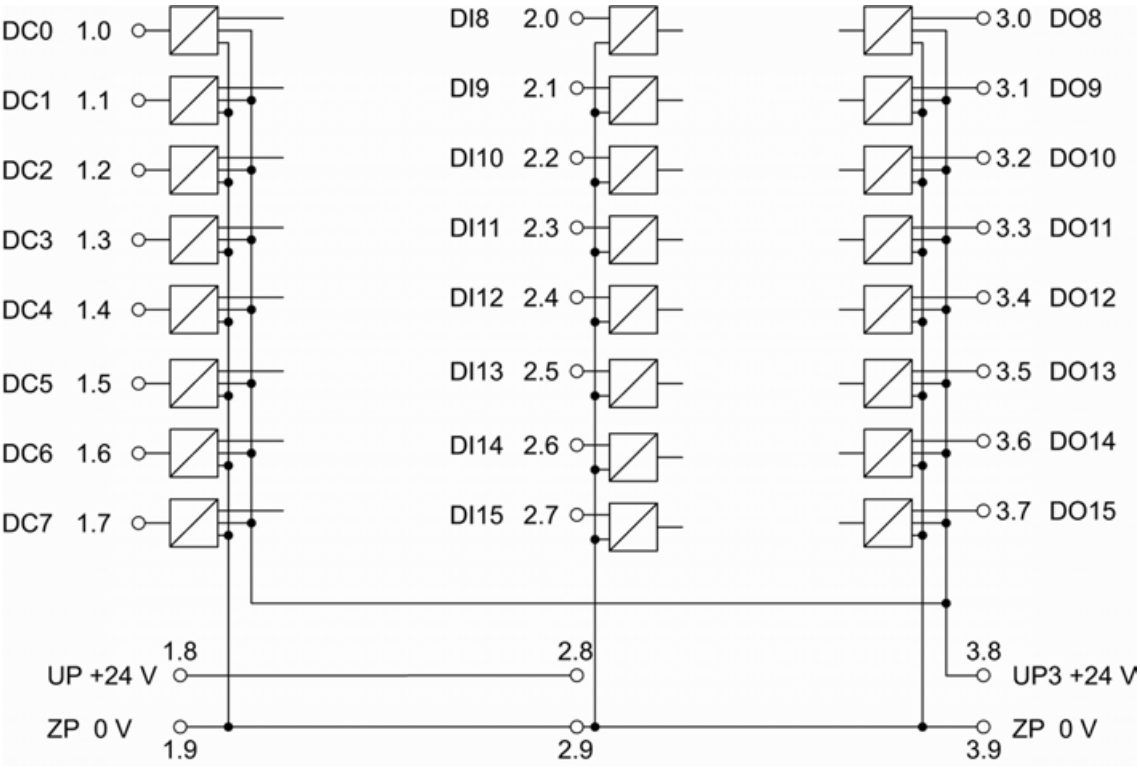
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

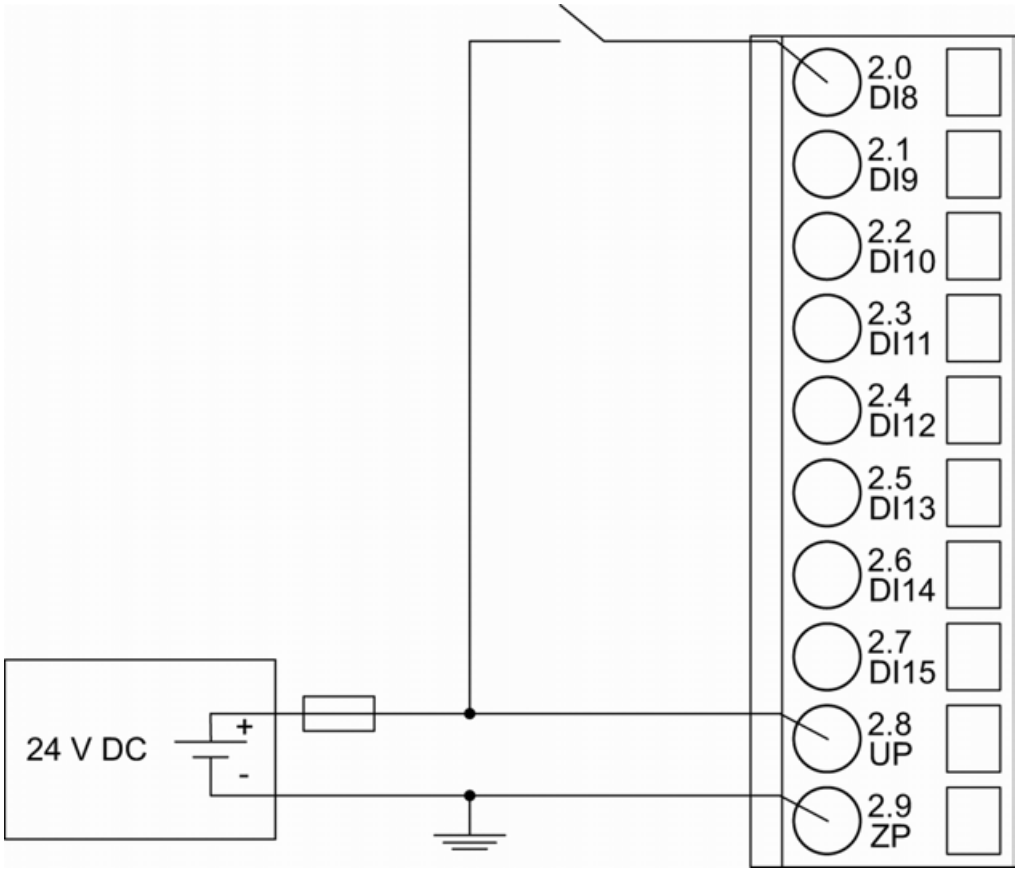
The following figure shows the connection of the Ethernet communication interface module CI502-PNIO.



Further information is provided in the System Technology chapter PROFINET *Chapter 1.6.5.3.2 “PROFINET communication interface module” on page 3629*.

Connection of the Digital inputs

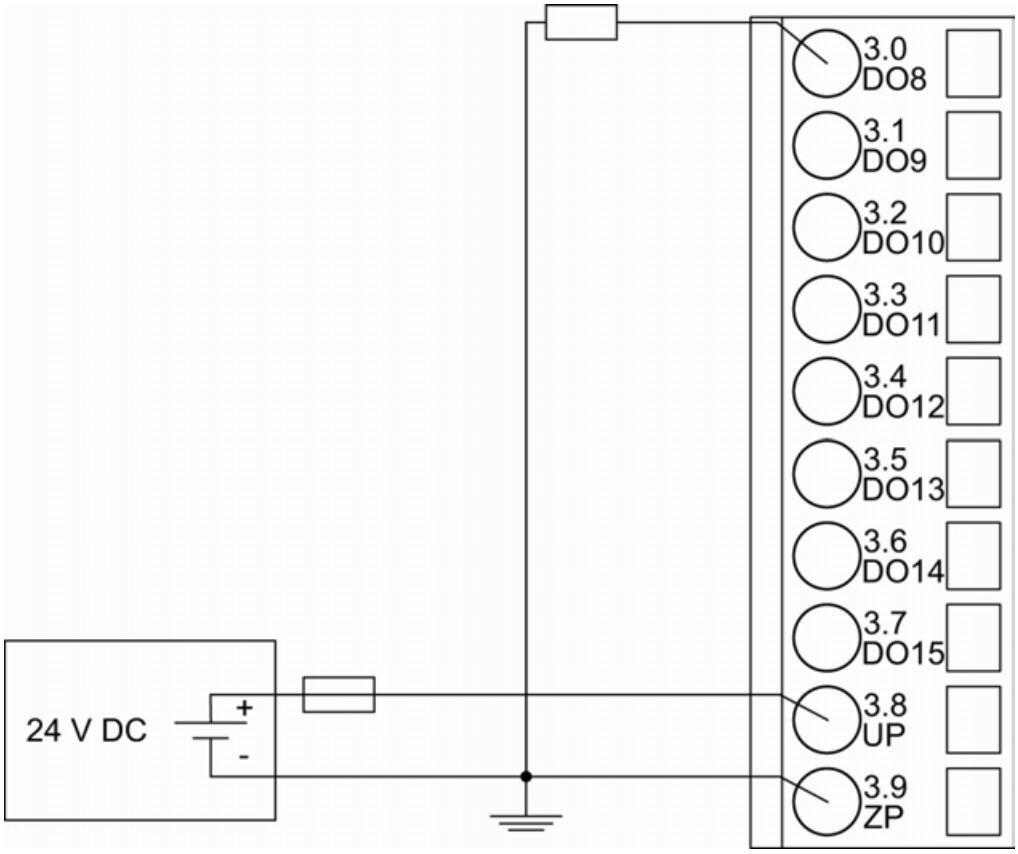
The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.



The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.5.3.8.1 “State LEDs” on page 3280*.

Connection of the Digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

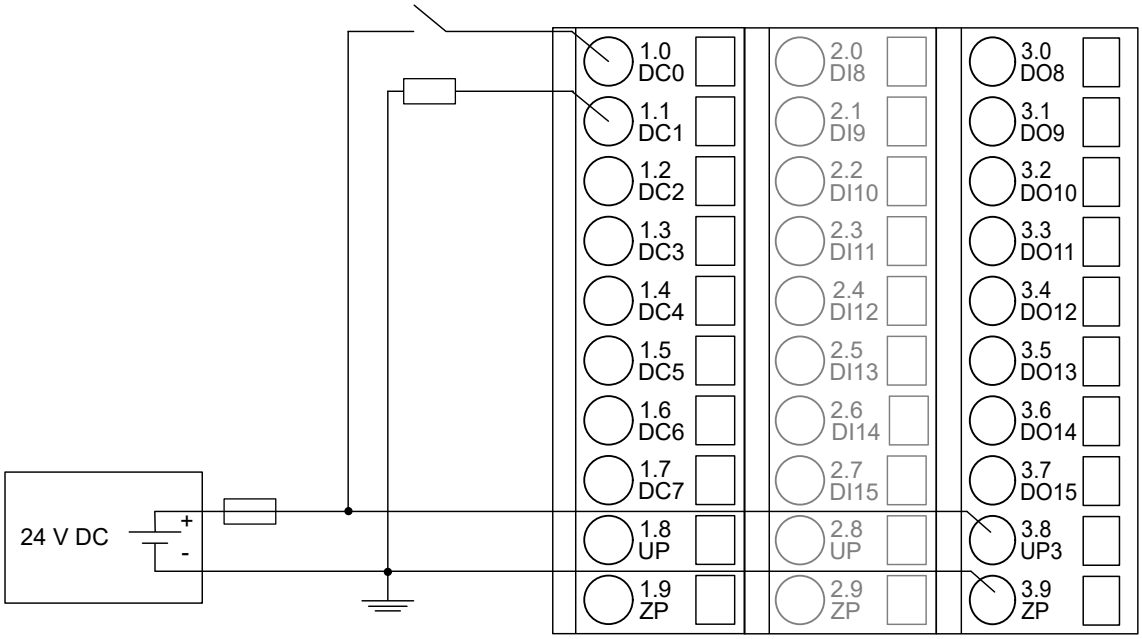


The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.5.3.8.1 “State LEDs” on page 3280*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

CAUTION!
If a DC channel is used as input, the source for the input signals should be the impressed UP3 of the device *Chapter 1.6.3.7.5.3.3 “Connections” on page 3265*.

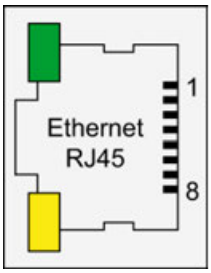


The meaning of the LEDs is described in Displays *Chapter 1.6.3.7.5.3.8.1 “State LEDs” on page 3280*.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.4.6.4.7 “Ethernet connection details” on page 3424.*

Internal data exchange

Parameter	Value
Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI502-PNIO stores some PROFINET configuration parameters (I/O device identifier, I/O device type and IP address configuration). No more configuration data is stored.

The digital I/O channels are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.3.7.5.3.7 “Parameterization” on page 3272*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7005	WORD	7005
Parameter length	Internal	8	BYTE	8

Name	Value	Internal value	Internal value, type	Default
Error LED / Failsafe function (Table Error LED / Failsafe function *Further information on page 3272*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
Process cycle time	1 ms process cycle time	1	BYTE	1 ms
	2 ms process cycle time	2		
	3 ms process cycle time	3		
	4 ms process cycle time	4		
	5 ms process cycle time	5		
	6 ms process cycle time	6		
	7 ms process cycle time	7		
	8 ms process cycle time	8		
	9 ms process cycle time	9		
	10 ms process cycle time	10		
	11 ms process cycle time	11		
	12 ms process cycle time	12		
	13 ms process cycle time	13		
	14 ms process cycle time	14		
	15 ms process cycle time	15		
	16 ms process cycle time	16		
Check supply	Off On	0 1	BYTE	1
Fast counter	0 : 10 [2])	0 : 10	BYTE	0
I/O-Bus reset	Off	0	BYTE	Off

Name	Value	Internal value	Internal value, type	Default
	On	1	BYTE	Off
[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.				
[2]) Counter operating modes *Chapter 1.6.3.6.1.2.9 "Fast counter" on page 2776*				

Table 580: Table Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameter Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.	

IO-BUS reset after PROFINET reconnection

IO-BUS reset after PROFINET reconnection controls the behavior of PROFINET CI modules in relation to connected I/O modules (both safety and non-safety I/O modules).

- IO-BUS reset after PROFINET reconnection = *"On"* resets and, thus, re-parameterizes all attached I/O modules. All internal I/O modules states are reset, including the related diagnosis information.

  Note that if the parameter is set to *"On"* then:
  - The bumpless re-start of non-safety I/O modules will not be supported. It means, for example, that non-safety output channels will go from fail-safe values to *"0"* values during the re-connection and re-parameterization time and after that go to new output values.
  - Safety I/O modules will be re-parameterized and re-started as newly started modules, which may not require their PROFIsafe reintegration, depending on safety CPU state, in the safety application.
- IO-BUS reset after PROFINET reconnection = *"Off"* will not reset all attached I/O modules. It will re-parameterize I/O modules only if parameter change is detected during the reconnection. All internal I/O modules states are not reset, including the related diagnosis information.

  Note that if the parameter is set to *"Off"* then:
  - The bumpless re-start of non-safety I/O modules is supported (if no parameters are changed). It means, for example, that non-safety output channels will not go from fail-safe values to *"0"* values during the re-connection and re-parameterization time, but directly from fail-safe values to new output values.
  - Safety I/O modules will not be re-parameterized (if no parameters are changed). Thus, they may continue their operation, which may require their PROFIsafe reintegration in the safety application on the safety CPU, e.g., if PROFIsafe watchdog time for this safety I/O module has expired. Any reintegration of such safety I/O modules will be not only application specific but also PROFIsafe specific and depend on the safety I/O handling in the safety application.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...65535	0000h...FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behaviour DO at comm. error is apply to DC and DO channels and only analyzed if the Failsafe-mode is ON.
[2])	The state "externally voltage detected" appears, if the output of a channel DC0...DC7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears, if externally voltage at digital outputs DC0...DC7 and accordingly DO0...DO7 has exceeded the process supply voltage UP3 ↯ *Chapter 1.6.3.7.5.3.3 “Connections” on page 3265* (see description in section). The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters on "OFF". This parameter should only be disabled in exceptional cases for voltage overflow may produce reverse voltage.

Diagnosis

Structure of the Diagnosis Block via PNIO_DEV_ALARM function block.

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI502-PNIO (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
3	-	31	31	31	40	Different hard-/firm-ware versions in the module	
3	-	31	31	31	43	Internal error in the module	
3	-	31	31	31	36	Internal data exchange failure	
3	-	31	31	31	9	Overflow diagnosis buffer	Restart
3	-	31	31	31	26	Parameter error	Check master
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O device	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O device type on socket	Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap ter-minal unit [9])	Plug I/O module, replace I/O module

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug pro-jected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap ter-minal unit [9])	Restart, if error persists replace terminal unit
4	1...6	255	2	0	45	The connected Communication Module has no con-nection to the net-work	Check cabeling
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	46	Reverse voltage from digital outputs DO0..DO7 to UP3 [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow at outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	8...15	46	Externally voltage detected at digital output DO0..DO7 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0..DC7 [6])	Check terminals
4	-	31	4	0...7	47	Short circuit at digital output DC0..DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO0..DO7[7])	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the Communication Module;14 = I/O-Bus; 31 = Module itself The identifier is not contained in the CI502-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1..10 = Expansion module
[3])	With "Module" the following allocation applies dependent of the master: Module error: 31 = Module itself Channel error: Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DC0...DC7 oder DO0...DO7 cause that other digital outputs are supplied through that voltage (voltage feedback, see description in 'Connections' *Chapter 1.6.3.7.5.3.3 "Connections" on page 3265*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage at digital outputs DC0...DC7 and accordingly DO0...DO7 has exceeded the process supply voltage UP3 *Chapter 1.6.3.7.5.3.3 "Connections" on page 3265*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DC0...DC7 or DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 2000 ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 581: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with IO Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System-LED "BF")	Green	---	Device configured, cyclic data exchange running	---

LED	Color	OFF	ON	Flashing
	Red	---	---	Device is not configured
STA2 ETH (System LED "SF")	Green	---	---	Got identification request from I/O controller
	Red	No system error	System error (collective error)	---
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 582: States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.4.6.1 “System data AC500” on page 3398* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of configurable digital inputs/outputs		8
Input data length		12 bytes
Output data length		20 bytes
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the I/O device identifier		With 2 rotary switches at the front side of the module
Diagnosis		See Diagnosis and Displays *Chapter 1.6.3.7.5.3.8 “Diagnosis” on page 3275*
Operation and error displays		34 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Extended ambient temperature (XC version)		> 60 °C on request
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 2.0 to 2.7

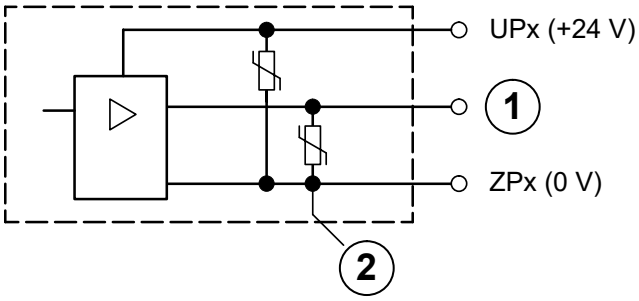
Parameter		Value
Reference potential for all inputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request

Parameter		Value
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC07	Terminals 1.0...1.7
If the channels are used as outputs		
	Channels DC0...DC07	Terminals 1.0...1.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the Ethernet network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

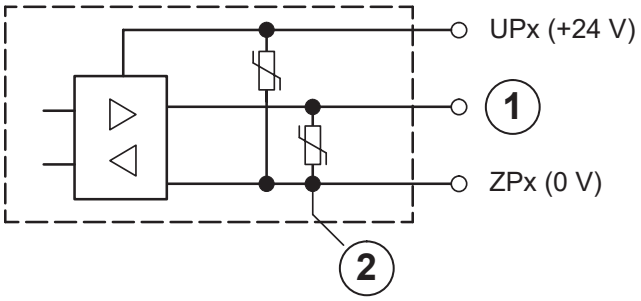
*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DC0 to DC7	Terminals 1.0 to 1.7
Reference potential for all outputs	Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1	UP3 (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	

Parameter		Value
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI8),Terminal 2.1 (DI9)
Used outputs	Terminal 3.0 (DO8)
Counting frequency	Depending on operation mode: Mode 1- 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

↳ *Chapter 1.6.5.1.12 “Fast counters” on page 3570*

Ordering data

Active	Active	Product life cycle phase *)
1SAP 220 700 R0001	CI502-PNIO (V3), PROFINET communication interface module, 8 DI, 8 DO and 8 DC	Active
1SAP 420 700 R0001	CI502-PNIO-XC (V3), PROFINET communication interface module, 8 DI, 8 DO and 8 DC, XC version	Active

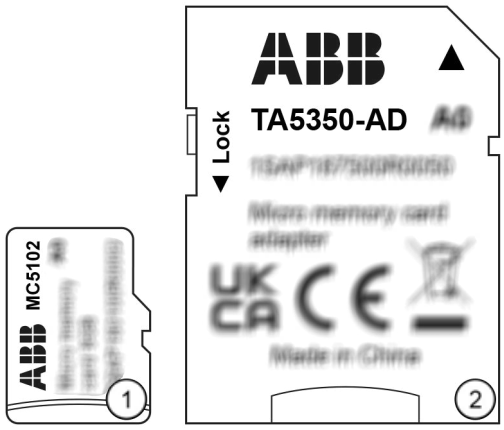
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.8 Accessories

1.6.3.8.1 AC500-eCo

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.

The TA5350-AD micro memory card adapter has a write protect switch.

In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

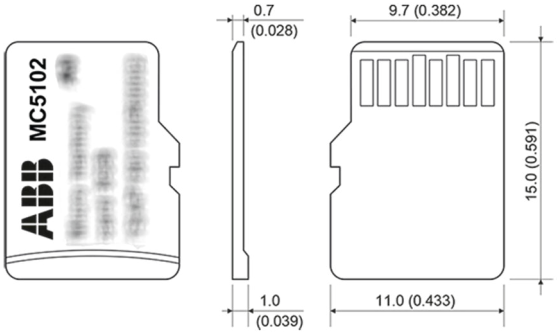
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

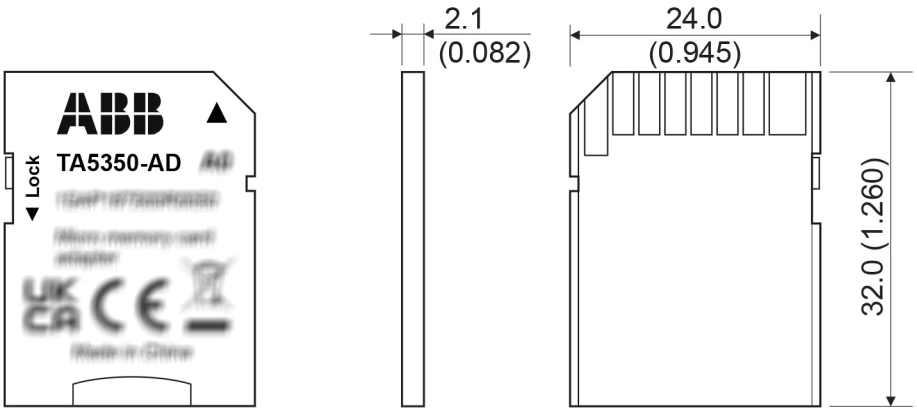
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V3

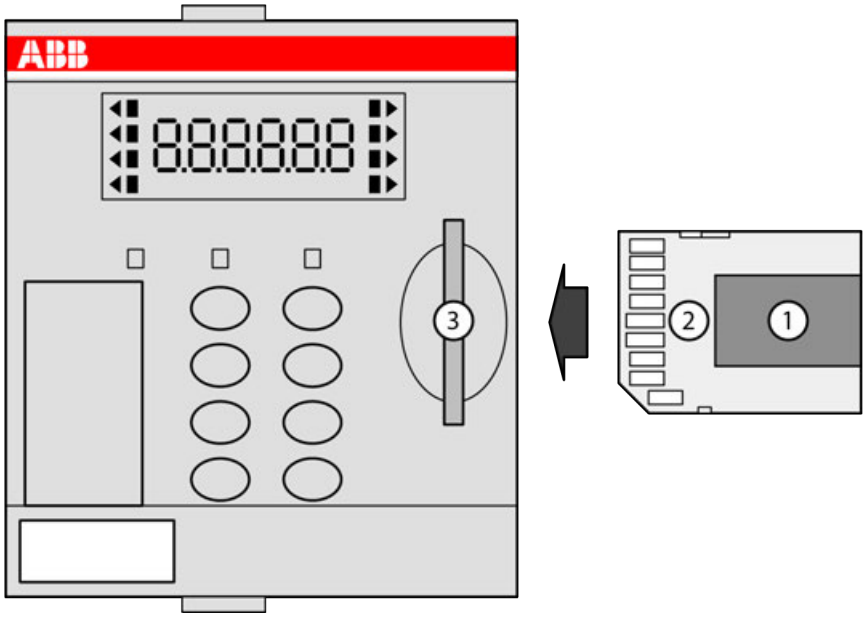
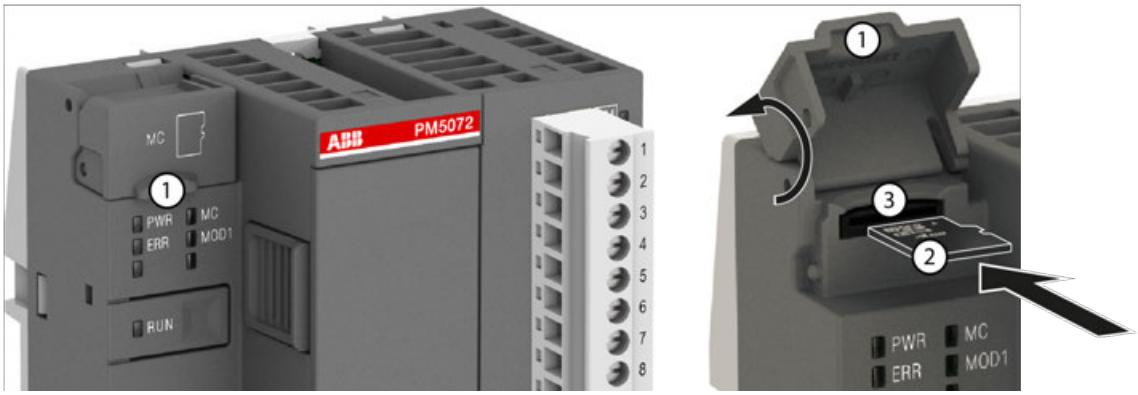


Fig. 255: Insert micro memory card into PM56xx

1 Micro memory card
2 TA5350-AD micro memory card adapter
3 Memory card slot

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Carefully insert the micro memory card into the micro memory card slot as far as it will go. Observe orientation of card.
3. Close the micro memory card slot cover by turning it downwards.

Remove the micro memory card

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

AC500 V3: Remove the micro memory card with micro memory card adapter only when no black square (■) is shown next to MC in the display.

AC500-eCo V3: Remove the micro memory card only when the MC LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

AC500 V3

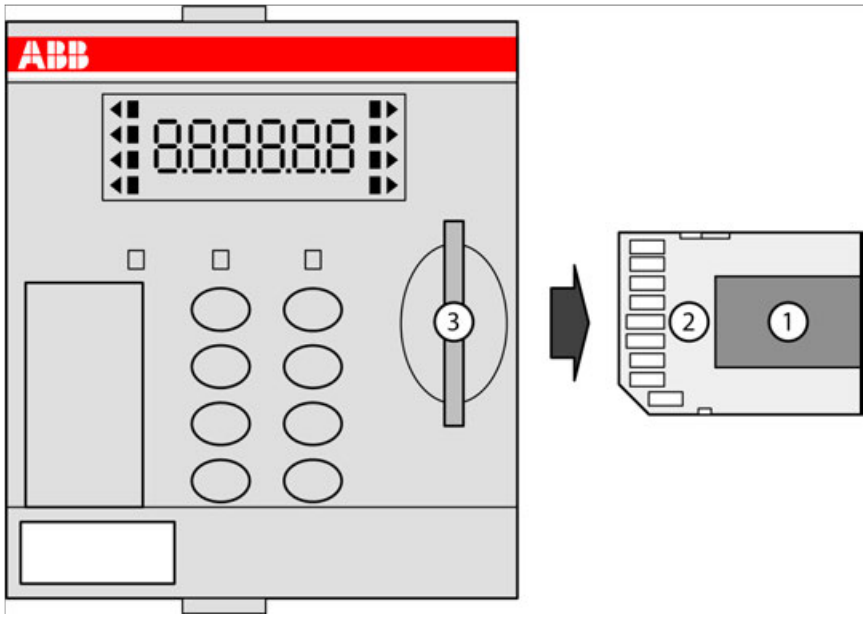
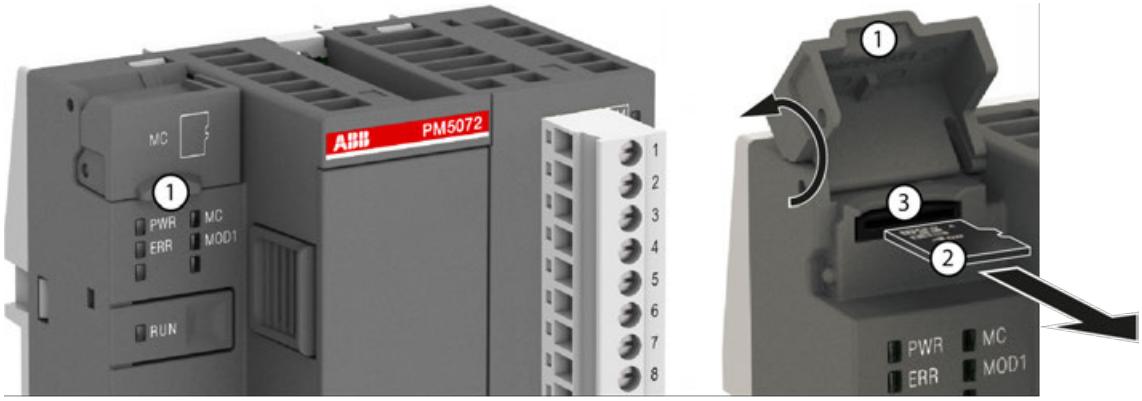


Fig. 256: Remove micro memory card from PM56xx

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Micro memory card can be removed from the micro memory card slot by gripping and pulling with two fingers.
3. Close the micro memory card slot cover by turning it downwards.

Technical data

Parameter	Value
Memory capacity	8 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter ↳ *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA52xx(-x) - Terminal block sets

Intended purpose

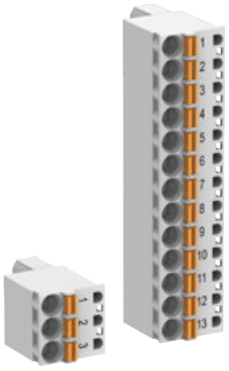
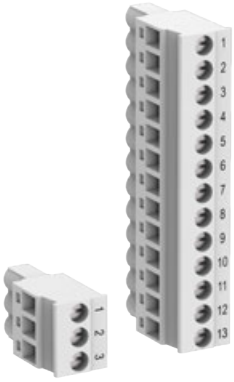
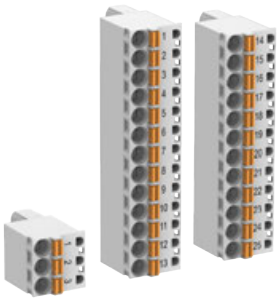
Removable terminal blocks are used for power supply and for I/O connectors on AC500-eCo V3 processor modules PM50x2.

For option boards there are different removable terminal blocks in spring version.

For the AC500-eCo V3 **Basic CPUs** a 3-pin terminal block for power supply and a 13-pin terminal block for I/O connectors are used.

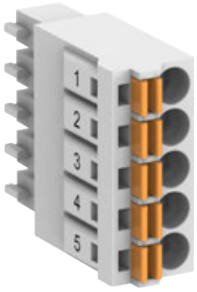
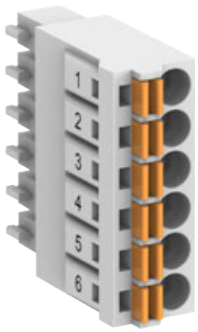
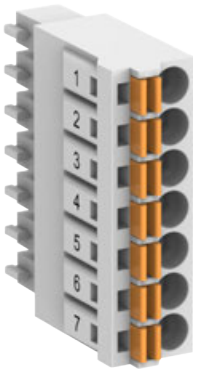
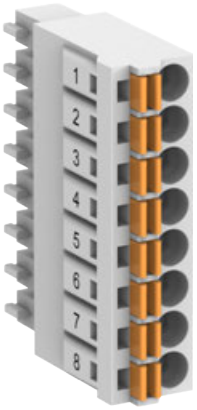
For the AC500-eCo V3 **Standard CPUs** and **Pro CPUs** a 3-pin terminal block for power supply, a 13-pin terminal block and a 12-pin terminal block for I/O connectors are used.

For all CPUs there is a screw and a spring variant available.

Basic CPU		**Standard and Pro CPUs**	
Spring type **TA5211-TSPF-B**	**Screw type** **TA5211-TSCL-B**	**Spring type** **TA5212-TSPF**	**Screw type** **TA5212-TSCL**
			

Various removable spring-type terminal blocks are available for option boards.

The following spare parts are available (depending on the number of pins).

Spring type			
TA5220-SPF5	**TA5220-SPF6**	**TA5220-SPF7**	**TA5220-SPF8**
			

CAUTION!

Risk of injury and damaging the product!

Improper installation and maintenance may result in injury and can damage the product!

- Installation and maintenance have to be performed according to the technical rules, codes and relevant standards, e.g. EN 60204-1.
- Read product documentation carefully before wiring. Improper wiring or wrong terminal block from other devices can damage the product!
- Only by qualified personnel.

CAUTION!

Risk of injury and damaging the processor module when using unapproved terminal blocks!

Only use terminal blocks approved by ABB to avoid injury and damage to the processor module.

Terminal block set for PM50x2

Processor modules PM50x2 CPU are not delivered with terminal blocks.

Screw type terminal block set:

- *TA5211-TSCL-B (1SAP187400R0001) for PM5012-x-ETH*
- *TA5212-TSCL (1SAP187400R0004) for PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH(W)*

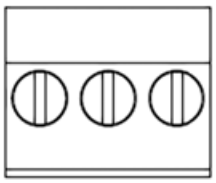
Spring type terminal block set:

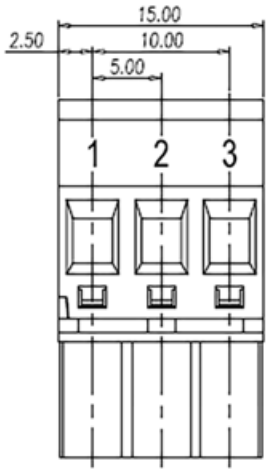
- *TA5211-TSPF-B (1SAP187400R0002) for PM5012-x-ETH*
- *TA5212-TSPF (1SAP187400R0005) for PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH(W)*

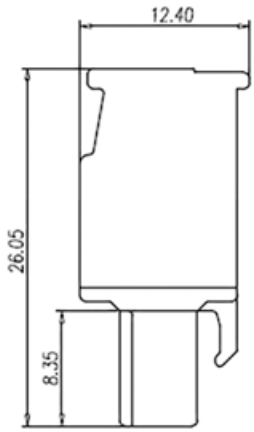
Dimensions

3-pin terminal block for power supply

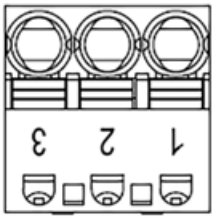
Screw type

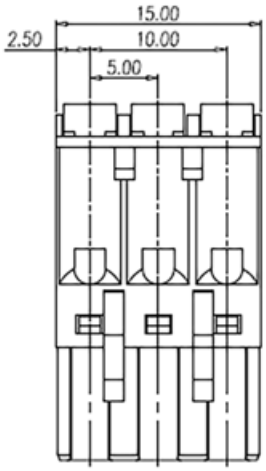


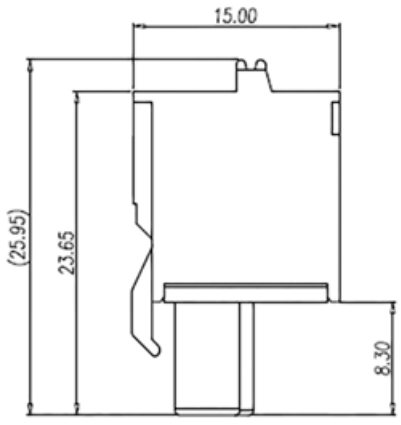




Spring type

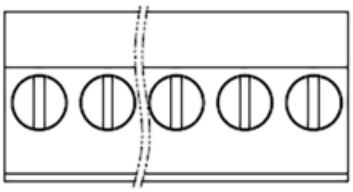


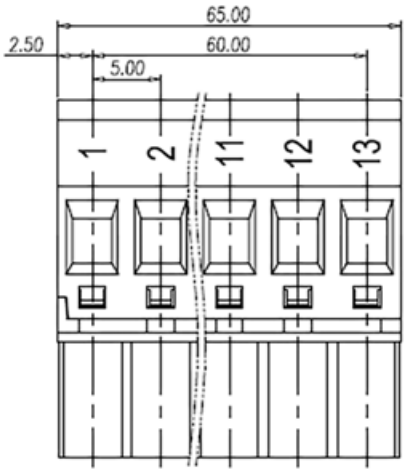


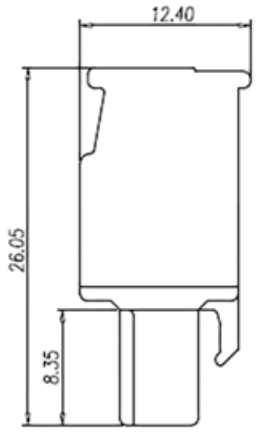


13-pin terminal block for I/O connectors

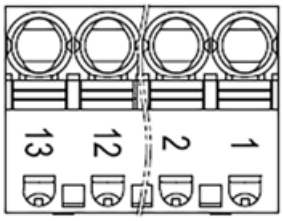
Screw type

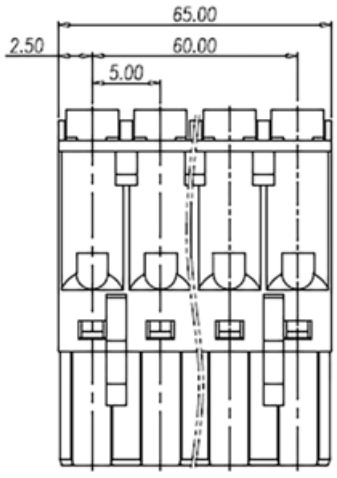


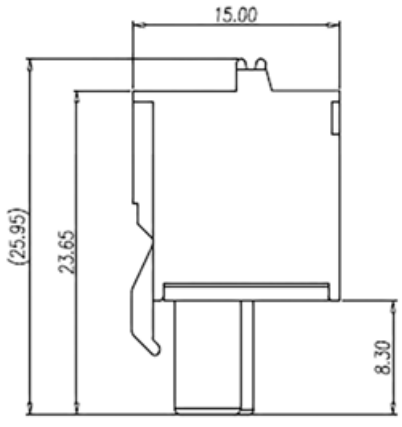




Spring type

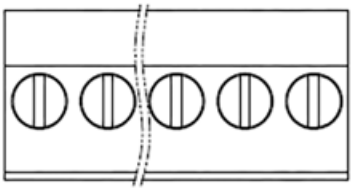


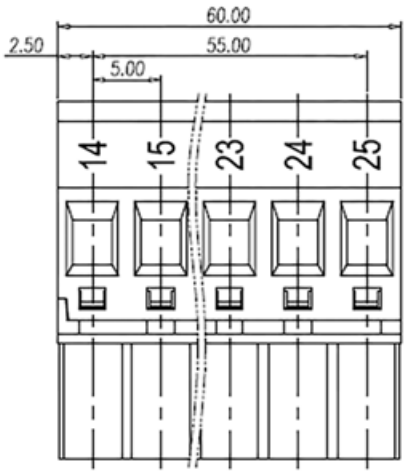


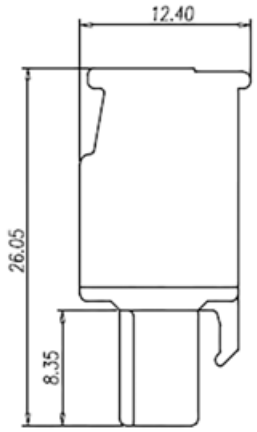


12-pin terminal block for I/O connectors

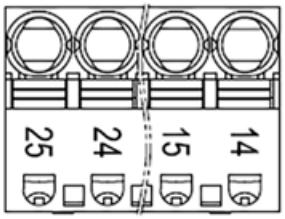
Screw type

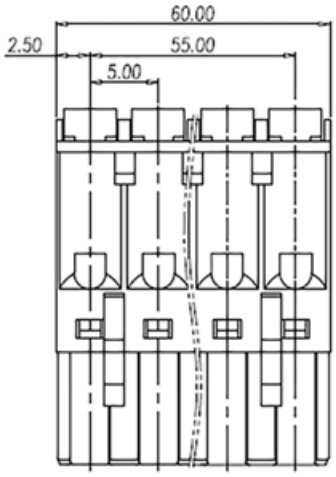


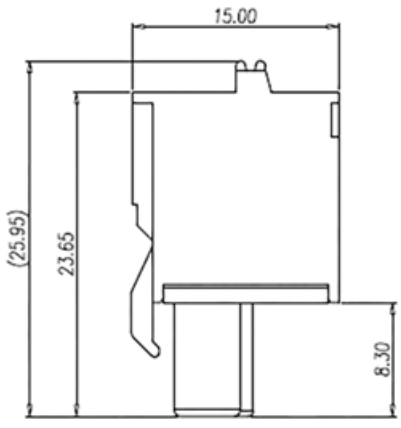




Spring type

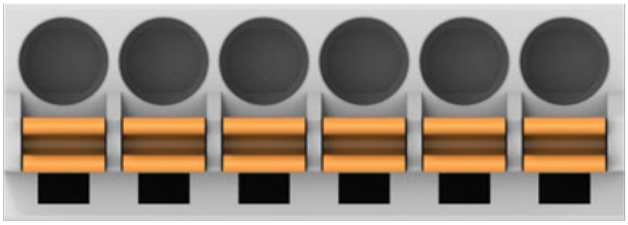


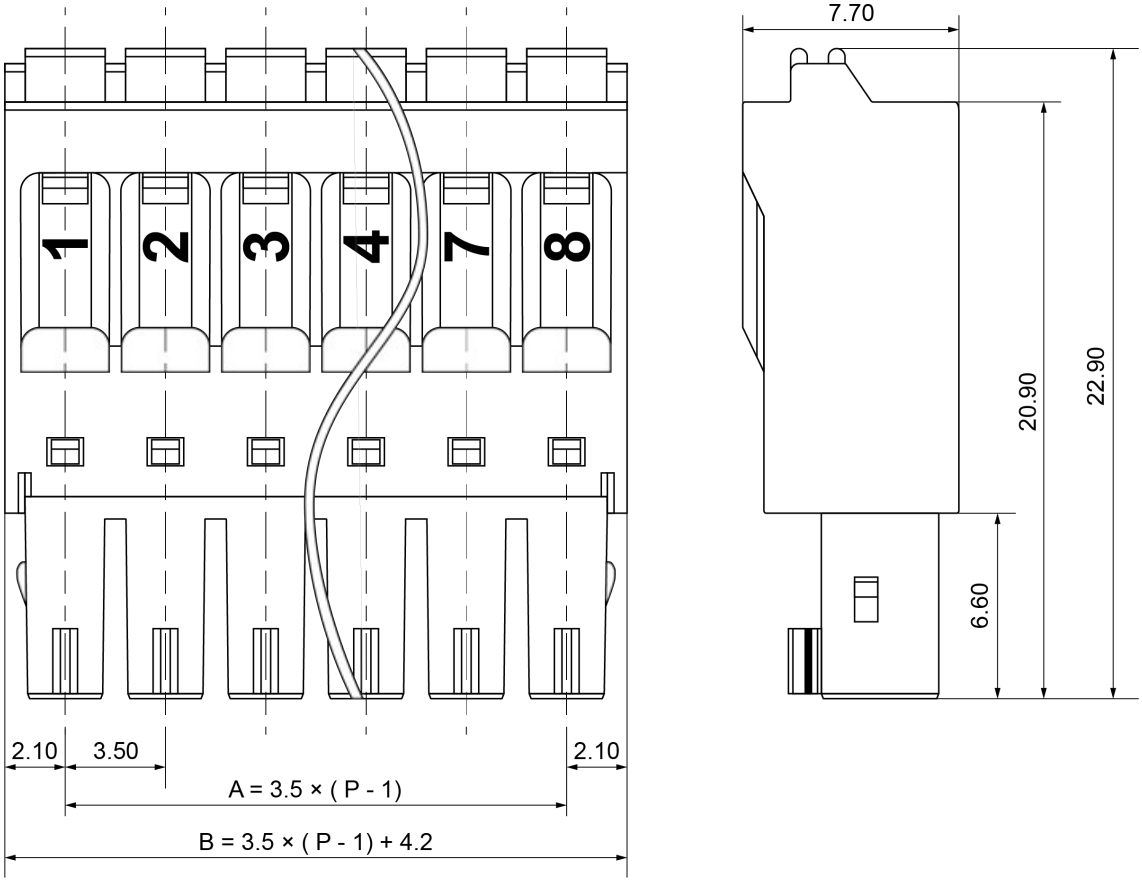




x-PIN terminal blocks for option boards

Only these x-pin blocks are available for the option boards.
*TA5220-SPF**x**, with **x** = 5...8*

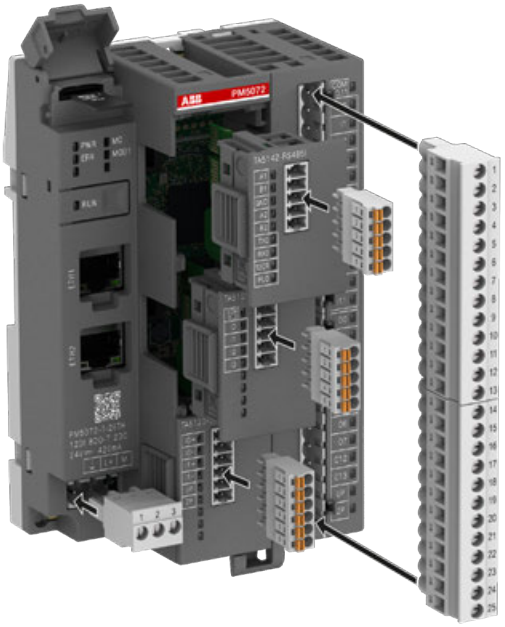




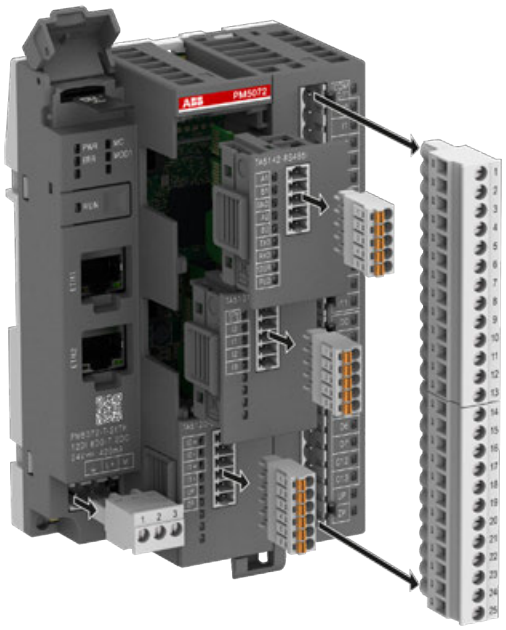
This results in these dimensions for the available spring terminal blocks.

Description	Pin	Length [mm]	Wide [mm]	Height [mm]
TA5220-SPF5	5	18.2	7.7	22.9
TA5220-SPF6	6	21.7	7.7	22.9
TA5220-SPF7	7	25.2	7.7	22.9
TA5220-SPF8	8	28.7	7.7	22.9

Assembly



Disassembly



Technical data

Table 583: Screw type terminal block for power supply

Parameter		Value
Type		
	TA5211-TSCL-B	Removable 3-pin terminal block:
	TA5212-TSCL	screw front/cable side 5.00 mm pitch
Usage		Power supply for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		7 mm

Parameter		Value
Fastening torque		0.5 Nm
Dimensions		
	3-pin terminal block	15 mm x 12.4 mm x 26.05 mm
Weight		
	TA5211-TSCL-B	150 g (2 terminal blocks)
	TA5212-TSCL	200 g (3 terminal blocks)

Table 584: Spring type terminal block for power supply

Parameter		Value
Type		
	TA5211-TSPF-B	Removable 3-pin terminal block:
	TA5212-TSPF	spring front/cable front 5.00 mm pitch
Usage		Power supply for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		11 mm
Dimensions		
	3-pin terminal block	15 mm x 15 mm x 25.95 mm
Weight		
	TA5211-TSPF-B	150 g (2 terminal blocks)
	TA5212-TSPF	200 g (3 terminal blocks)

Table 585: Screw type terminal block for onboard I/Os

Parameter		Value
Type		
	TA5211-TSCL-B	Removable 13-pin terminal block: screw front/cable side 5.00 mm pitch
	TA5212-TSCL	Removable 13-pin and 12-pin terminal block: screw front/cable side 5.00 mm pitch
Usage		Onboard I/Os for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		7 mm
Fastening torque		0.5 Nm
Dimensions		
	13-pin terminal block	65 mm x 12.4 mm x 26.05 mm
	12-pin terminal block	60 mm x 12.4 mm x 26.05 mm
Weight		

Parameter		Value
	TA5211-TSCL-B	150 g (2 terminal blocks)
	TA5212-TSCL	200 g (3 terminal blocks)

Table 586: Spring type terminal block for onboard I/Os

Parameter		Value
Type		
	TA5211-TSPF-B	Removable 13-pin terminal block: spring front/cable front 5.00 mm pitch
	TA5212-TSPF	Removable 13-pin and 12-pin terminal block: spring front/cable front 5.00 mm pitch
Usage		Onboard I/Os for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		11 mm
Dimensions		
	13-pin terminal block	65 mm x 15 mm x 25.95 mm
	12-pin terminal block	60 mm x 15 mm x 25.95 mm
Weight		
	TA5211-TSPF-B	150 g (2 terminal blocks)
	TA5212-TSPF	200 g (3 terminal blocks)

Table 587: Spring type terminal block for option boards

Parameter		Value
Type		
	TA5220-SPF5	Removable 5-pin terminal block: spring front, cable front 3.50 mm pitch
	TA5220-SPF6	Removable 6-pin terminal block: spring front, cable front 3.50 mm pitch
	TA5220-SPF7	Removable 7-pin terminal block: spring front, cable front 3.50 mm pitch
	TA5220-SPF8	Removable 8-pin terminal block: spring front, cable front 3.50 mm pitch
Usage		Connectors for AC500-eCo V3 option boards
Conductor cross section		
	Solid (copper)	0.2 mm^2...1.5 mm^2
	Flexible (copper)	0.2 mm^2...1.5 mm^2
Stripped conductor end		8 mm...10 mm
Dimensions		
	TA5220-SPF5	18.2 mm x 7.7 mm x 22.9 mm
	TA5220-SPF6	21.7 mm x 7.7 mm x 22.9 mm

Parameter		Value
	TA5220-SPF7	25.2 mm x 7.7 mm x 22.9 mm
	TA5220-SPF8	28.7 mm x 7.7 mm x 22.9 mm
Weight		
	TA5220-SPF5	150 g
	TA5220-SPF6	170 g
	TA5220-SPF7	180 g
	TA5220-SPF8	200 g

Ordering data

Part no.	Description
1SAP 187 400 R0001	TA5211-TSCL-B: screw terminal block set for AC500-eCo V3 CPU Basic screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors
1SAP 187 400 R0002	TA5211-TSPF-B: spring terminal block set for AC500-eCo V3 CPU Basic spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors

Part no.	Description
1SAP 187 400 R0004	TA5212-TSCL: screw terminal block set for AC500-eCo V3 Standard and Pro CPU screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors
1SAP 187 400 R0005	TA5212-TSPF: spring terminal block set for AC500-eCo V3 Standard and Pro CPU spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors

Part no.	Description
Spare parts	
1SAP 187 400 R0012	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0013	TA5220-SPF6: spring terminal block, removable, 6-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0014	TA5220-SPF7: spring terminal block, removable, 7-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0015	TA5220-SPF8: spring terminal block, removable, 8-pin, spring front, cable front, 6 pieces per packing unit

TA5300-CVR - Option board slot cover

Intended purpose

TA5300-CVR option board slot covers for PM50xx processor modules are necessary to protect not used option board slots.

CAUTION!

Risk of injury and damaging the product!

Always plug in the option board slot cover when the option board is not inserted.

If the option board slot cover is lost, please order the replacement TA5300-CVR (1SAP187500R0001).

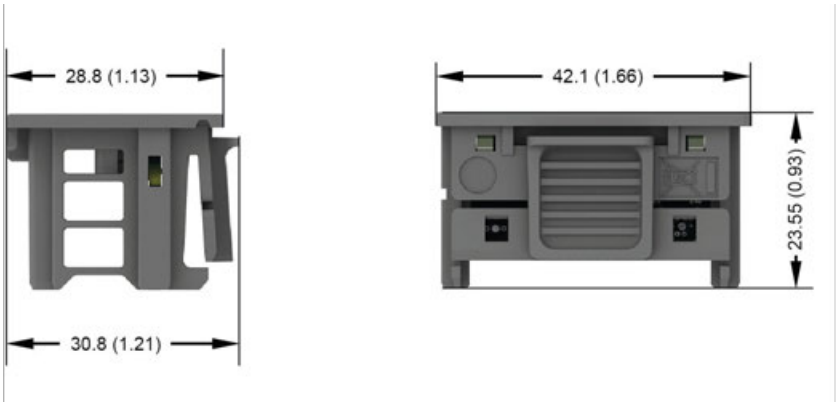
Never power up the CPU with uncovered option board slot, otherwise it may cause serious injury and/or damage the product.

The AC500-eCo V3 processor modules are delivered with option board slot cover(s).

The option board slot cover has to be removed before inserting an option board.

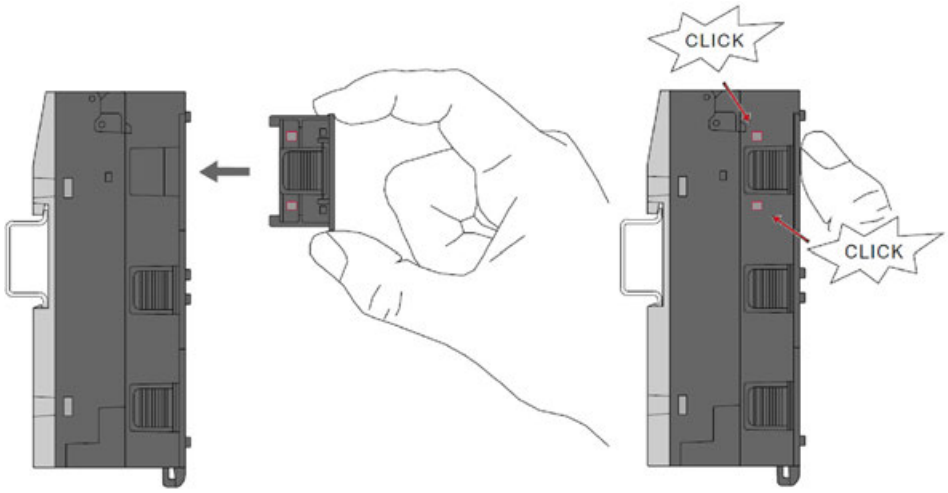
The TA5300-CVR option board slot covers are available as spare parts.

Dimensions



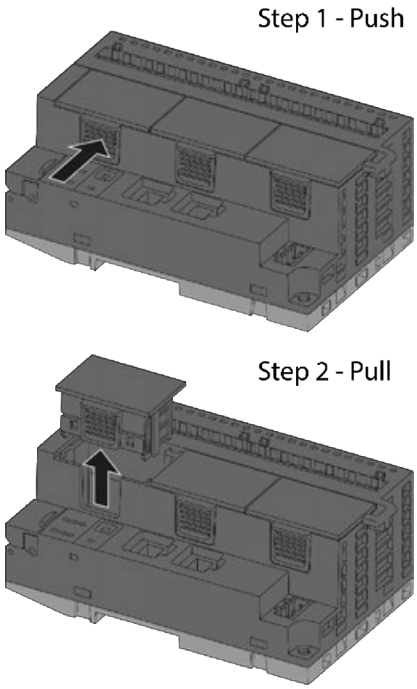
The dimensions are in mm and in brackets in inch.

Inserting of the option board slot cover



1. Press on the option board slot cover to insert it in the not used option board slot of the processor module PM50xx.
2. The option board slot cover must click into the not used option board slot.

Removing of the option board slot cover



1. Press the side of the inserted option board slot cover.
2. At the same time, pull the option board slot cover out of the option board slot of the processor module PM50xx.

Technical data

The system data of AC500-eCo V3 apply *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Weight	47 g
Dimensions	42.1 mm x 30.8 mm x 23.55

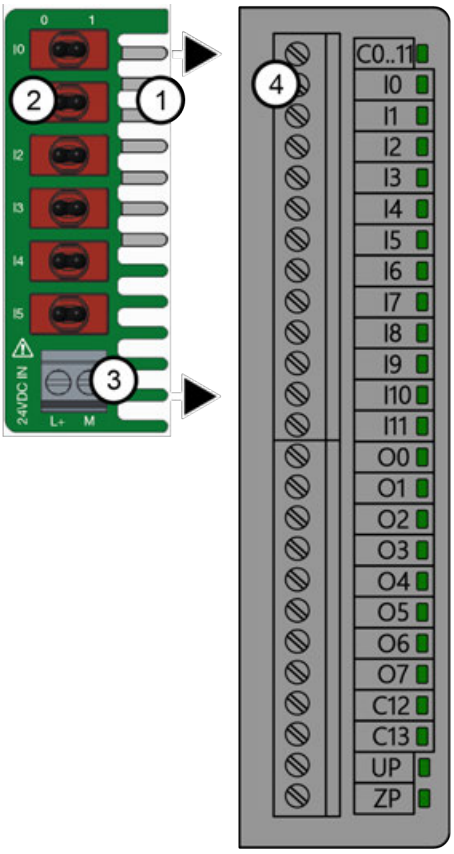
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 500 R0001	TA5300-CVR: option board slot cover, removable plastic part, 6 pieces per packing unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA5400-SIM - Input simulator

- TA5400-SIM input simulator for 6 digital inputs 24 V DC
- For usage with AC500-eCo V3 processor modules



1 Contacts for connecting the input simulator to the terminal block for I/O connectors
2 6 switches for the digital inputs DI0 ... DI5 (0 means opened switch, 1 means closed switch)
3 Screw terminal block for power supply
4 Screw terminal block(s) for I/O connectors

Intended purpose

TA5400-SIM

The TA5400-SIM input simulator is only intended for testing and training purposes for AC500-eCo V3 processor modules PM50x2.

Continuous operation in a productive system is not permitted.

The TA5400-SIM input simulator may only be used with screw-type terminal blocks.

The TA5400-SIM input simulator must not be used with spring-type terminal blocks.

Environmental conditions for testing and training purposes

In order not to impair the functionality of the product, avoid any kind of disturbing environmental influences:

- *mechanical disturbances*
- *climatic influences*

Make sure that the parameters are within the normal range:

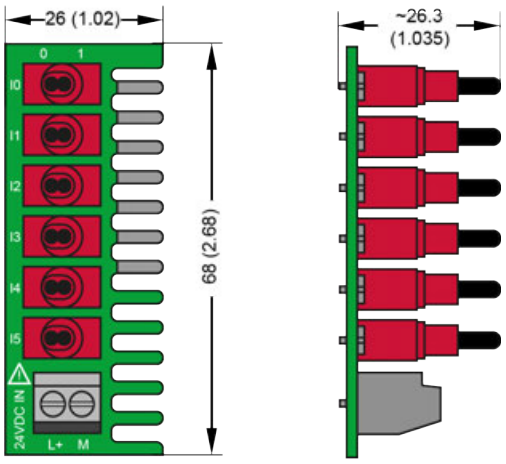
- *temperature*
- *air pressure*
- *humidity*
- *altitude*

The TA5400-SIM input simulator can simulate 6 digital 24 V DC input signals to the digital inputs I0...I5 of onboard I/Os.

With the TA5400-SIM input simulator, the digital 24 V DC inputs I0...I5 can be turned OFF and ON separately:

- If the lever of the switch is on the right side (1), the input is ON.
- If the lever of the switch is on the left side (0), the input is OFF.

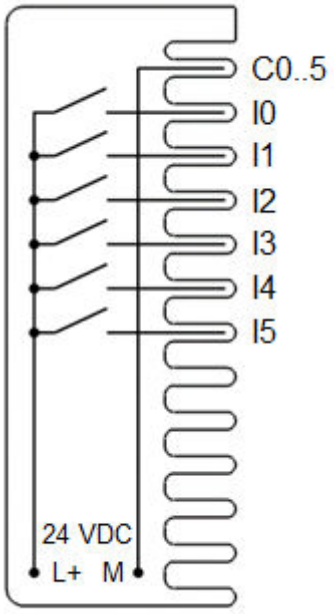
Dimensions



The dimensions are in mm and in brackets in inch.

Electrical diagram

The diagram below shows the connection of the TA5400-SIM input simulator.



NOTICE!

Risk of damage to the TA5400-SIM input simulator!

Do not remove the terminal block while the TA5400-SIM input simulator is connected.

Do not apply mechanical forces to the input simulator when it is connected to the terminal block.

In both cases the input simulator could be damaged.

Assembly

Insertion of the input simulator

1. Make sure that the power supply of the processor module is turned off.

CAUTION!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overvoltages and short circuits.

Make sure, that all voltage sources (supply and process voltage) are switched off before you start working on the system.

Never connect voltages > 24 V DC to the terminal block of the TA5400-SIM input simulator.

CAUTION!

Risk of damaging the input simulator and/or PLC modules!

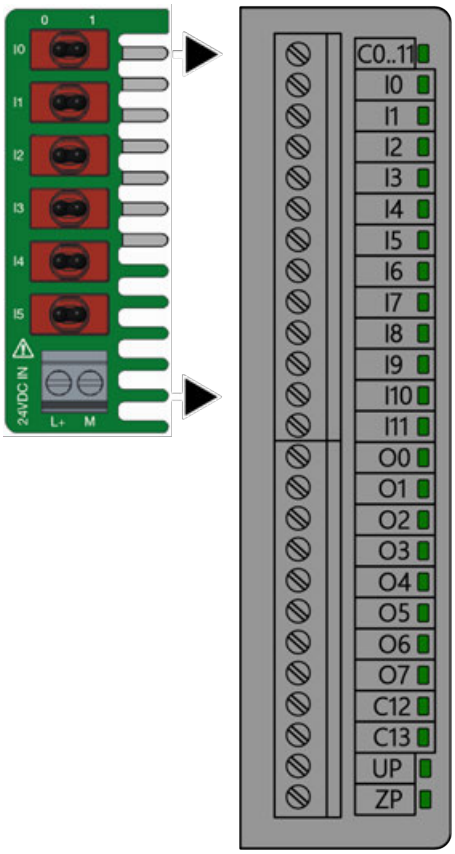
The TA5400-SIM input simulator may only be used with AC500-eCo V3 processor modules PM50x2.

Never use the input simulator with other devices.

The input simulator may only be used with screw-type terminal blocks.

The input simulator is only intended for testing and training purposes. Never use it within productive sytems.

2. Make sure that all clamps of the onboard I/Os are totally open.
3. Insert the TA5400-SIM input simulator into the screw terminal block as shown in the figure.



4. Tighten all screws of the onboard I/O clamps.
5. Make sure all switches are in OFF state (0).
6. Connect 24 V DC to the power supply of the TA5400-SIM (L+ and M). Tighten the screws.
7. Connect the processor module power supply wires (24 V DC). See PM50xx ↳ *"Pin assignment" on page 3371*.

Disassembly

Removal of the input simulator

1. Make sure that the power supply of the processor module is turned off.

> **CAUTION!**
> **Risk of damaging the PLC modules!**
> The PLC modules can be damaged by overvoltages and short circuits.
> Make sure that all voltage sources (supply and process voltage) are switched off before you start working on the system.

2. Disconnect the TA5400-SIM power supply wires (24 V DC) with a flat-blade screwdriver from the terminal block for power supply (L+ and M).
3. Loosen all screws of the onboard I/Os.
4. Remove the input simulator by pulling it to the left side.

Technical data

The system data of AC500-eCo V3 apply ↳ *Chapter 1.6.4.5.1 "System data AC500-eCo V3" on page 3352*

Only additional details are therefore documented below.

Table 588: Technical data of the module

Parameter		Value
Process supply voltage		
	Connections	Terminal (L+) for +24 V DC and terminal (M) for 0 V DC
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
Galvanic isolation		Yes (on processor module PM50xx)
Isolated Groups		1 (6 channels per group)
Weight		18 g
Mounting position		Horizontal or vertical

Table 589: Technical data of the inputs

Parameter	Value
Number of channels per module	6 digital input channels (+24 V DC)
Distribution of the channels into groups	1 (6 channels per group)
Connections of channels I0 to I5	Terminals 2...7
Reference potential for the channels I0 to I5	Terminal 1 (negative pole of the process supply voltage, signal name C0...5)
Input current per active channel (at input voltage +24 V DC) The current is given through the used processor module.	Typ. 5 mA
Inrush current per active channel The current is given through the used processor module.	Typ. 5 mA

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 600 R0001	TA5400-SIM, input simulator for PM50x2	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.3.8.2 AC500 (standard)

MC502 - Memory card

- Solid state flash memory storage

1 MC502 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

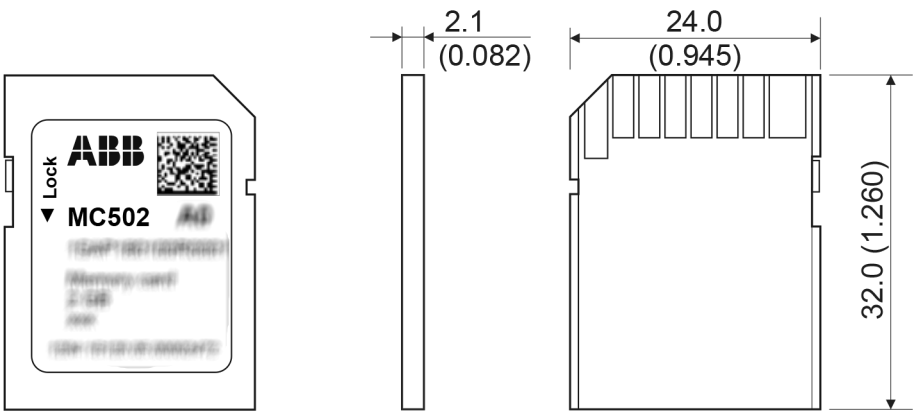
Processor modules are supplied without (micro) memory card. It must be ordered separately.

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V3

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

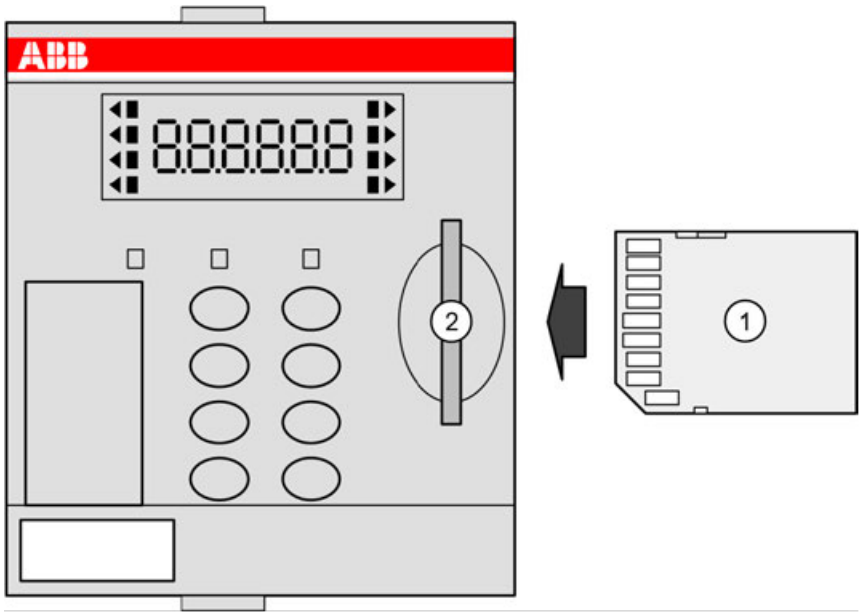


Fig. 257: Insert memory card into PM56xx

1 Memory card
2 Memory card slot

Remove the memory card

AC500 V3

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when no black square (■) is shown next to MC in the display.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

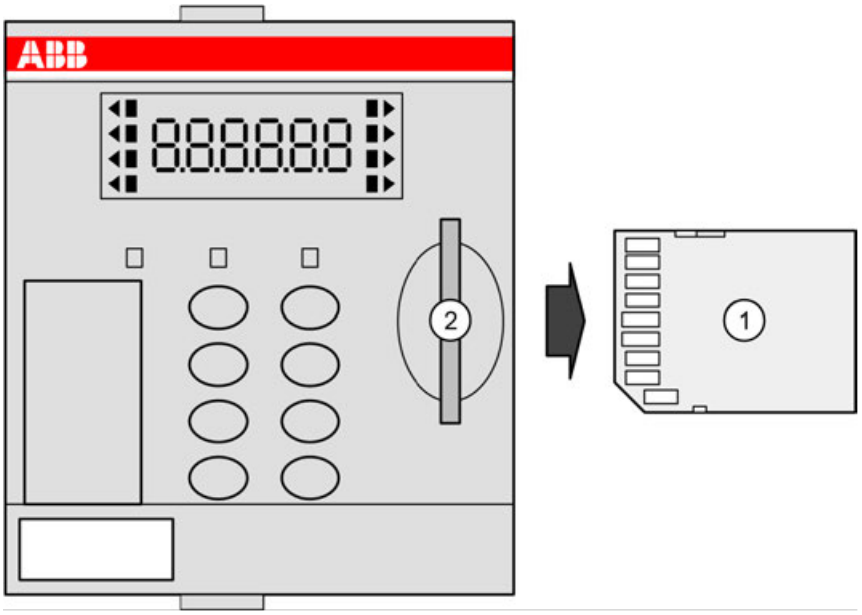


Fig. 258: Remove memory card from PM56xx

1 Memory card
2 Memory card slot

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter
↳ *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

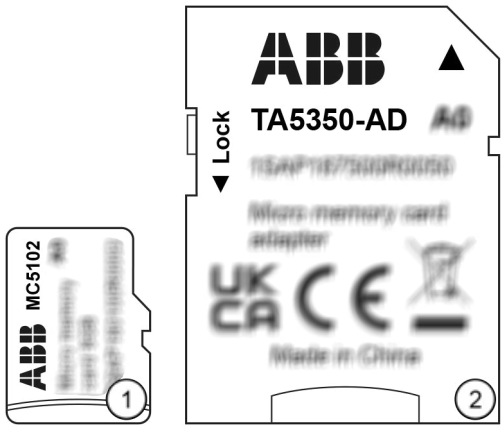
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0001	MC502, memory card	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.
The TA5350-AD micro memory card adapter has a write protect switch.
In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

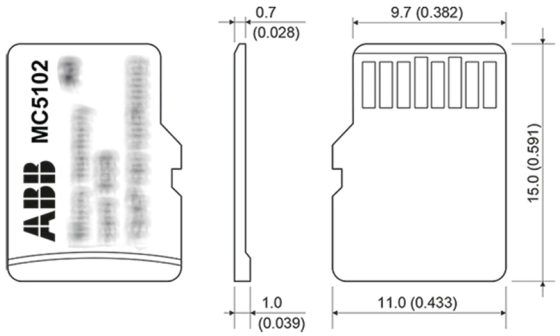
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

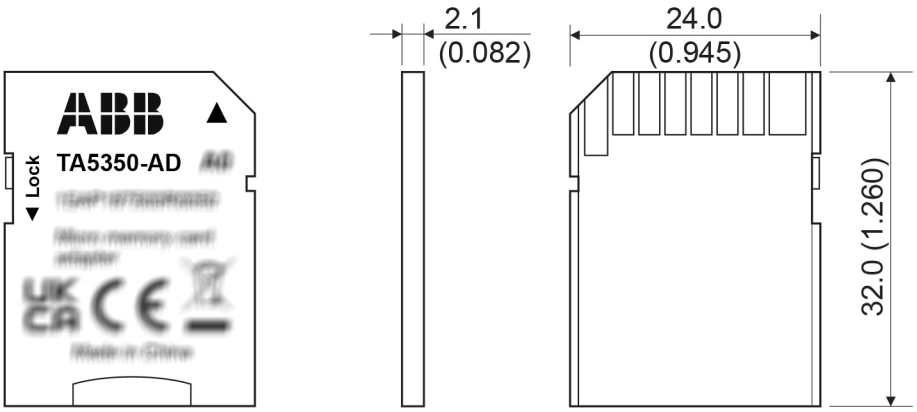
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V3

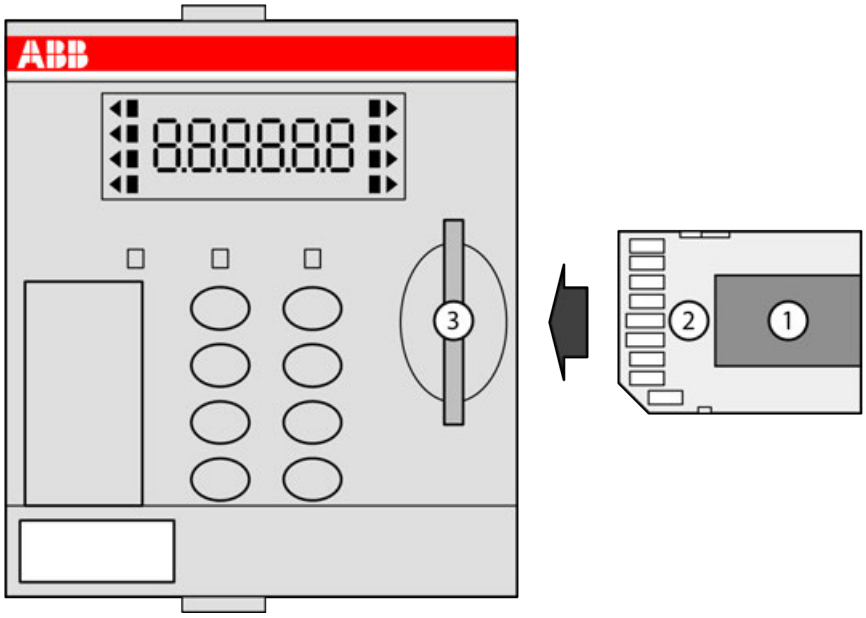
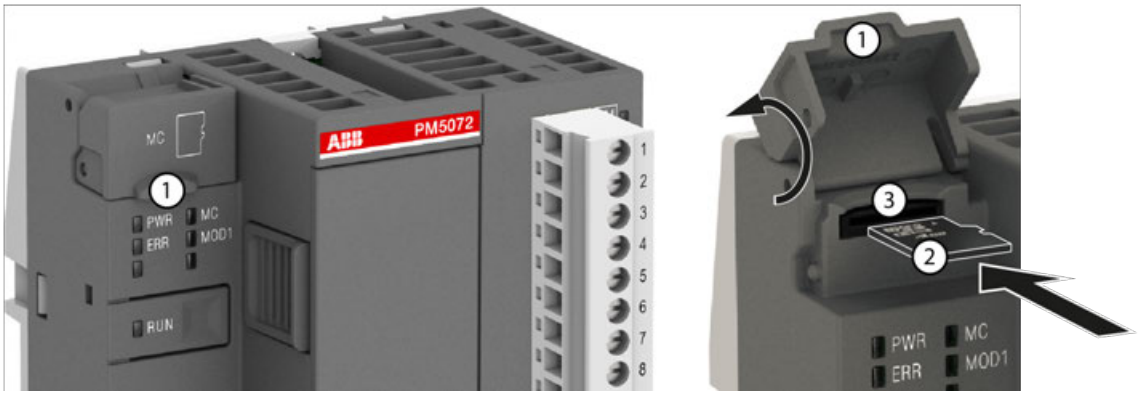


Fig. 259: Insert micro memory card into PM56xx

1 Micro memory card
2 TA5350-AD micro memory card adapter
3 Memory card slot

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Carefully insert the micro memory card into the micro memory card slot as far as it will go. Observe orientation of card.
3. Close the micro memory card slot cover by turning it downwards.

Remove the micro memory card

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

AC500 V3: Remove the micro memory card with micro memory card adapter only when no black square (■) is shown next to MC in the display.

AC500-eCo V3: Remove the micro memory card only when the MC LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

AC500 V3

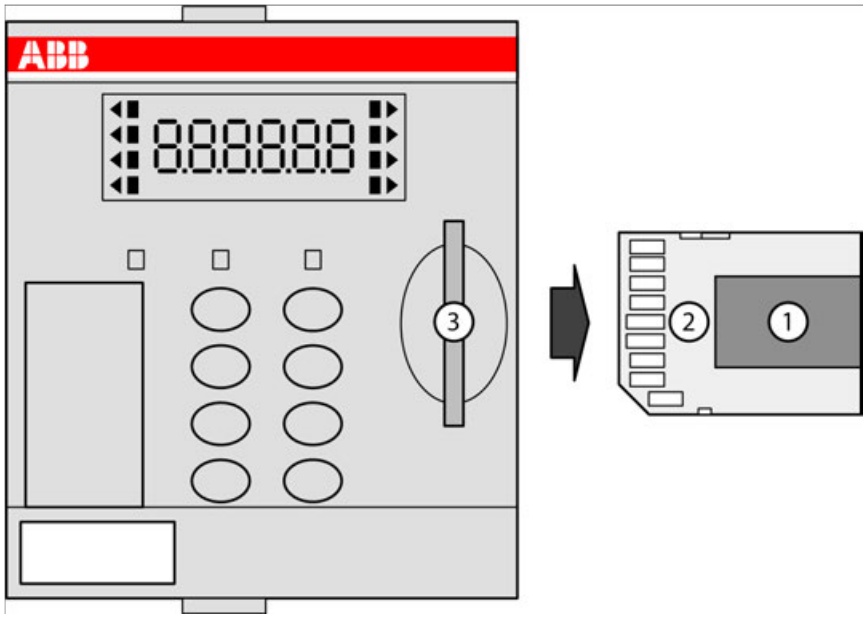
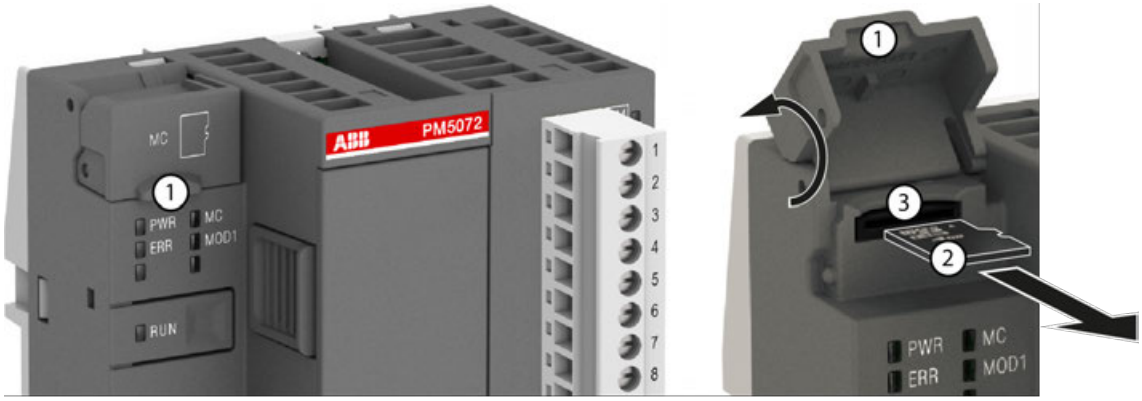


Fig. 260: Remove micro memory card from PM56xx

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Micro memory card can be removed from the micro memory card slot by gripping and pulling with two fingers.
3. Close the micro memory card slot cover by turning it downwards.

Technical data

Parameter	Value
Memory capacity	8 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC5141 - Memory card

- Solid state flash memory storage

1 MC5141 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	**AC500 V2**	**AC500-XC V2**	**AC500-eCo V2 [3])**	**AC500 V3**	**AC500-XC V3**	**AC500-eCo V3**
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

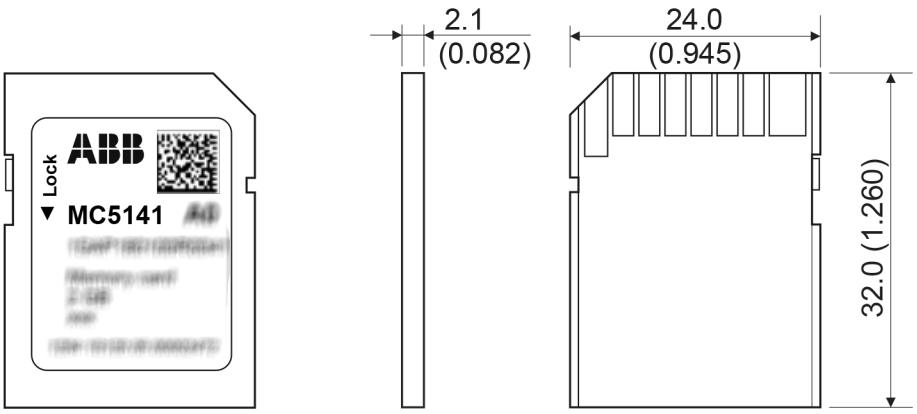
Processor modules are supplied without (micro) memory card. It must be ordered separately.

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V3

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

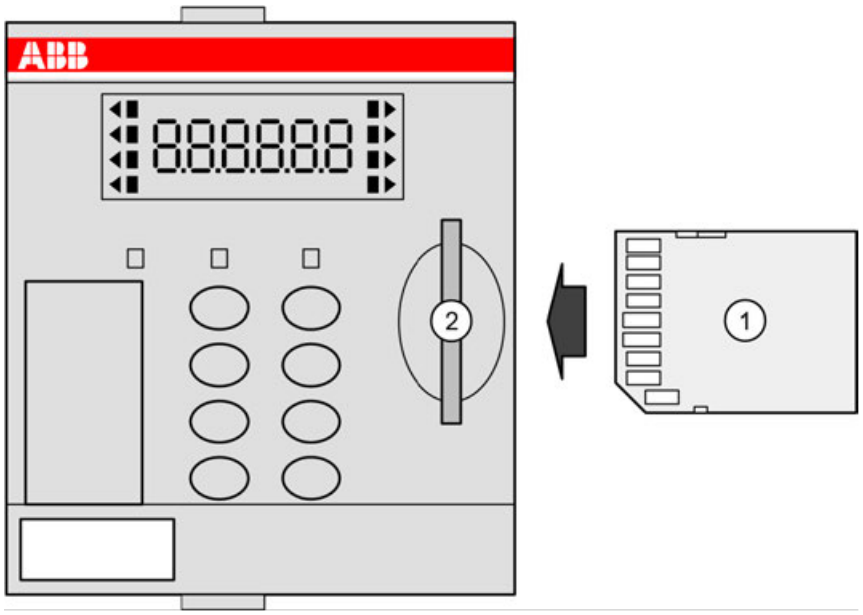


Fig. 261: Insert memory card into PM56xx

1 Memory card
2 Memory card slot

Remove the memory card

AC500 V3

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when no black square (■) is shown next to MC in the display.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

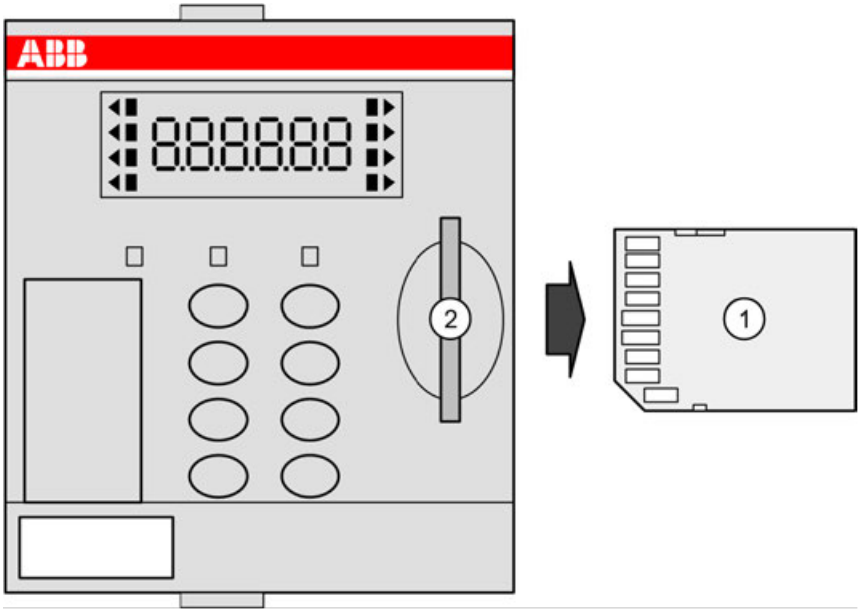


Fig. 262: Remove memory card from PM56xx

1 Memory card
2 Memory card slot

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

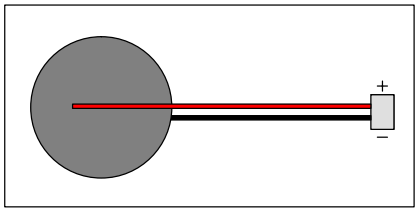
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0041	MC5141, memory card	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA521 - Battery

- Manganese dioxide lithium battery, 3 V, 560 mAh
- Non-rechargeable



Purpose

The TA521 battery is the only applicable battery for the AC500 processor modules *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*. It cannot be recharged.

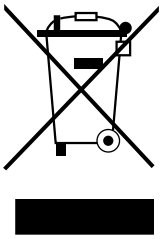
The processor modules are supplied without lithium battery. It must be ordered separately. The TA521 lithium battery is used for data (SRAM) and RTC buffering while the processor module is not powered.

See system technology - AC500 battery. *Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479*

The CPU monitors the discharge degree of the battery. A warning is issued before the battery condition becomes critical (about 2 weeks before). Once the warning message appears, the battery should be replaced as soon as possible.

Handling instructions

- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Replace the battery with supply voltage ON in order not to risk data being lost.
- Recycle exhausted batteries meeting the environmental standards.



Battery lifetime

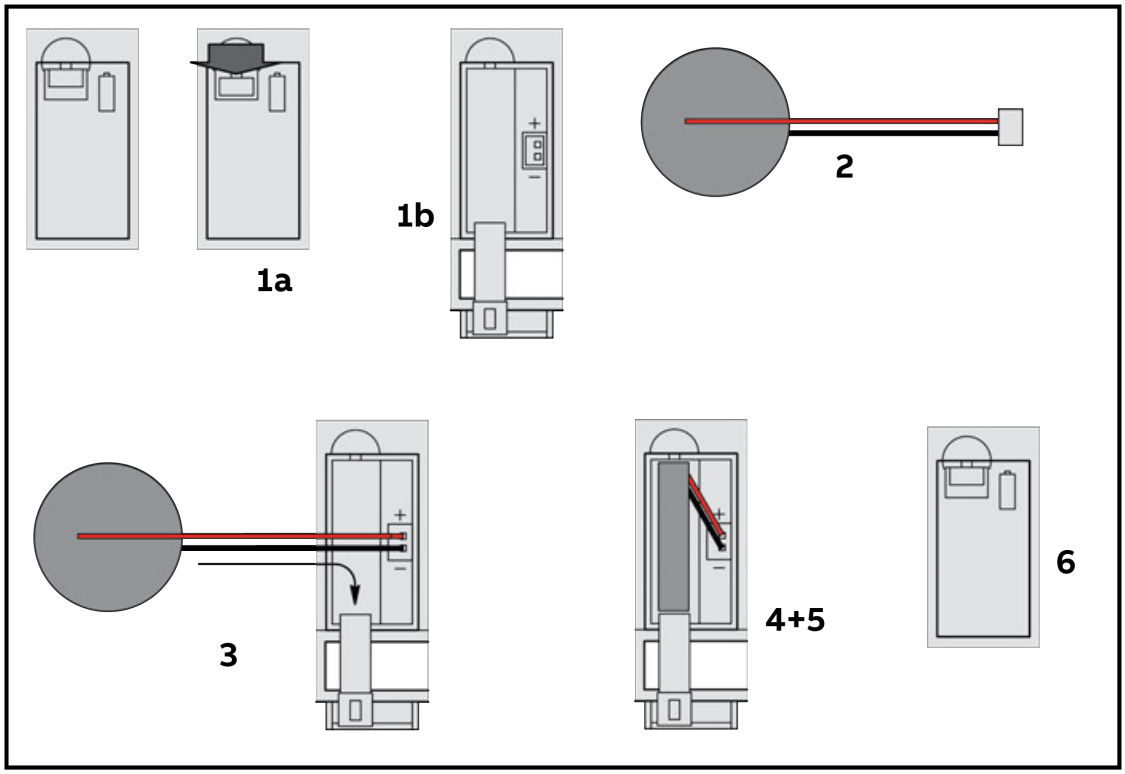
The battery lifetime is the time, the battery can store data while the processor module is not powered. As long as the processor module is powered, the battery will only be discharged by its own leakage current.

To avoid a short battery discharge, the battery should always be inserted or replaced while the process module is under power, then the battery is correctly recognized and will not shortly discharged.

Insertion

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

WARNING!
Risk of fire or explosion!
Use of incorrect Battery may cause fire or explosion.



1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front face of the processor module and cannot be removed.
2. Remove the TA521 battery from its package and hold it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.
3. Insert the battery connector into the small connector port of the compartment. The connector is keyed to find the correct polarity (red = positive pole = above).
4. Insert first the cable and then the battery into the compartment, push it until it reaches the bottom of the compartment.
5. Arrange the cable in order not to inhibit the door to close.
6. Pull-up the door and press until the locking mechanism snaps.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Replacement of the battery

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front view of the processor module and cannot be removed.
2. Remove the old TA521 battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.

3. Follow the previous instructions to insert a new battery.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Technical data

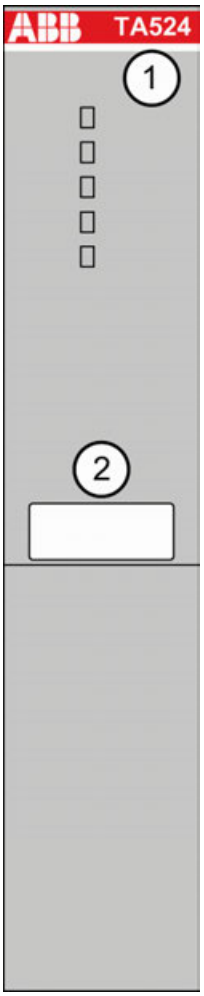
Parameter	Value
Nominal voltage	3 V
Nominal capacity	560 mAh
Temperature range (index below C0)	Operating: 0 °C...+60 °C Storage: -20 °C...+60 °C Transport: -20 °C...+60 °C
Temperature range (index C0 and above)	Operating: -40 °C...+70 °C Storage: -40 °C...+85 °C Transport: -40 °C...+85 °C
Battery lifetime	Typ. 3 years at 25 °C
Self-discharge	2 % per year at 25 °C 5 % per year at 40 °C 20 % per year at 60 °C
Protection against reverse polarity	Yes, by mechanical coding of the plug.
Insulation	The battery is completely insulated.
Connection	Red = positive pole = above at plug, black = negative pole,
Weight	7 g
Dimensions	Diameter of the button cell: 24.5 mm Thickness of the button cell: 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 300 R0001	TA521, lithium battery	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA524 - Dummy communication module



1 Type
2 Label

Purpose

TA524 is used to cover an unused communication module slot of a terminal base *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*. It protects the terminal base from dust and inadvertent touch.

Handling instructions

TA524 is mounted in the same way as a common communication module *Chapter 1.6.4.6.3.5 “Mounting/Demounting the communication modules” on page 3414*.

Technical data

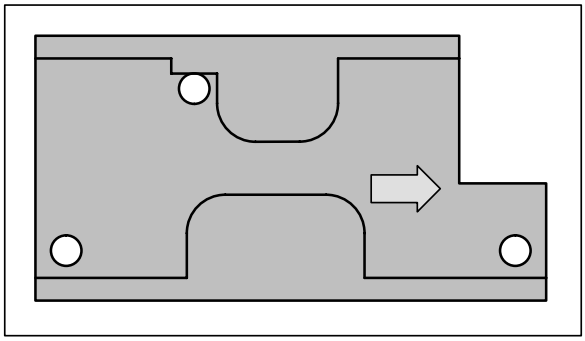
Parameter	Value
Weight	50 g
Dimensions	135 mm x 28 mm x 62 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 600 R0001	TA524, dummy communication module	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA526 - Wall mounting accessory



Purpose

If a terminal base TB5xx or a terminal unit TU5xx should be mounted with screws, the wall mounting accessories TA526 must be inserted at the rear side first. This plastic parts prevent bending of terminal bases and terminal units while screwing up.

Handling instructions

Handling of the wall mounting accessory is described in detail in the section *Mounting and disassembling the terminal unit* ↬ *"Mounting with screws" on page 3411* and *Mounting/Disassembling Terminal Bases and Function Module Terminal Bases* ↬ *"Mounting with screws" on page 3409*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	67 mm x 35 mm x 5,5 mm

Ordering data

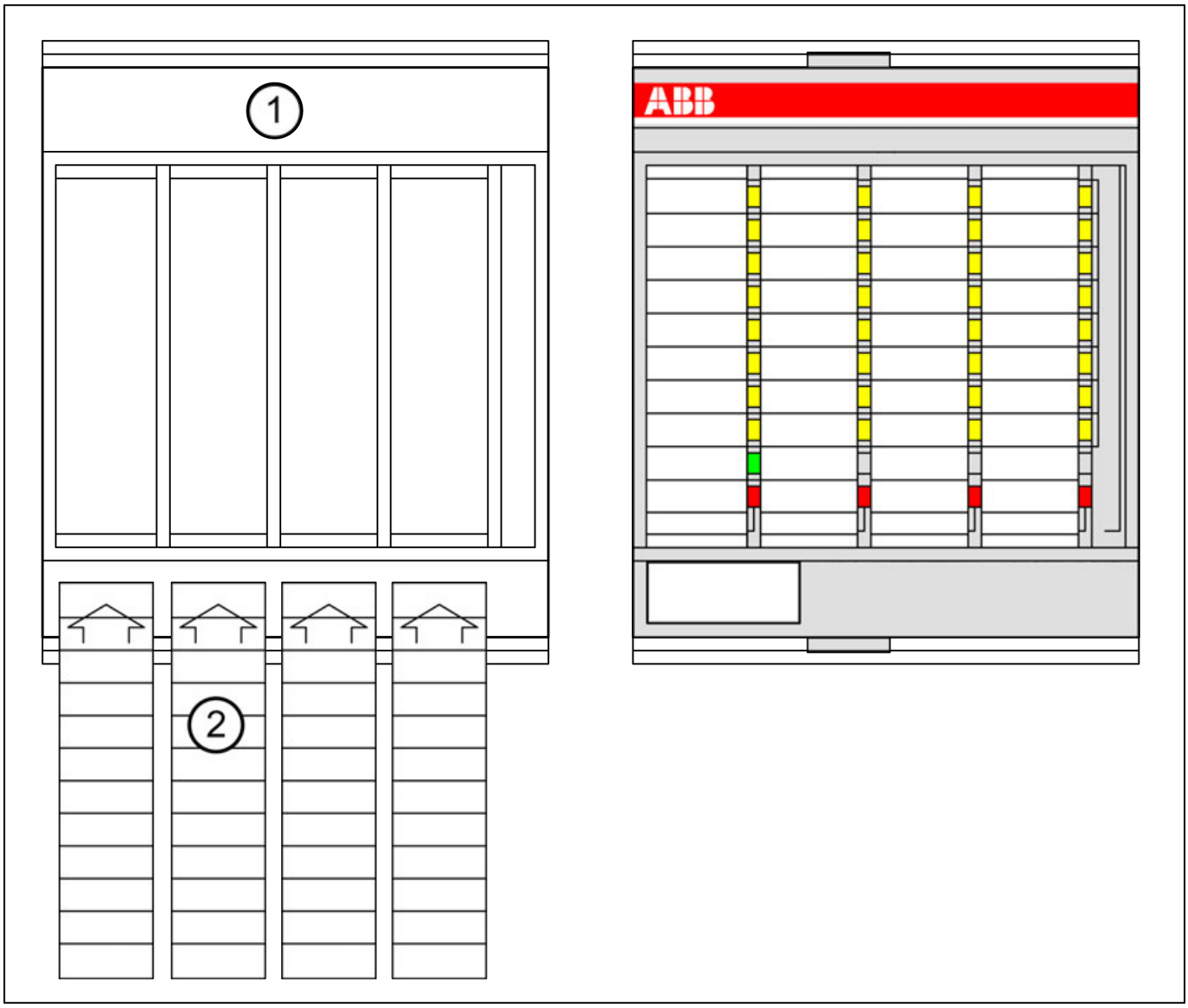
Part no.	Description	Product life cycle phase *)
1SAP 180 800 R0001	TA526, wall mounting accessory	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

1.6.3.8.3 S500

TA523 - Pluggable label mounting

For labelling the channels of S500 I/O modules.



1 Pluggable label mounting TA523
2 Plastic labels to be inserted into the holder

Purpose

The pluggable label mounting is used to hold 4 plastic labels, on which the meaning of the I/O channels of I/O modules can be written down. The holder is transparent so that after snapping it onto the module the LEDs shine through.

Handling instructions

The plastic labels can be printed out from TA563.doc *http://new.abb.com/products/ABB1SAP180500R0001*.

Technical data

Parameter	Value
Use	For labelling channels of I/O modules
Mounting	Snap-on to the module
Weight	20 g
Dimensions	82 mm x 67 mm x 13 mm

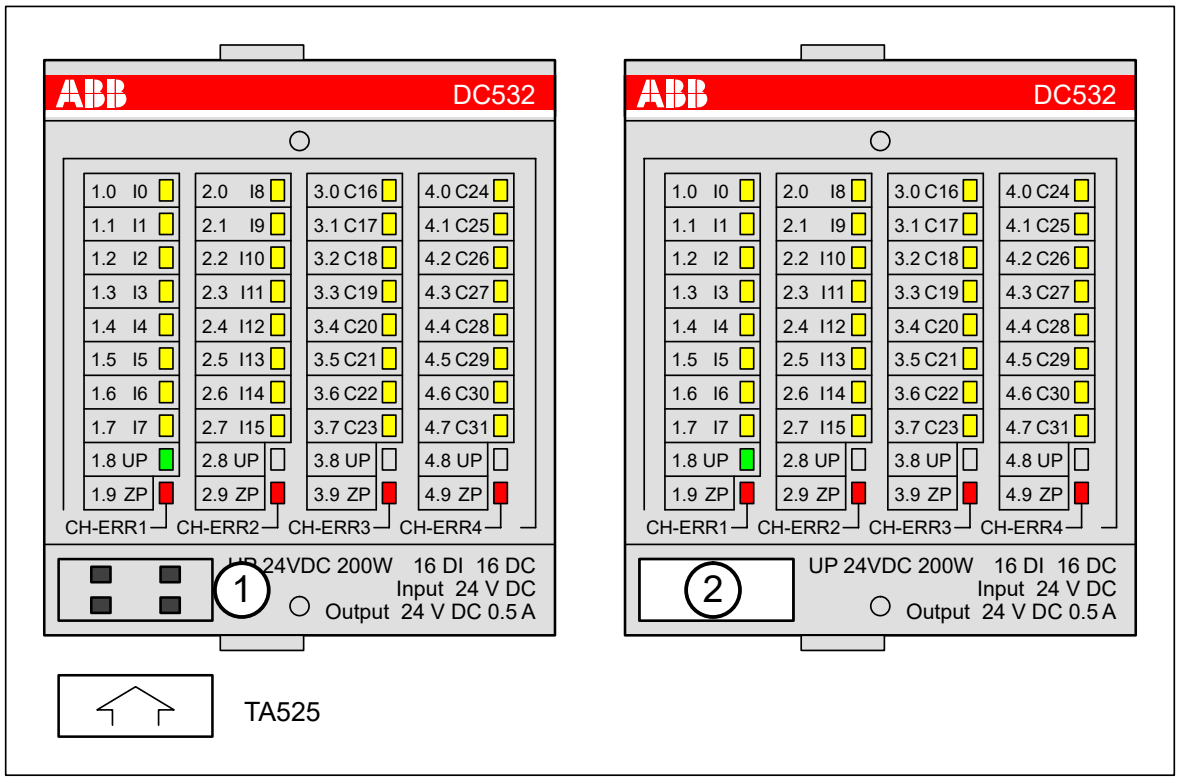
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 500 R0001	TA523, pluggable label mounting (10 pieces)	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA525 - Plastic labels

Accessory to label AC500 and S500 modules.



1 Module without plastic label TA525
2 Module with plastic label TA525

Purpose

The plastic labels are suitable for labelling AC500 and S500 modules (CPUs, communication modules and I/O modules). The small plastic parts can be written on with a standard waterproof pen.

Handling instructions

The plastic labels are inserted under a slight pressure. For disassembly, a small screwdriver is inserted at the lower edge of the module.

Technical data

Parameter	Value
Use	For labelling AC500 and S500 modules
Mounting	Insertion under a slight pressure

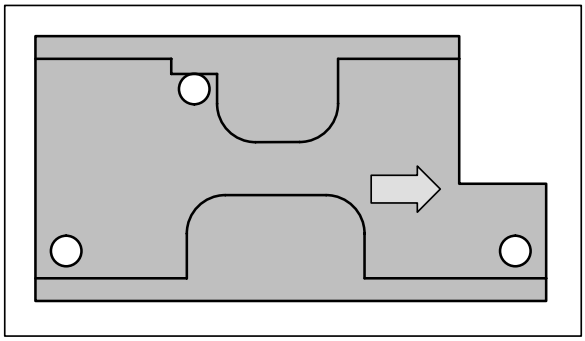
Parameter	Value
Disassembly	With a small screwdriver
Scope of delivery	10 pieces
Weight	1 g per piece
Dimensions	8 mm x 20 mm x 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 700 R0001	TA525, Set of 10 white plastic labels	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA526 - Wall mounting accessory



Purpose

If a terminal base TB5xx or a terminal unit TU5xx should be mounted with screws, the wall mounting accessories TA526 must be inserted at the rear side first. This plastic parts prevent bending of terminal bases and terminal units while screwing up.

Handling instructions

Handling of the wall mounting accessory is described in detail in the section *Mounting and disassembling the terminal unit* ↬ *"Mounting with screws" on page 3411* and *Mounting/Disassembling Terminal Bases and Function Module Terminal Bases* ↬ *"Mounting with screws" on page 3409*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	67 mm x 35 mm x 5,5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 800 R0001	TA526, wall mounting accessory	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA535 - Protective caps for XC devices

Purpose

Accessory to cover unused connectors of XC devices in salt mist environments.

One TA535 package includes different cap types for the following connectors:

- RJ45 connectors
- 9-pole D-sub connector
- FieldBusPlug connector

Protection should be done for all unused slots of -XC devices.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 300 R0001	TA535, Protective Caps for XC devices	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

1.6.4 System assembly, construction and connection

1.6.4.1 Introduction

This chapter provides information on assembly, construction and connection of control systems of the product family AC500.

The AC500 product family consists of the sub-families:

- AC500 (standard): standard PLC that offers a wide range of performance levels and scalability.
- AC500-eCo: cost-effective PLC that offers total inter-operability with the core AC500 range.
- AC500-S: PLC for special safety requirements in all functional safety applications.

AC500 (standard) and AC500-S provide devices with -XC extension as a product variant. Those devices operate mainly identical to the appropriate AC500 product family, however, can be operated under extreme conditions ↳ *Chapter 1.6.4.7.1 “System data AC500-XC” on page 3450*.

AC500 product family is characterized by functional modularity, i.e. the devices of all AC500 sub-families can be combined flexible.

As assembly, construction and connection for the devices of the AC500 product family is similar, information that is valid for all sub-families is provided within an overall section. Details that are only valid for a specific AC500 sub-family are described in separate sections.

As assembly, construction and connection for the devices of the AC500 product family is similar, information that is valid for all sub-families is provided within an overall section *Chapter 1.6.4.4 “Overall information (valid for complete AC500 product family)” on page 3338*. Details that are only valid for a specific AC500 sub-family are described in separate sections.

Consider the safety instructions

In the description, special attention must be paid to designs using galvanic isolation, grounding and EMC measures for the reasons stated. Consider the safety instructions for AC500 product family Chapter 1.6.4.3 “Safety instructions” on page 3335.

1.6.4.2 Regulations

Appropriate system setup

The following regulations have to be taken into due consideration:

- DIN VDE 0100: "Regulations for the Setting up of Power Installations"
- DIN VDE 0110 Part 1 and Part 2: "The Rating of Creepage Distances and Clearances"
- DIN VDE 0160 and DIN VDE 0660 Part 500: "The Equipment of Power Installations with Electrical Components"

To ensure project success and proper installation of all systems, customers must be familiar and proficient with the following standards and must comply with their directives:

- DIN VDE 0113 Part 1 & Part 200: "Working & Process Machinery"
- DIN VDE 0106 Part 100: "Close proximity to dangerous voltages"
- DIN VDE 0160, DIN VDE 0110 Part 1: "Protection against direct contact"

The user has to guarantee that the devices and the components are mounted following these regulations. For operating the machines and installations, other national and international relevant regulations, concerning prevention of accidents and using technical working means, also have to be met.

AC500 devices are designed according to IEC 1131 Part 2 under overvoltage category II per DIN VDE 0110 Part 2.

For direct connection of AC Category III overvoltages provide protection measures for overvoltage category II according to IEC-Report 664/1980 and DIN VDE 0110 Part 1.

Equivalent standards:

- DIN VDE 0110 Part 1 ↔ IEC 664
- DIN VDE 0113 Part 1 ↔ EN 60204 Part 1
- DIN VDE 0660 Part 500 ↔ EN 60439-1 ↔ IEC 439-1

Qualified personnel

Both the control system AC500 and other components in the vicinity are operated with dangerous contact voltages. Touching parts, which are under such voltages, can cause grave damage to health.

In order to avoid such risks and the occurrence of material damage, persons involved with the assembly, starting up and servicing must possess pertinent knowledge of the following:

- Automation technology sector
- Dealing with dangerous voltages
- Using standards and regulations, in particular VDE, accident prevention regulations and regulations concerning special ambient conditions (e.g. areas potentially endangered by explosive materials, heavy pollution or corrosive influences).

1.6.4.3 Safety instructions

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variants and requirements associated with any particular installation, ABB cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by ABB with respect to use of information, circuits, equipment or software described in this manual. No liability is assumed for the direct or indirect consequences of the improper use, improper application or inadequate maintenance of these devices. In no event will ABB be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

PLC specific safety notices

The product family AC500 control system is designed according to EN 61131-2 IEC 61131-2 standards. Data, different from IEC 61131, are caused by the higher requirements of Maritime Services. Other differences are described in the technical data description of the devices.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

NOTICE!

PLC damage due to operation conditions

Protect the devices from dampness, dirt and damage during transport, storage and operation!

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

Cleaning instruction

Do not use cleaning agent for cleaning the device.

Use a damp cloth instead.

Connection plans and user software must be created so that all technical safety aspects, legal regulations and standards are observed. In practice, possible shortcircuits and breakages must not be able to lead to dangerous situations. The extent of resulting errors must be kept to a minimum.

Do not operate devices outside of the specified, technical data!

Trouble-free functioning cannot be guaranteed outside of the specified data.

NOTICE!

PLC damage due to missing grounding

– Ensure to earth the devices.
– The grounding (switch cabinet grounding, PE) is supplied both by the mains connection (or 24 V supply voltage) and via DIN rail. The DIN rail must be connected to the ground before the device is subjected to any power. The grounding may be removed only if it is certain that no more power is being supplied to the control system.

In the description for the devices (operating manual or AC500 system description), reference is made at several points to grounding, galvanic isolation and EMC measures. One of the EMC measures consists of discharging interference voltages into the grounding via Y-type capacitors. Capacitor discharge currents must basically be able to flow off to the grounding (in this respect, see also VBG 4 and the relevant VDE regulations).

CAUTION!

Do not obstruct the ventilation for cooling!

The ventilation slots on the upper and lower side of the devices must not be covered.

CAUTION!

Run signal and power wiring separately!

Signal and supply lines (power cables) must be laid out so that no malfunctions due to capacitive and inductive interference can occur (EMC).

WARNING!

Labels on or inside the device alert people that dangerous voltage may be present or that surfaces may have dangerous temperatures.

WARNING!

Splaying of strands can cause hazards!

During wiring of terminals with stranded conductors, splaying of strands shall be avoided.

– Ferrules can be used to prevent splaying.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

Information on batteries

CAUTION!

Use only ABB approved lithium battery modules!

At the end of the battery's lifetime, always replace it only with a genuine battery module.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

Environment considerations

Recycle exhausted batteries. Dispose batteries in an environmentally conscious manner, in accordance to local-authority regulations.

Environment and enclosure information

This equipment is intended for use in a Pollution Degree 2 industrial environment, in overvoltage Category II applications (as defined in IEC publication 60664-1), at altitudes up to 2.000 meters without derating.

This equipment is considered Group 1, Class A industrial equipment according to IEC/CISPR Publication 11. Without appropriate precautions, there may be potential difficulties ensuring electromagnetic compatibility in other environments due to conducted as well as radiated disturbance.

This equipment is supplied as "open type" equipment. It must be mounted within an enclosure that is suitably designed for those specific environmental conditions that will be present and appropriately designed to prevent personal injury resulting from accessibility to live parts. The interior of the enclosure must be accessible only by the use of a tool. Subsequent sections of this publication may contain additional information regarding specific enclosure type ratings that are required to comply with certain product safety certifications.

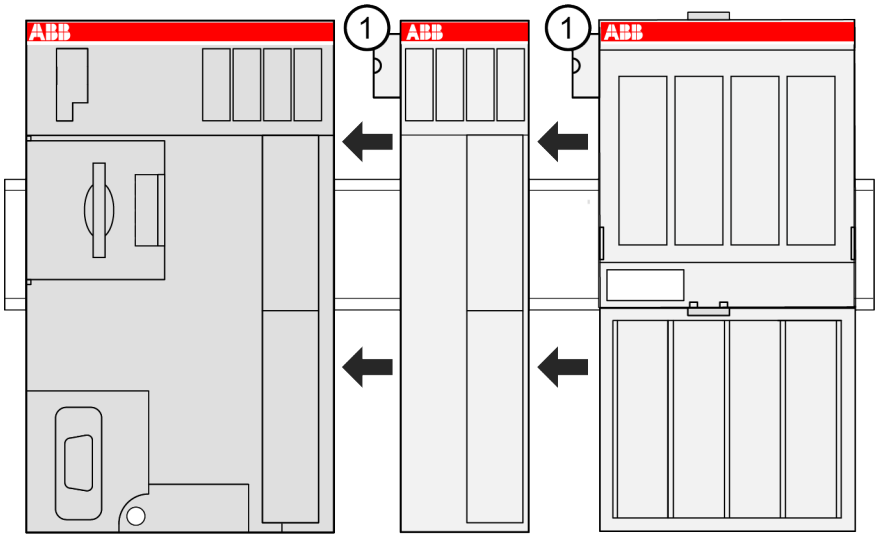
Refer to NEMA Standards publication 250 and IEC publication 60529, as applicable, for explanations of the degrees of protection provided by different types of enclosure. Also see the appropriate sections in this manual.

1.6.4.4 Overall information (valid for complete AC500 product family)

1.6.4.4.1 Serial I/O bus

The synchronized serial I/O bus is the I/O data bus for the I/O modules connected with the processor modules or communication interface modules. Through this bus, I/O and diagnosis data are transferred.

Up to 10 I/O terminal units (for 1 I/O module each) can be added to one terminal base or to one AC500-eCo processor module. The I/O terminal units and the AC500-eCo I/O modules, have a bus input at the left side and a bus output at the right side. Thus the length of the I/O bus increases with the number of attached I/O modules.



1 I/O bus connection

The connection of the I/O bus is performed automatically by telescoping the modules on the DIN rail. The I/O bus provides the following signals:

- Supply voltage of 3.3 V DC for feeding the electronic interface components
- 3 data lines for the synchronized serial data exchange
- several control signals

NOTICE!

The I/O bus is not designed for plugging and unplugging modules while in operation. If a module is plugged or replaced while the bus is in operation, the following consequences are possible

- reset of the station or of the CPU
- system lockup
- damage of the module

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

With its fast data transmission, the I/O bus obtains very low reaction times. Depending on the device and on the version of firmware and Automation Builder, the following numbers of I/O devices can be connected to the I/O bus.

Device	Version Automation Builder	Version firmware	Max. number of I/O devices
CANopen bus modules CI581-CN and CI582-CN	As of V2.1.0	All	0
PROFINET bus modules CI501-PNIO and CI502-PNIO	As of V2.1.0	all	10
EtherCAT communication interface module CI511-ETHCAT and CI512-ETHCAT	As of V2.1.0	As of V2.0.x	10

Table 590: General data

Parameter	Value
Supply voltage, signal level	3.3 V DC ± 10 %
Max. supply current	On request
Type of the data interface	Synchronized serial data exchange

Parameter	Value
Bus data transmission speed	1.8 Mb/s
Minimum bus cycle time	500 µs [1])
Galvanic isolation	I/O bus is galvanic connected to CPU and communication interface logic ciruits. Galvanic isolation of I/O bus is I/O module specific. See each module specification for details.
Protection against electrostatic discharge (ESD)	TB5xx, TB56xx: with protection diodes, no ESD discharge allowed on the port.
Max. bus length	1 m
[1]) Minimum bus cycle time: This value is valid for all module combinations (from 1 to 10 I/O modules)	

Table 591: Wiring (bus connection)

Parameter	Value
Bus connection	Left-side and right-side connection from module to module via a 10-pole HE plug (male at the left side, female at the right side)
Mechanical connection	Established by the terminal units
Max. bus length	1 m

1.6.4.4.2 Mechanical encoding

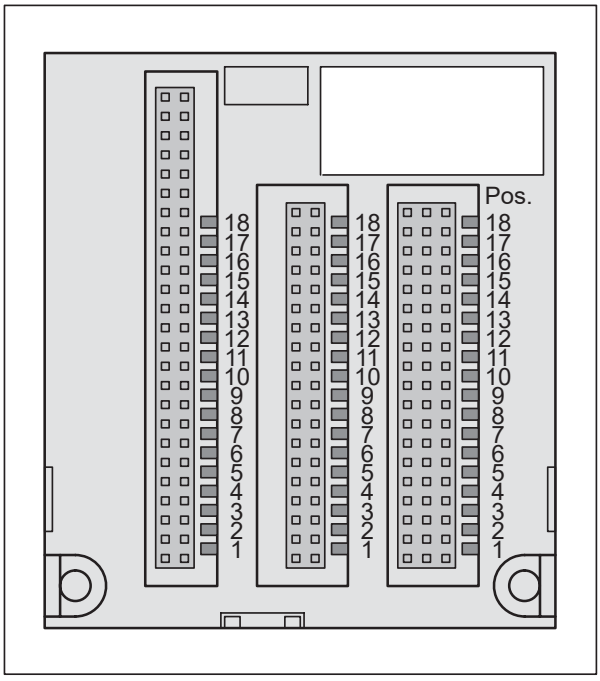


Fig. 263: Possible positions for mechanical encoding (1 to 18)

NOTICE!
Terminal units and terminal bases have a mechanical coding which prevents modules (from) being inserted into the wrong places for cases that might result in dangerous parasitic voltages or if modules could be destroyed.

The coding either makes it impossible to insert the module to the wrong place or blocks its electrical function (outputs are not activated).

The following figures show the possible encodings.

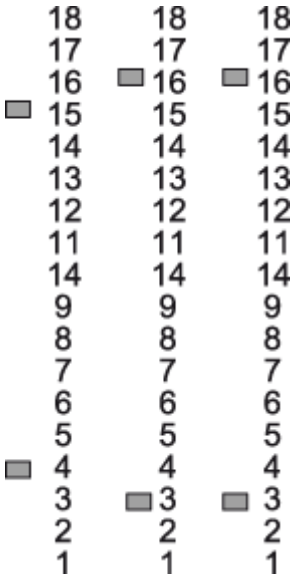


Fig. 264: Encoding for processor modules with Ethernet interface

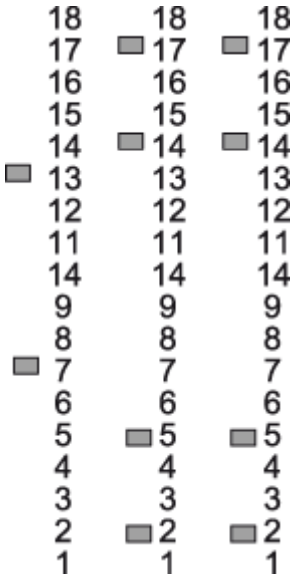


Fig. 265: Encoding for real-time Ethernet modules

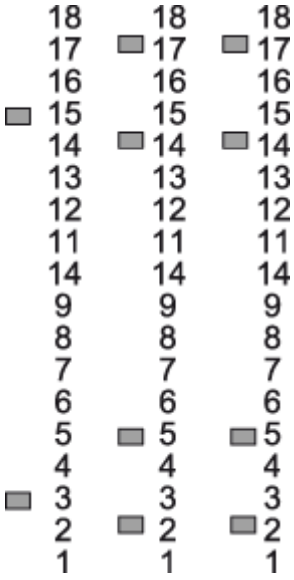


Fig. 266: Encoding for communication interface modules

Fig. 267: Encoding for I/O modules (24 VDC)

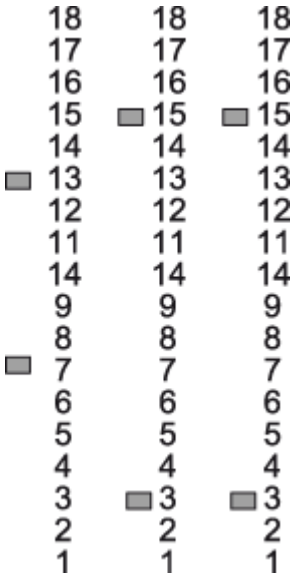


Fig. 268: Encoding for communication interface modules with PROFINET interface

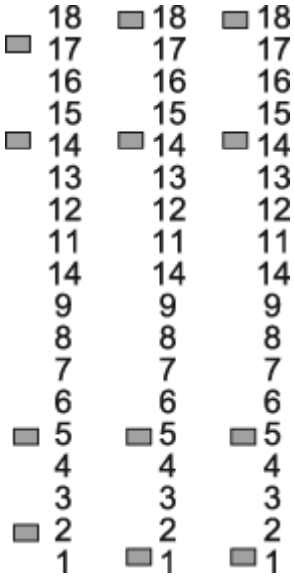


Fig. 269: Encoding for I/O modules (120 VAC / 230 VAC)

Fig. 270: Encoding for positioning modules

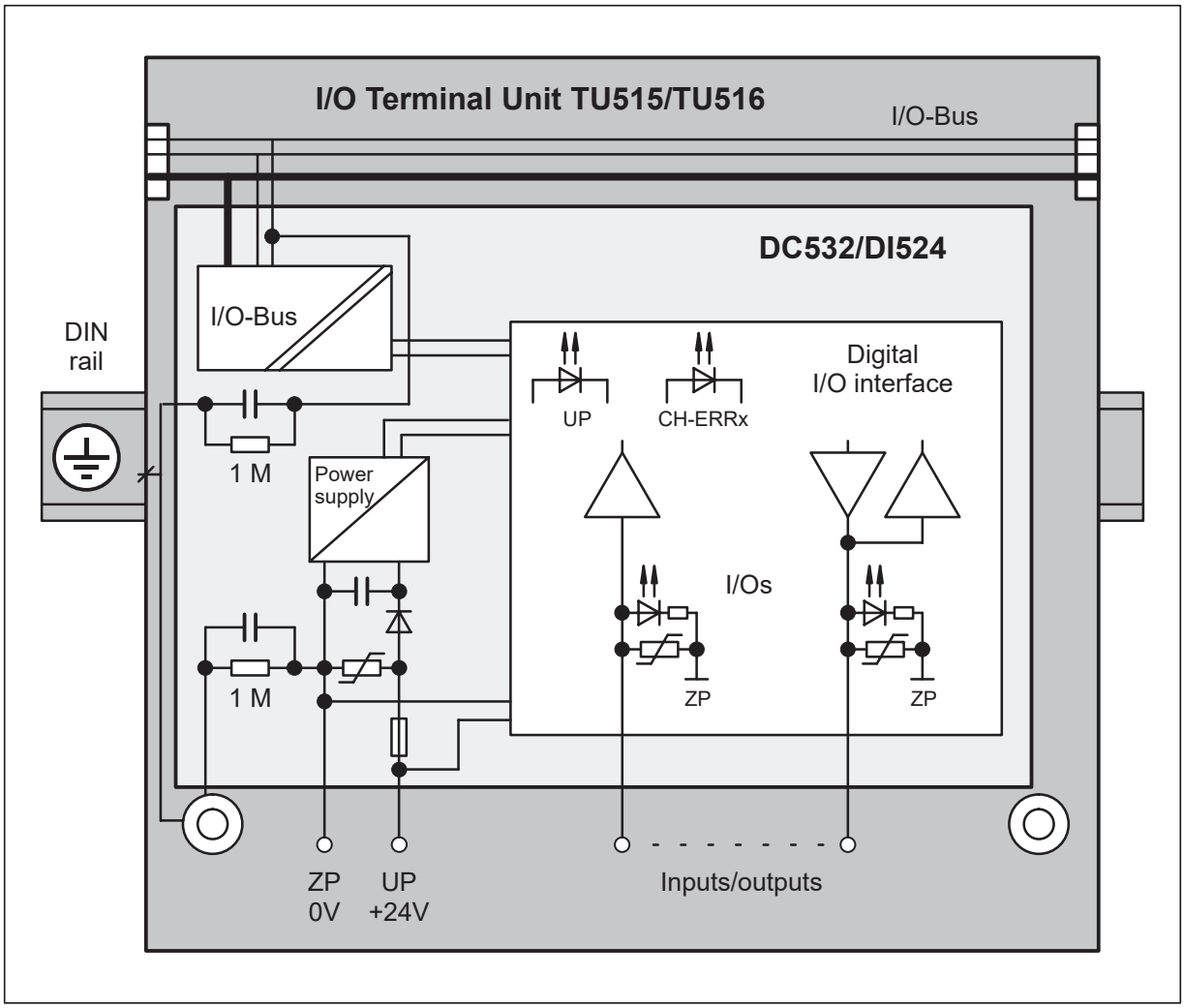
1.6.4.4.3 Earthing concept (Block diagrams)

NOTICE!

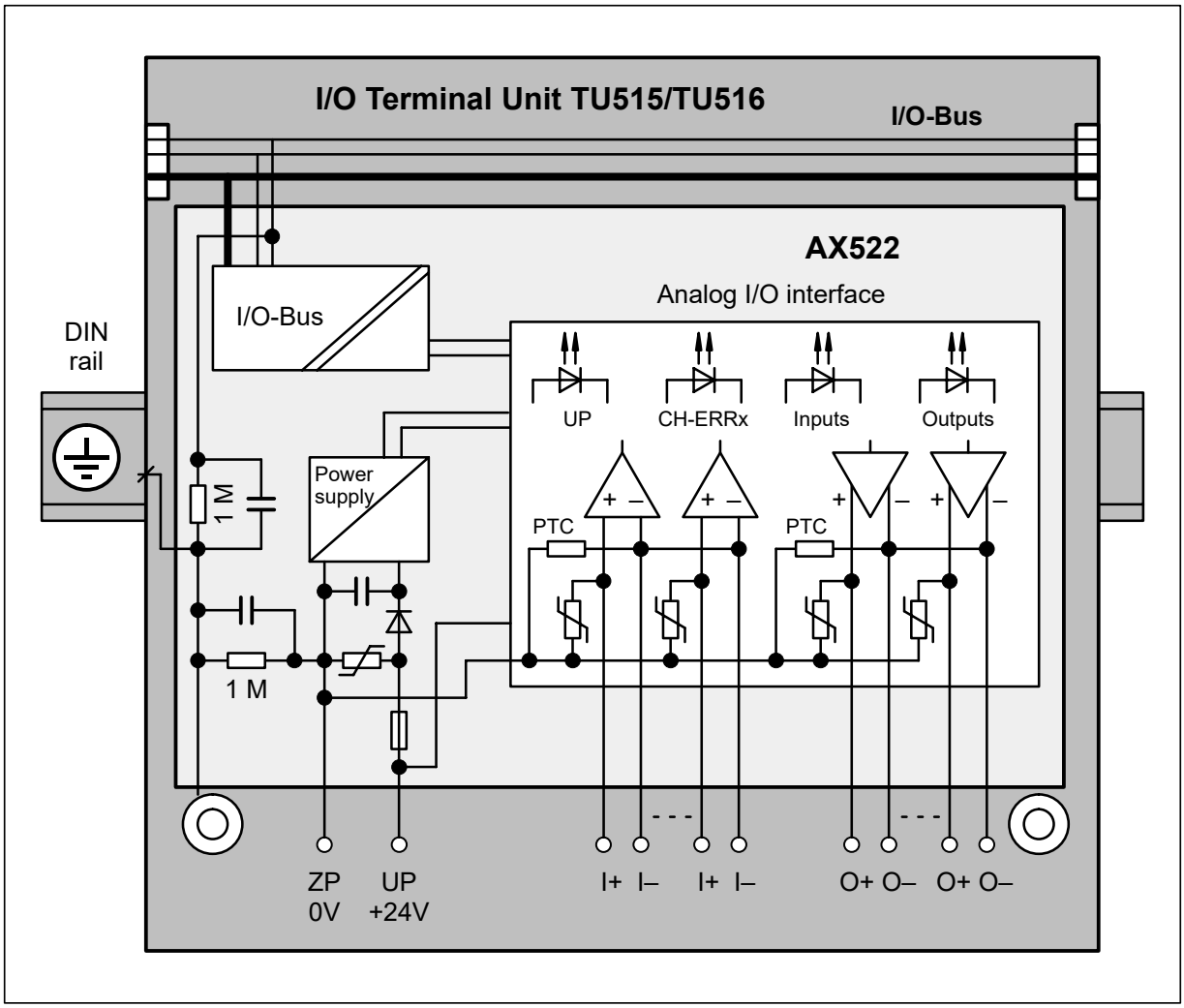
PLC damage due to missing grounding

- Ensure to earth the devices.
- The grounding (switch cabinet grounding, PE) is supplied both by the mains connection (or 24 V supply voltage) and via DIN rail. The DIN rail must be connected to the ground before the device is subjected to any power. The grounding may be removed only if it is certain that no more power is being supplied to the control system.

**Block diagram:
Digital I/O
modules**



Block diagram: Analog I/O modules



1.6.4.4.4 EMC-conforming assembly and construction

General principles

General considerations

Electric and electronical devices have to work correctly on site. This is also valid when electro-magnetic influences affect them in defined and/or expected strength. The devices themselves must not emit electro-magnetic noises.

Advant Controller components have a very high noise immunity.

When the wiring and grounding instructions are met, an error-free operation is given.

High electro-magnetic noises of nearby mounted applications must be taken in consideration during the planning phase.

An EMC compatible earthing concept will also guarantee an error-free operation here.

There are three important principles to be especially considered:

- *Keep all connections as short as possible (in particular the grounding conductors)*
- *Use large conductor cross sections (in particular for the grounding conductors)*
- *Create low-impedance, i.e. good and large-sized contacts (in particular for the grounding conductors)*

Pay attention to the following:

- *Use vibration-resistant connections*
- *Clean metallic contact areas*
- *Use solid plug and screw-type connections*
- *Use earth cable shields with clips on a well-grounded metallic surface*
- *Do not use aluminium parts*
- *Do not use sheath wires*
- *Do not use toothed lock washers under screw connections*

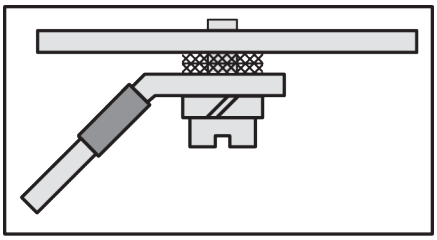


Fig. 271: Assembly: wrong

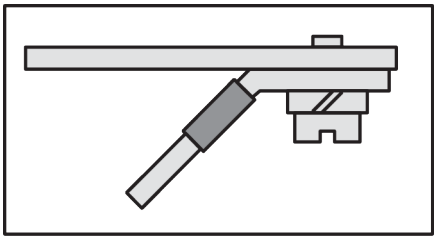


Fig. 272: Assembly: correct

Make a connection between the DIN rails and PE (Protective Earth). For this, use an grounding wire with a minimum conductor cross section of 10 mm^2.

The wire is connected to the DIN rail with an M6 screw.

A large-area contact of the DIN rail with the metallic mounting plate improves the EMC behavior significantly, as the disturbances can be discharged more effective.

Cable routing

- Route cables meeting the standards.
- Sort the cables into cable groups:
  - Power current cables
  - Power supply cables
  - Signal cables
  - Data cables

- Rout signal cables and data cables separately from the power cables.
  – Separate cable ducts or cable bundles.
  – The distance should be 20 cm or greater.
- Lay signal and data cables close to earthed surfaces.

Cable shields

- Use only shielded data cables. The shield should be grounded at both ends.
  A cable shield only grounded at one end can only protect from capacitively coupled interference and low-frequency disturbances (50 Hz hum).
- Avoid parasitic currents flowing through the cable shields.
  This can be done by installing current-carrying equipotential bondings.
- Use only cables with braided shields.
  Foil shields are not robust enough, cannot be contacted well and have poor HF properties.
- Use only metallic or [metal]-plated plugs for shielded data cables.
- Use only shielded cables for analog signals.
  For small signals ground the shield only at one end.
- Ground the cable shield directly with a clip when entering the switchgear cabinet.
  Do not cut the shield until the cable reaches the module connected.

The connection between the PE bar and the shield bar must have a low impedance.

Switchgear cabinet

According to DNV GL mounting in a seperate metall cabinet is required for:

- *SM560-S-FD-1*
- *CI521-MODTCP*
- *CI522-MODTCP*

Connections The connections between the switchgear cabinet, the mounting plates, the PE bar and the shield bar must have a low impedance.

Grounding Ground the switchgear cabinet doors with short and highly flexible conductors.

Illumination Only use filament lamps (bulbs) or fluorescent tubes with interference suppression.

For supplying the PC Use the mains socket which is located inside the switchgear cabinet.

↳ *Chapter 1.6.4.6.2.1 “Switchgear cabinet assembly” on page 3403*

Reference potential

- Provide a uniform reference potential in the entire installation and ground all electrical appliances if possible.
- Route your grounding conductors in a star configuration so that no ground loops can occur.

Equipotential bonding

The Installation of equipotential bondings are necessary if there are present or expected potential differences between parts of your application.

- *The impedance of equipotential bonding must be equal or lower than 10 % of the shield impedance of the shielded signal cables between the same points.*
- *The conductor cross section of a equipotential bonding must be 16 mm² to withstand the maximum possible compensating current.*
- *Equipotential bondings and shielded signal cables should be laid close to each other.*
- *Equipotential bondings must be connected to PE with low impedance.*

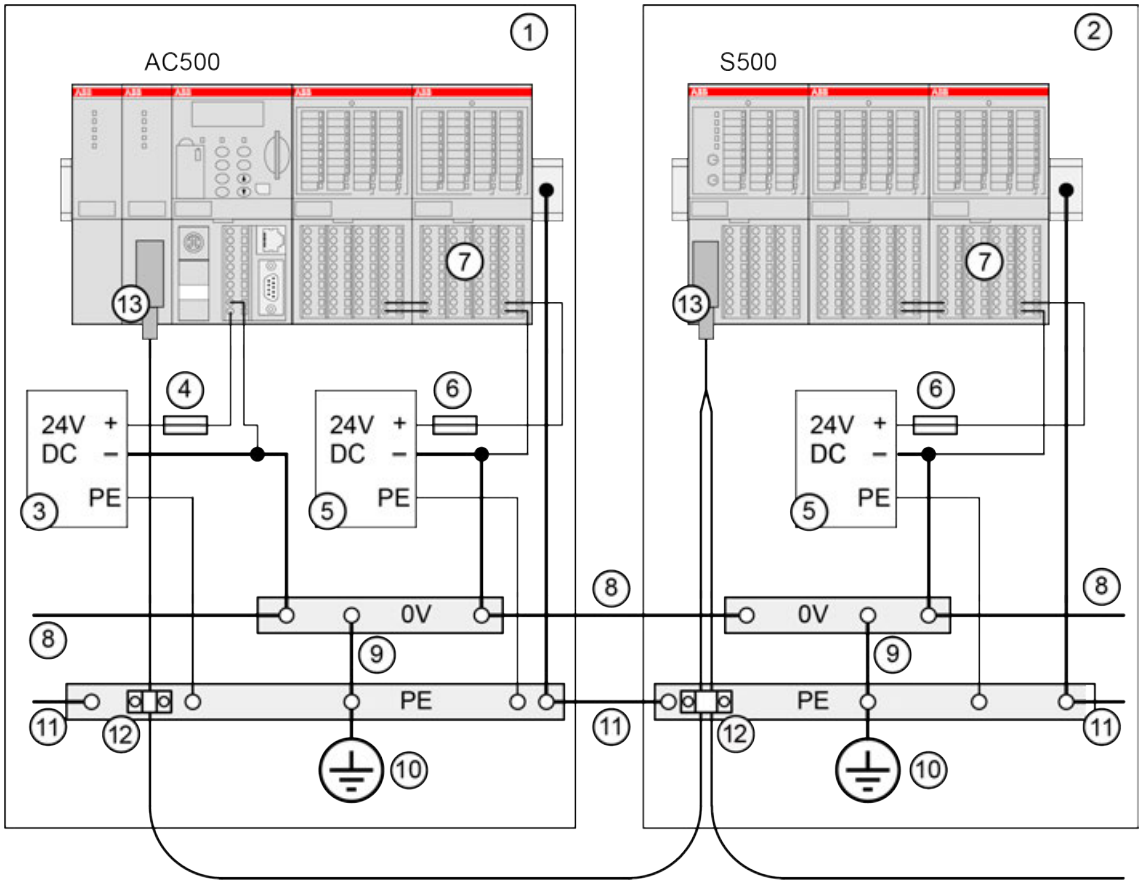


Fig. 273: AC500, equipotential bonding

1 Cabinet 1
2 Cabinet 2
3 Power supply for the CPU
4 Fuse for the CPU power
5 Power supply for the I/Os
6 Fuse for the I/O power
7 For fuses for the contacts of the relay outputs
8 0V rail
9 Grounding of the 0V rail
10 Cabinet grounding
11 Equipotential bonding between the cabinets min. 16 mm^2
12 Cable shields grounding
13 Fieldbus connection (e.g. Ethernet)

1.6.4.4.5 Power consumption of an entire station

The power consumption of a complete station consists of the sum of all individual consumptions.

- Consumers over terminals L+ and M on the AC500 terminal base/AC500-eCo CPU:
  - CPU itself
  - I/O modules attached on the I/O bus
  - Communication modules attached (AC500 terminal base)
- Consumers over the process supply voltage terminals ZP and UP of the AC500 terminal units / the L+/M or UP/ZP terminals of the AC500-eCo I/O modules:
  - Digital I/O modules
  - Analog I/O modules

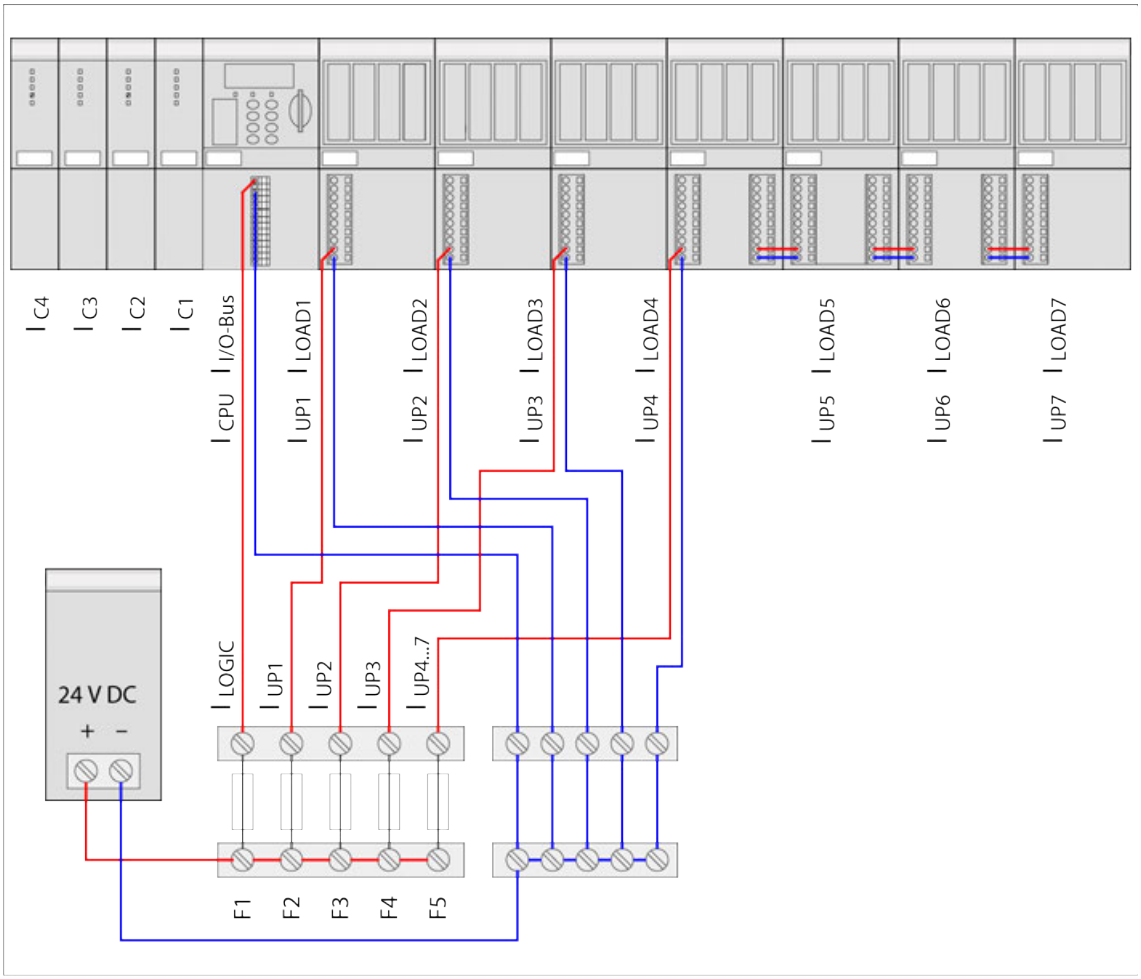
The two supply voltages can be provided by the same power supply unit. The CPU and the I/O modules should, however, be fused separately. Of course also separate power supplies are possible.

Calculation of the total current consumption

Example

In the example, the AC500 control system consists of the following devices:

- AC500 CPU with Ethernet interface
- 4 communication modules
- 7 I/O modules (digital and analog)
- As well as the required terminal bases and terminal units



Because of the high total current consumption of the digital I/O modules (from UP = 24 V DC), the supply is divided up into several electric circuits fused separately.

The maximum permitted total current over the supply terminals of the I/O terminal units is 8 A.

The total current can be calculated as follows:

$I_{Total} = I_{LOGIC} + I_{UP}$

with the assumptions

$I_{LOGIC} = I_{CPU} + I_{I/O\ bus} + I_{C1} + I_{C2} + I_{C3} + I_{C4}$ (CPU + communication modules + I/O bus)

$I_{I/O\ bus}$ = Number of expansion modules × Current consumption through the I/O bus per module

and

$I_{UP} = I_{UP1} + I_{LOAD1} + I_{UP2} + I_{LOAD2} + I_{UP3} + I_{LOAD3} + I_{UP4} + I_{LOAD4} + I_{UP5} + I_{LOAD5} + I_{UP6} + I_{LOAD6} + I_{UP7} + I_{LOAD7}$

If one assumes that all outputs are switched on and are operated with their maximum permitted load currents (under compliance with the maximum permitted currents at the supply terminals), then the following values are the result for an example shown above:

	I_{CPU} *)	I_{Cx} *)	$I_{I/O\ bus}$ *)	I_{UPx} *)	I_{LOADx} *)
CPU / communication module part					
CPU	0.110 A	-	-	-	-
C1	-	0.050 A	-	-	-
C2	-	0.085 A	-	-	-
C3	-	0.050 A	-	-	-
C4	-	0.050 A	-	-	-
I/O module part					
Analog1	-	-	0.002 A	0.150 A	-
Analog2	-	-	0.002 A	0.150 A	0.160 A
Analog3	-	-	0.002 A	0.100 A	0.080 A
Analog4	-	-	0.002 A	0.100 A	0.080 A
Digital1	-	-	0.002 A	0.050 A	8.000 A
Digital2	-	-	0.002 A	0.050 A	8.000 A
Digital3	-	-	0.002 A	0.050 A	8.000 A
Σ columns	0.110 A	0.235 A	0.014 A	0.650 A	24.320 A
	$\Sigma\ I_{LOGIC} \approx 0.4$ A			$\Sigma\ I_{UP} \approx 25$ A	
	$I_{Total} \approx 25.4$ A				
*) All values in this column are exemplary values					

Dimensioning of the fuses

To be able to select the fuses for the station correctly, both the current consumption and the inrush currents (melting integral for the series-connected fuse) must be taken into consideration.

Fuse	for	Σ of the melting integrals in A^2s	$I_{Logic\ A}$	$I_{UPx\ A}$	Recommended fuse	
					Type	Value
F1	CPU logic	1.000	≈ 0.4	-	Quick	10 A
F2	Module Digital1	0.005	-	8.050	Quick	10 A
F3	Module Digital2	0.008	-	8.050	Quick	10 A
F4	Module Digital3	0.007	-	8.050	Quick	10 A
F5	Modules Analog1 + Analog2 + Analog3 + Analog4	0.130	-	0.820	Quick	10 A

1.6.4.4.6 Decommissioning

Secure decommissioning of a functional CPU

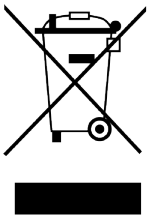
1. Delete the runtime licenses ↳ *Chapter 1.6.6.2.2.2.4 “Returning a license” on page 3671*.
2. Delete certificates available on the CPU ↳ *Chapter 1.8.2.4.1 “View 'Security Screen' - 'Devices'” on page 4125*.
3. Delete applications ↳ *Chapter 1.4.1.20.3.6.12 “Command 'Reset Origin'” on page 1039* ↳ *Chapter 1.4.1.20.3.6.13 “Command 'Reset Origin Device'” on page 1040*.
4. Delete applications from memory card, if available ↳ *Chapter 1.4.1.14 “Copying files to/ from PLC” on page 441*.
5. If available, remove memory card and battery from CPU.
6. Delete all user accounts and user data ↳ *“Tab 'User'” on page 996*.
7. Demount and dispose the hardware modules ↳ *Chapter 1.6.4.5.3 “Mounting and demounting” on page 3360* ↳ *Chapter 1.6.4.6.3 “Mounting and demounting” on page 3408* ↳ *Chapter 1.6.4.4.7 “Recycling” on page 3352*.

Secure decommissioning of a not functional CPU

▷ If you can not access the data stored in the CPU, e.g., because the CPU is not functional any more, then physically destroy the device.

⇨ This ensures that the credentials that are stored in the device, can not be misused.

1.6.4.4.7 Recycling



Disposal and recycling information

This symbol on the product (and on its packaging) is in accordance with the European Union's Waste Electrical and Electronic Equipment (WEEE) Directive.

The symbol indicates that this product must be recycled/disposed of separately from other household waste.

It is the end user's responsibility to dispose of this product by taking it to a designated WEEE collection facility for the proper collection and recycling of the waste equipment.

The separate collection and recycling of waste equipment will help to conserve natural resources and protect human health and the environment.

For more information about recycling, please contact your local environmental office, an electrical/electronic waste disposal company or the store where you purchased the product.

1.6.4.5 AC500-eCo

1.6.4.5.1 System data AC500-eCo V3

Environmental conditions

Table 592: Process and supply voltages

Parameter		Value
24 V DC		
	Voltage	24 V (-15 %, +20 %)
	Protection against reverse polarity	Yes
24 V AC		
	Voltage	24 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
100 V AC		
	Voltage	100 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
230 V AC		
	Voltage	230 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
100 V AC...240 V AC wide-range supply		
	Voltage	100 V...240 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
Allowed interruptions of power supply, according to EN 61131-2		
	DC supply	Interruption < 10 ms, time between 2 interruptions > 1 s, PS2

NOTICE!

Exceeding the maximum power supply voltage (> 30 V DC) for process or supply voltages could lead to unrecoverable damage of the system. The system might be destroyed.

Parameter				Value				
				PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH	PM5072-T-2ETHW
Temperature								
	Operating							
		Horizontal mounting						
			Standard temperature range	0 °C...+55 °C	0 °C...+60 °C			-
			Wide temperature range	-				-20 °C...+70 °C I/O derating in range 60 °C…70 °C: 75 %
		Vertical mounting (output load reduced to 50 % per group)						
			Standard temperature range	0 °C...+40 °C				-
			Wide temperature range	-				-20 °C...+40 °C
	Storage			-40 °C...+70 °C				
	Transport			-40 °C...+70 °C				
Humidity				Max. 95 %, without condensation				
Air pressure								
	Operating			> 800 hPa / < 2000 m				
	Storage			> 660 hPa / < 3500 m				
Ingress protection				PLC System: IP 20 in accordance with IEC 60529 ● with all modules or option boards plugged in ● with all terminal blocks plugged in ● with all covers closed				

Option boards	Temperature range
TA5101-4DI	0 °C... 60 °C
TA5105-4DOT	0 °C... 60 °C
TA5110-2DI2DOT	0 °C... 60 °C
TA530-KNXPB	0 °C... 60 °C
TA5131-RTC	0 °C...+55 °C
TA5141-RS232I	0 °C... 60 °C
TA5142-RS485I	0 °C... 60 °C
TA5142-RS485	0 °C... 60 °C

Creepage distances and clearances

The creepage distances and clearances meet the requirements of the overvoltage category II, pollution degree 2.

Power supply units

For the supply of the modules, power supply units according to SELV or PELV specifications must be used.

Safety Extra Low Voltage (SELV) and Protective Extra Low Voltage (PELV)

To ensure electrical safety of AC500/AC500-eCo extra low voltage circuits, 24 V DC supply, communication interfaces, I/O circuits, and all connected devices must be powered from sources meeting requirements of SELV, PELV, class 2, limited voltage or limited power according to applicable standards.

WARNING!

Improper installation can lead to death by touching hazardous voltages!

To avoid personal injury, safe separation, double or reinforced insulation and separation of the primary and secondary circuit must be observed and implemented during installation.

- Only use power converters for safety extra-low voltages (SELV) with safe galvanic separation of the primary and secondary circuit.
- Safe separation means that the primary circuit of mains transformers must be separated from the secondary circuit by double or reinforced insulation. The protective extra-low voltage (PELV) offers protection against electric shock.

Electromagnetic compatibility

Electromagnetic Compatibility		
Device suitable for:		
	Industrial applications	Yes
	Domestic applications	Yes
Immunity against electrostatic discharge (ESD):		According to IEC 61000-4-2, zone B, criterion B
	Electrostatic voltage in case of air discharge	8 kV
	Electrostatic voltage in case of contact discharge	6 kV
	ESD with communication connectors	In order to prevent operating malfunctions, it is recommended, that the operating personnel discharge themselves prior to touching communication connectors or perform other suitable measures to reduce effects of electrostatic discharges.
Immunity against the influence of radiated (CW radiated):		According to IEC 61000-4-3, zone B, criterion A
	Test field strength	10 V/m
Immunity against transient interference voltages (burst):		According to IEC 61000-4-4, zone B, criterion B
	Supply voltage units (DC)	2 kV
	Digital inputs/outputs (24 V DC)	1 kV

Electromagnetic Compatibility		
	Digital inputs/outputs (100 V AC...240 V AC)	Relay 2 kV
	Ethernet	1 kV
	Serial interfaces	1 kV
Immunity against the influence of line-conducted interferences (CW conducted):		According to IEC 61000-4-6, zone B, criterion A
Test voltage		10 V pass A
High energy surges		According to IEC 61000-4-5, zone B, criterion B
	Power supply DC	1 kV CM / 0.5 kV DM [1])
	DC I/O supply	1 kV CM / 0.5 kV DM [1])
	Ethernet	1 kV CM [1])
	Serial interfaces	1 kV CM [1])
	AC I/O unshielded	2 kV CM, 1 kV DM [1])
	I/O analog, I/O DC unshielded	1 kV CM [1])
Radiation (radio disturbance)		According to IEC 55011, group 1, class A

[1]) CM = Common Mode, DM = Differential Mode

Mechanical data

Parameter	**Value**
Mounting	Horizontal
Degree of protection	EN61131-2: IP20 with all option boards or option board slot covers attached (and all terminal screws are tightened)
Housing	Classification V0 according to UL 94
Vibration resistance acc. to EN 61131-2	all three axes (DIN rail mounting) 5 Hz...8.2 Hz: ±7.5 mm peak 8.2 Hz...150 Hz: 2 g peak
Shock test	All three axes 15 g, 11 ms, half-sinusoidal
Mounting of the modules:	
DIN rail according to DIN EN 50022	35 mm, depth 7.5 mm or 15 mm
Mounting with screws	M3
Fastening torque	1.2 Nm

Approvals and certifications

Information on approvals and certificates can be found in the corresponding chapter of the *Main catalog, PLC Automation*.

1.6.4.5.2 Mechanical dimensions

Switchgear cabinet assembly (indoor use)

> *Information on EMC-conforming assembly and construction is provided within the overall functions section ↳ Chapter 1.6.4.4.4 “EMC-conforming assembly and construction” on page 3345.*

PLC enclosure

To protect PLCs against:

- unauthorized access,
- dusting and pollution,
- moisture and wetness and
- mechanical damage,

switchgear cabinet IP54 for common dry factory floor environment is suitable.

Maintain spacing from:

- enclosure walls
- wireways
- adjacent equipment

Allow a minimum of 20 mm clearance on all sides. This provides ventilation and galvanic isolation.

It is recommended to mount the modules on an grounded mounting plate, or an grounded DIN rail, independent of the mounting location.

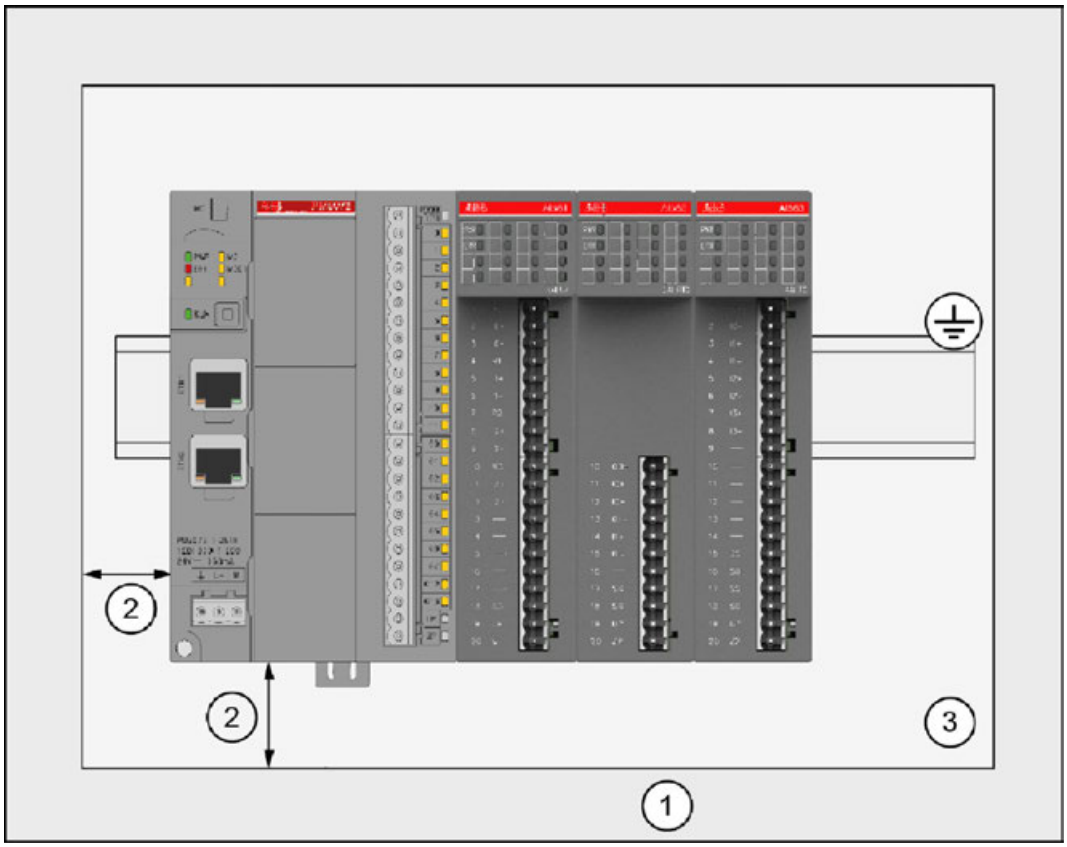


Fig. 274: Installation of AC500-eCo V3 CPU/S500 modules in a switchgear cabinet

1 Cable duct
2 Distance from cable duct ≥20 mm
3 Mounting plate, grounded

NOTICE!

Horizontal mounting is highly recommended.

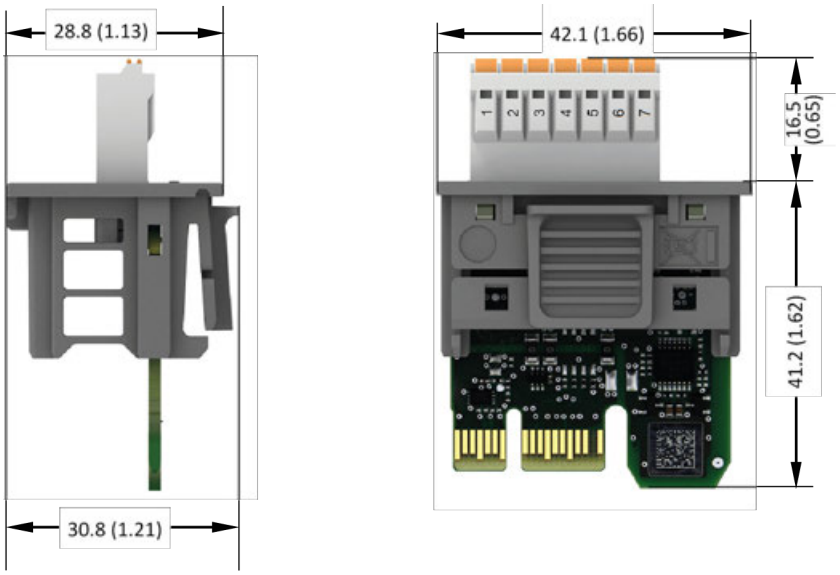
Vertical mounting is possible, however, derating consideration should be made to avoid problems with poor air circulation and overheating.

When vertically mounted, always place an end-stop terminal block (e.g. type BADL, P/N: 1SNA399903R0200) on the bottom and on the top of the modules to properly secure the modules.

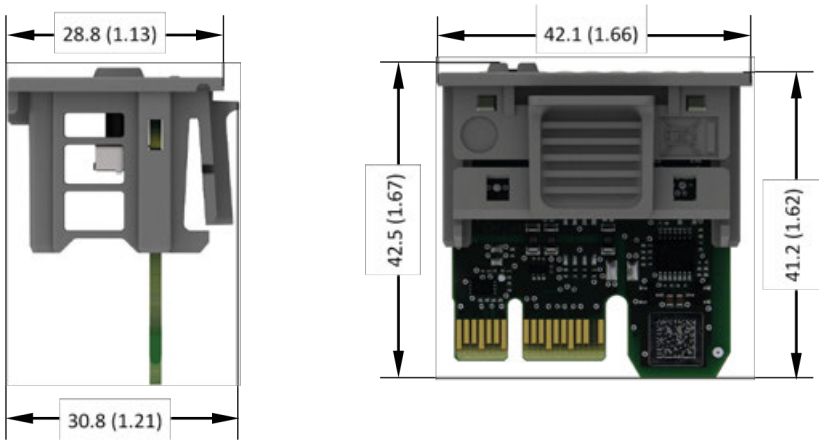
With high vibration applications and horizontal mounting, we also recommend to place end-stop terminals at the right and left side of the device to properly secure the modules, e.g. type BADL, P/N: 1SNA399903R0200.

Mechanical dimensions AC500-eCo V3 option boards

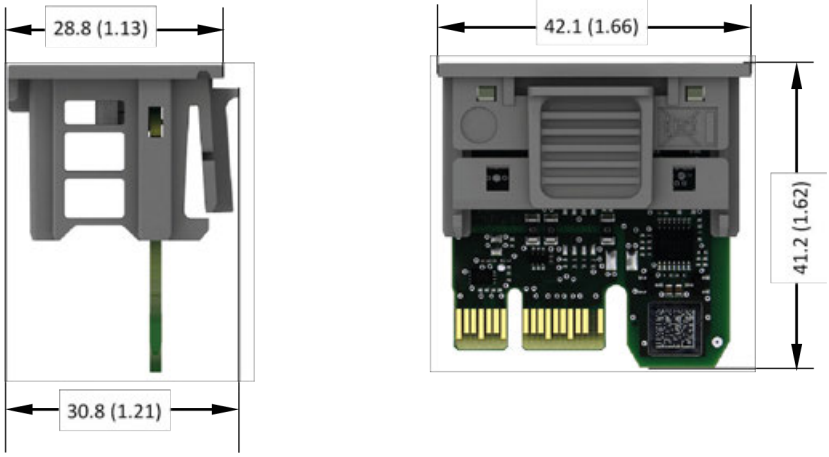
TA5105, TA5110



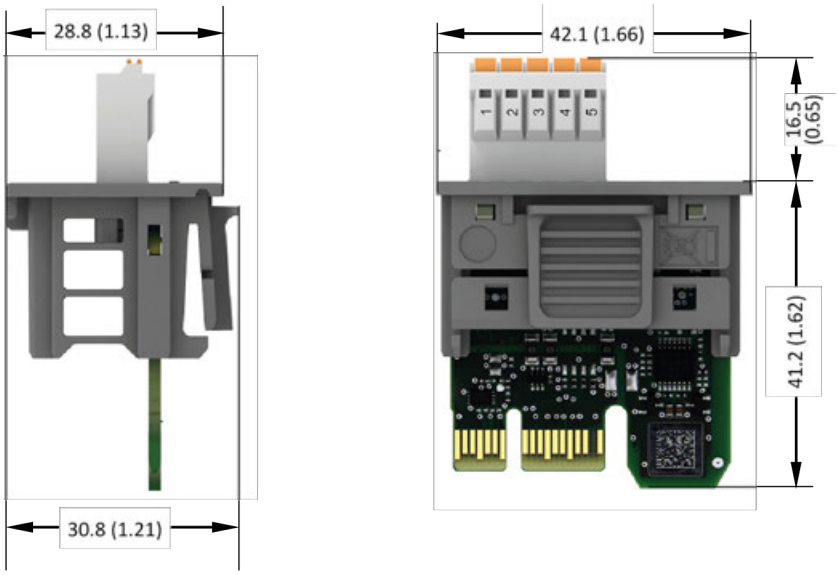
TA5130



TA5131-RTC



TA5101, TA514x



Mechanical dimensions AC500-eCo V3

All mechanical dimensions are given in millimeters and inches. The value in brackets is the inch-value.

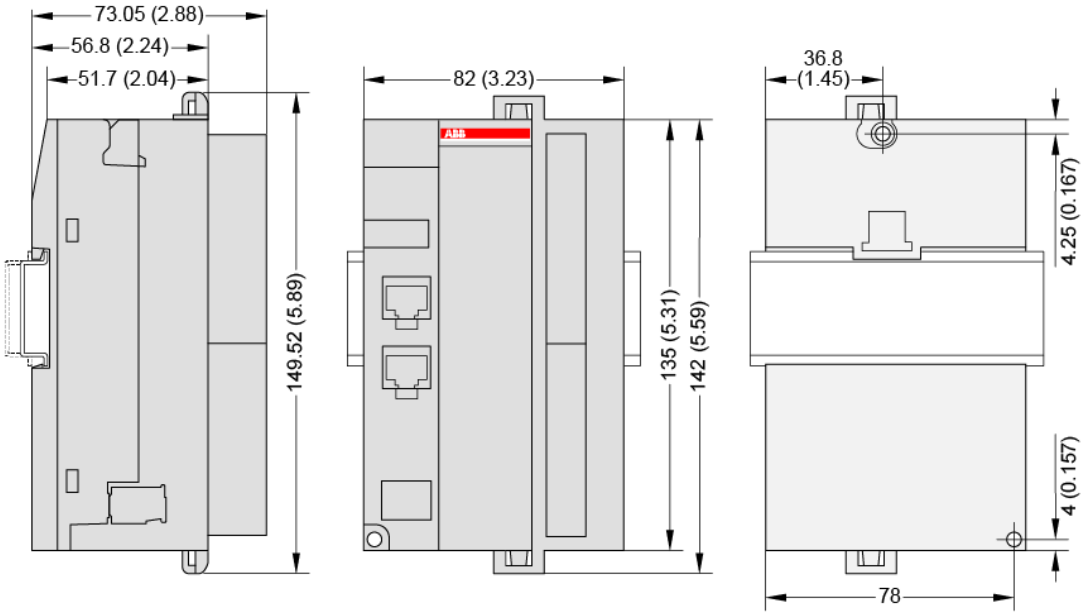


Fig. 275: Side, front and back view

Mechanical dimensions S500-eCo

> *All mechanical dimensions are given in millimeters and inches. The value in brackets is the inch-value.*

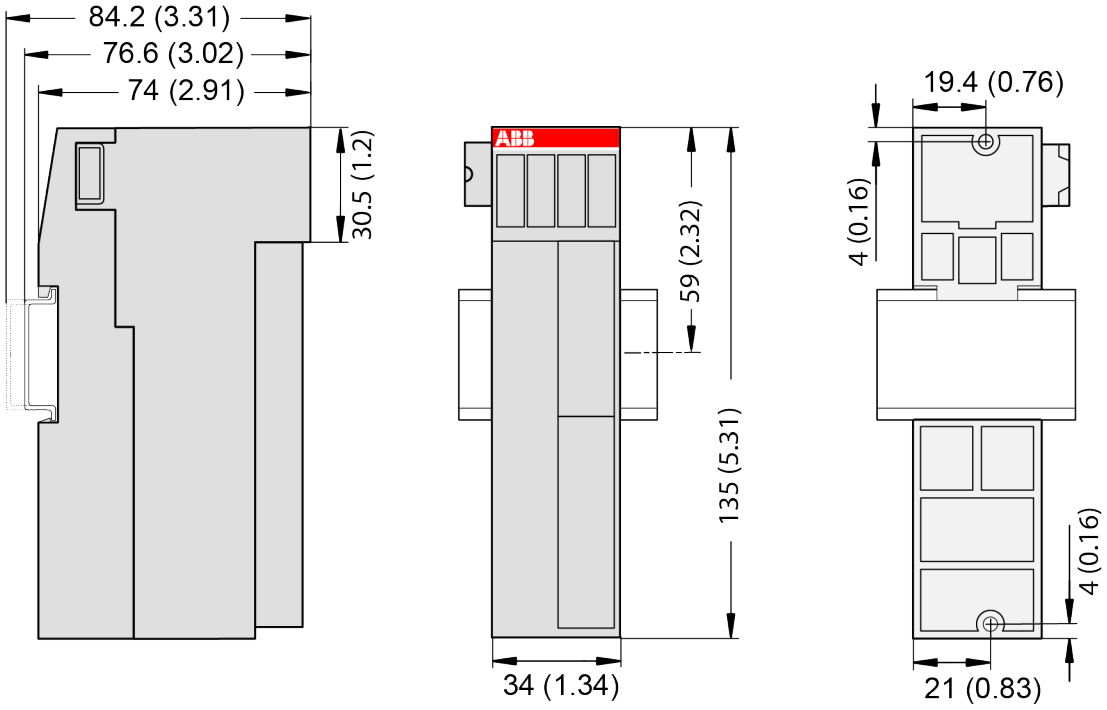


Fig. 276: Side, front and back view

1.6.4.5.3 Mounting and demounting

The control system is designed to be mounted to a well-grounded mounting surface such as a metal panel. Additional grounding connections from the mounting tabs or DIN rail (if used), are not required unless the mounting surface cannot be grounded.

During panel or DIN rail mounting of all devices, be sure that all debris (metal chips, wire strands, etc.) is kept from falling into the controller. Debris that falls into the controller could cause damage while the controller is energized.

All devices are grounded through the DIN rail to chassis ground. Use zinc plated yellow-chromate stell DIN rail to assure proper grounding. The use of other DIN rail materials (e.g. aluminium, plastic, etc.) that can corrode, oxidize, or are poor conductors, can result in improper or intermittent grounding.

Mounting and demounting of the AC500-eCo V3 CPUs

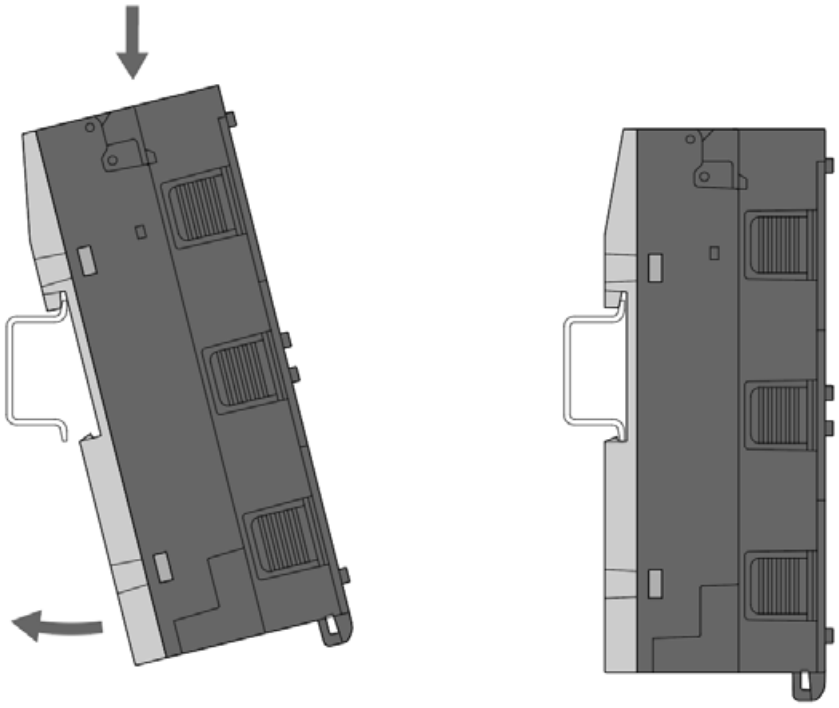
Mounting a processor module on a DIN rail

NOTICE!

Risk of function faults!

The processor module is grounded via DIN rail.

The DIN rail must be included into the earthing conception of the plant.

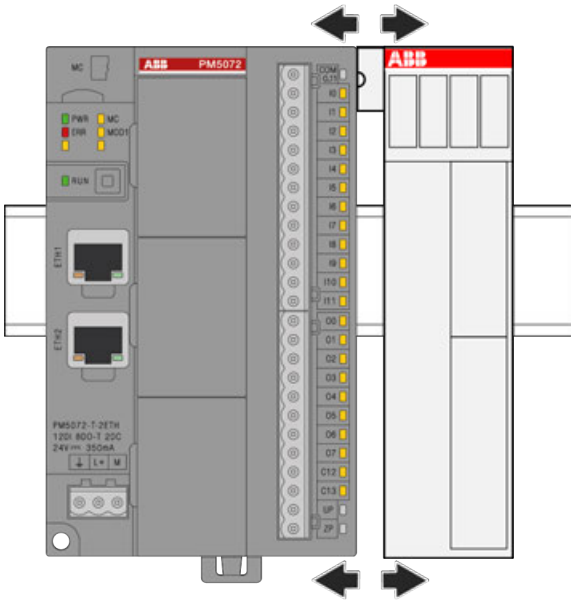


Mount the processor module at the top of the DIN rail, then snap it in below.

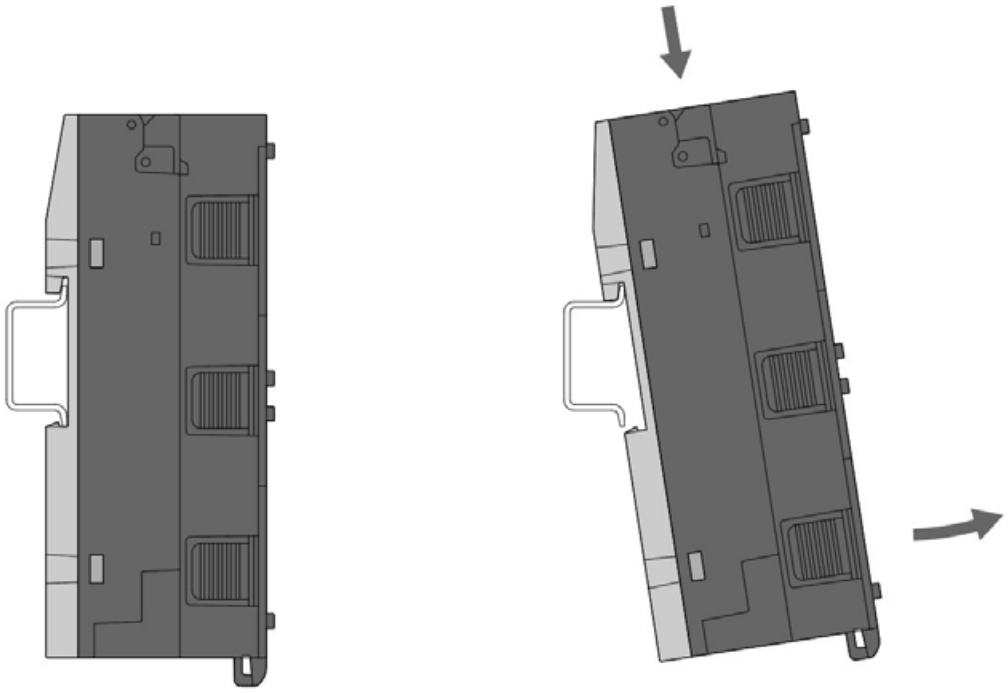
See hardware description of PM50xx ↳ Chapter 1.6.3.3.1.1 “PM50xx” on page 2440 for connection.

Demounting a processor module mounted on a DIN rail

1. Remove I/O modules if connected.



2. While pressing down processor module pull it away from DIN rail.

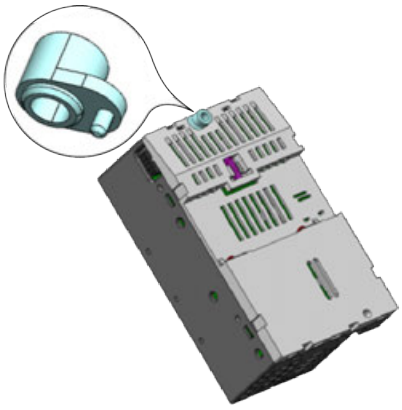


Mounting a processor module on a metal plate

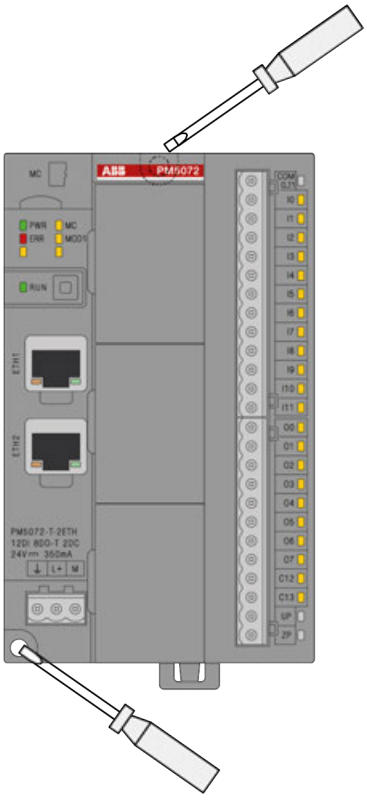
NOTICE!
Risk of function faults!
Missing electrical contact by isolating screws or washers!
Use metal screws on the metal plate.
The metal plate must be included into the earthing concept of the plant.
Do NOT use insulating washers!

One TA543 wall mounting accessory ⇳ Chapter 1.6.4.5.5.5 “TA543 - Screw mounting accessory” on page 3396 is needed per processor module.

1. Snap in the TA543 at the back side of the processor module.



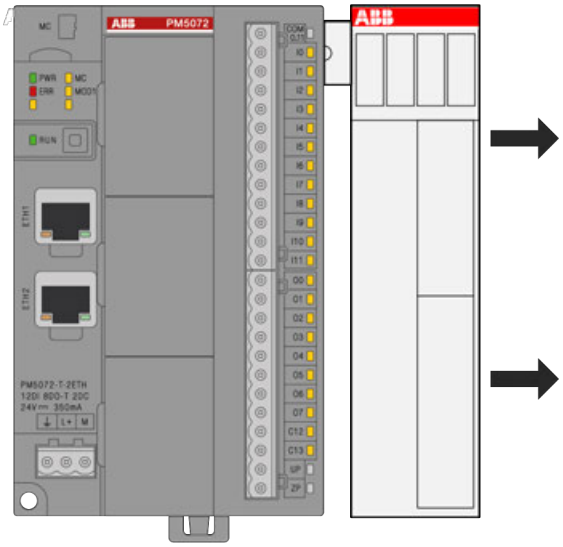
2. Fasten the processor module with two screws (max. diameter: 4 mm) to the metal plate.



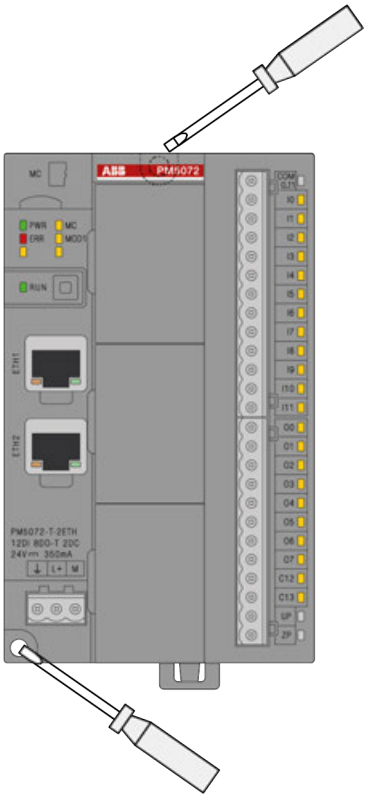
> *See hardware description of PM50xx ↳ Chapter 1.6.3.3.1.1 “PM50xx” on page 2440 for connection.*

Demounting a processor module mounted on a metal plate

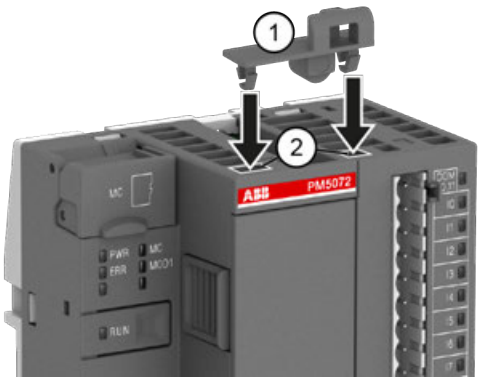
1. Remove I/O modules if connected.



2. Remove the 2 screws.



Mounting of TA5301-CFA



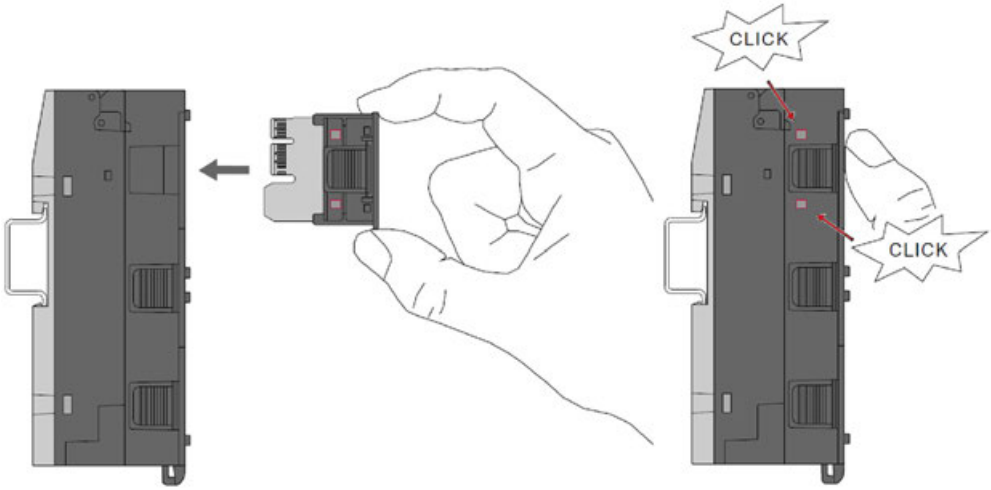
1 TA5301-CFA cable fixing accessory
2 2 openings on the PM50x2 processor module

▷ Insert the TA5301-CFA cable fixing accessory into the two openings on the PM50x2 processor module marked white in the figure.

Mounting and demounting option boards

Inserting the option board

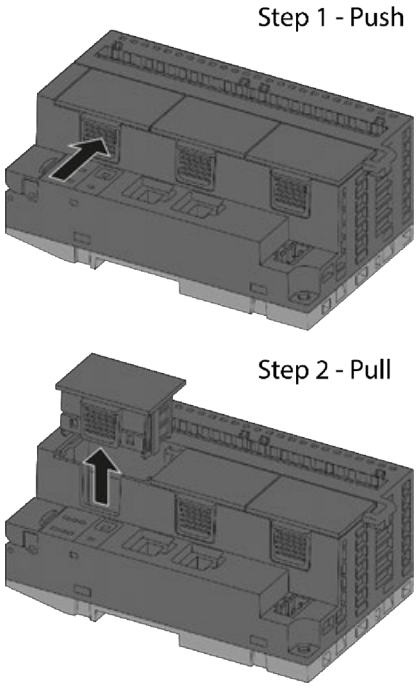
After mounting the PM50x2 processor module on the DIN rail, mount the option board.



▷ Press the option board TA51xx (or TA5300-CVR) into the slot of the processor module PM50x2 until it locks in place.

The option board must click into the slot of the processor module.

Removing the option board



1. Push the option board on the side to release the lock.
2. At the same time, pull the option board out of the slot.

CAUTION!
Risk of injury and damaging the product!
Always plug in the option board slot cover when the option board is not inserted.

If the option board slot cover is lost, please order the replacement TA5300-CVR (1SAP187500R0001).

Never power up the CPU with uncovered option board slot, otherwise it may cause serious injury and/or damage the product.

Mounting and demounting of S500-eCo I/O modules

S500-eCo I/O modules can be mounted either on a DIN rail or with screws on a metal plate.

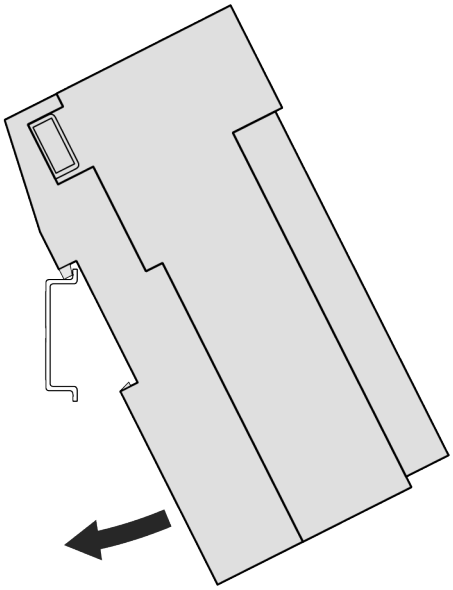
Mounting I/O modules on a DIN rail

NOTICE!
Risk of function faults!
The S500-eCo I/O modules are grounded via the DIN rail.

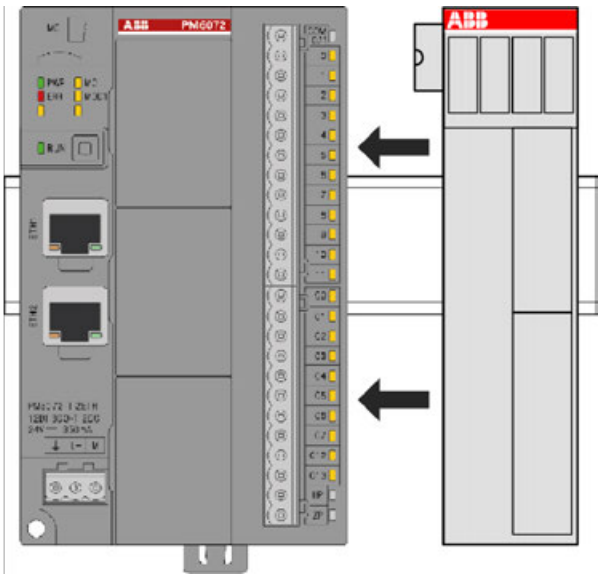
The DIN rail must be included into the earthing concept of the plant.

Use only metal screws.

1. Mount I/O module at the top of the DIN rail, then snap it in below.

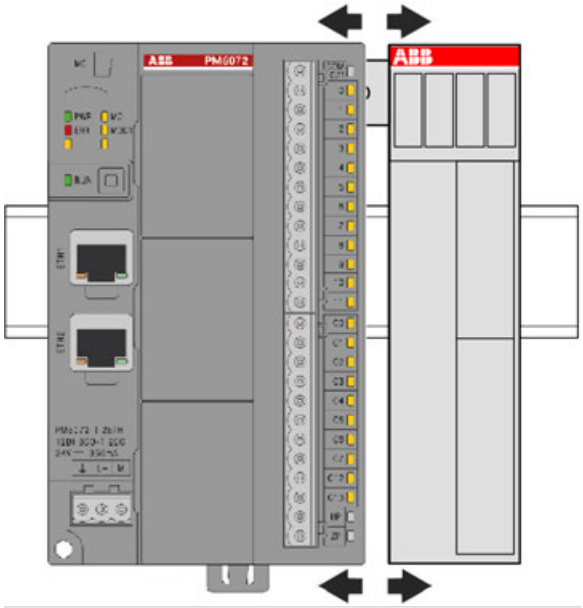


2. Attach I/O module by hand to an other module. The serial I/O bus is connected automatically.

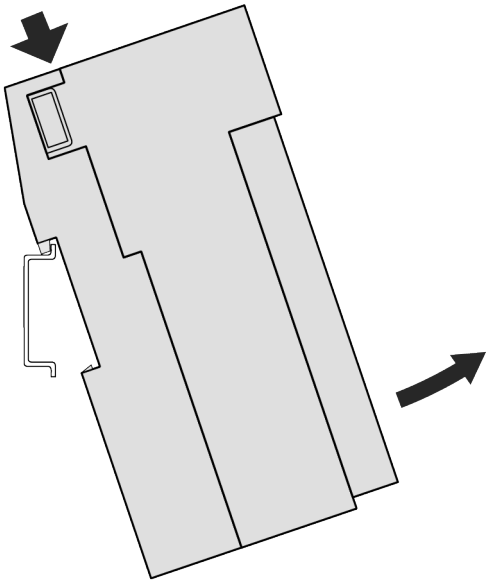


Demounting I/O modules mounted on a DIN rail

1. Remove I/O module by hand if connected.



2. While pressing down I/O module pull it away from DIN rail.



Mounting I/O modules on a metal plate

NOTICE!
Risk of function faults!
Missing electrical contact by isolating screws or washers!

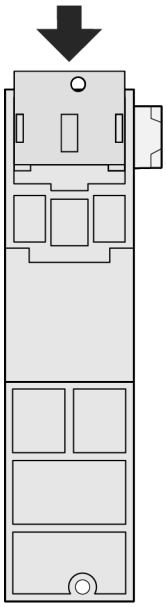
Use metal screws on the metal plate.

The metal plate must be included into the earthing concept of the plant.

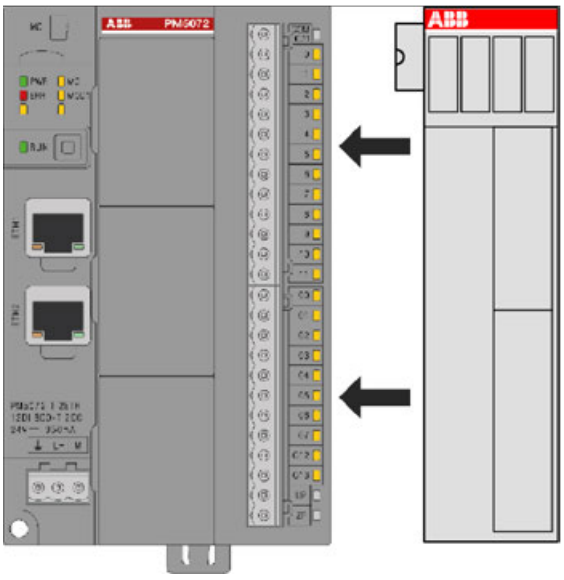
Do NOT use insulating washers!

> *One TA566 wall mounting accessory ↳ Chapter 1.6.4.5.5.6 “ TA566 - Wall mounting accessory” on page 3397 is needed per S500-eCo I/O module.*

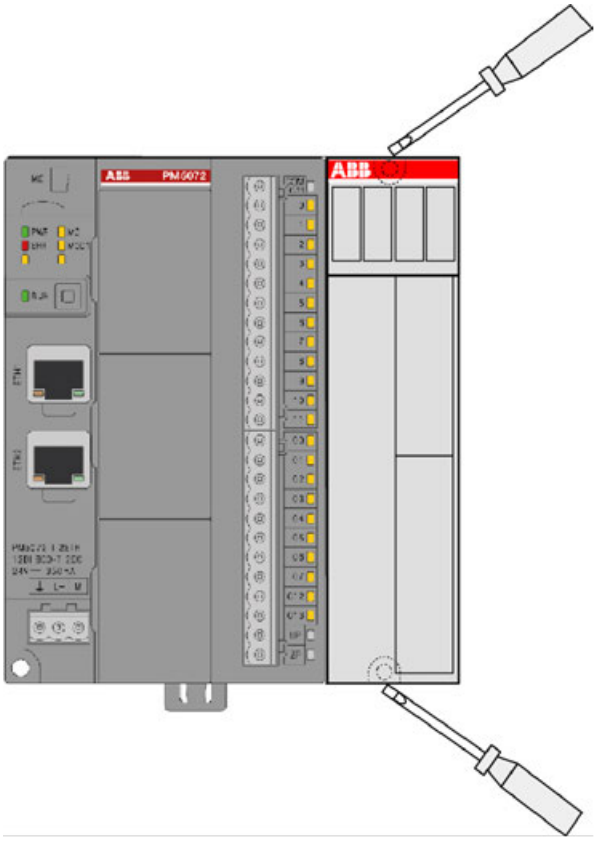
1. Snap in the TA566 at the back side of the I/O module.



2. Attach the I/O module by hand to an other module. The serial I/O bus is connected automatically.

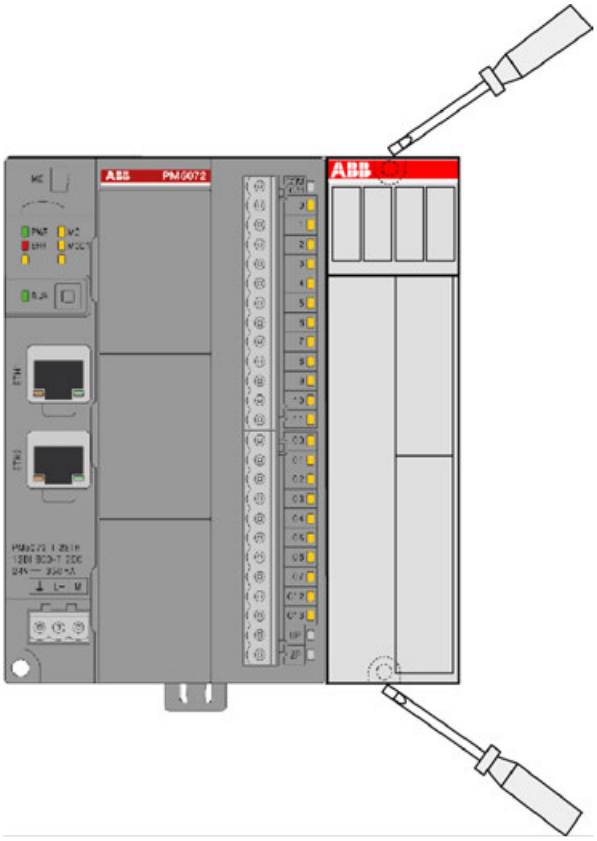


3. Fasten the I/O module with two screws (max. diameter: 4 mm) to the metal plate.

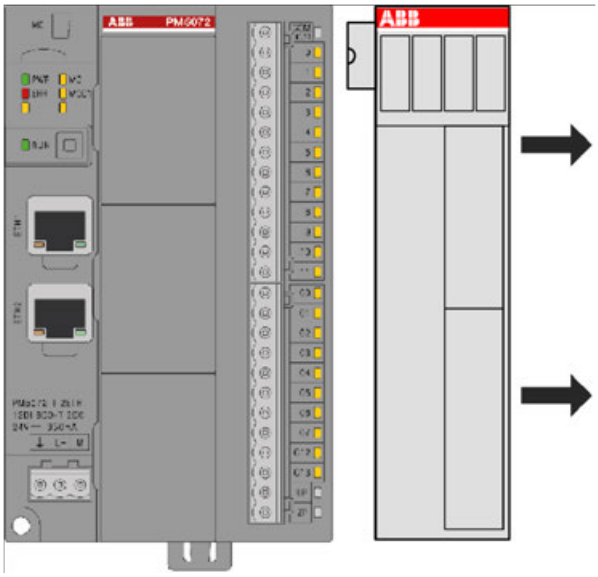


Demounting I/O modules mounted on a metal plate

1. Remove the 2 screws.



2. Remove the I/O module from the connected module by hand.



1.6.4.5.4 Connection and wiring

For detailed information such as technical data of your mounted devices (AC500 product family) refer to the hardware device specification of the appropriate device.

Power supply

The processor modules PM50x2 can be connected to the 24 V DC supply voltage via a removable 3-pin spring terminal block or a 3-pin screw terminal block.

Table 593: Removable terminal block for the supply voltage 24 V DC

3-pin spring terminal block	3-pin screw terminal block
	

The terminal block is available as a set for AC500-eCo V3 processor modules.

Basic CPU (PM5012)		**Standard CPUs (PM5032, PM5052) and Pro CPUs (PM5072)**	
Spring type	**Screw type**	**Spring type**	**Screw type**
TA5211-TSPF-B	TA5211-TSCL-B	TA5212-TSPF	TA5212-TSCL

Further information on the terminal blocks concerning power supply and onboard inputs/outputs are provided under pluggable connectors for screw and spring connection *Chapter 1.6.4.5.5.2 “TA52xx(-x) - Terminal block sets” on page 3379*.

Pin assignment

Pin Assignment	Pin	Label	Function	Description
 Terminal block inserted	1	⏚	FE	Functional earth
	2	L+	+24 V DC	Positive pin of the power supply voltage
	3	M	0 V	Negative pin of the power supply voltage

Faulty wiring on power supply terminals

CAUTION!

Risk of damaging the AC500-eCo V3 processor module and the connected modules!

Voltages > 30 V DC might damage the processor module and the connected modules.

Make sure that the supply voltage never exceeds 30 V DC.

Processor module interfaces

I/O bus

The I/O bus is not available for PM5012-T-ETH and PM5012-R-ETH. I/O channel extension using option board slot only.

The I/O bus is the I/O data bus for the I/O modules. Through this bus, I/O and diagnosis data are transferred between the processor module and the I/O modules. Up to 10 I/O modules for PM5032-x-ETH (but with a limit of 128 Bytes input/ 128 Bytes output variables) and 10 I/O modules for PM5052-x-ETH and PM5072-T-2ETH can be added.

Option board slot interface

Depending on the processor module variants, an additional option board can be connected to the option board slot to extend the feature of the processor module ↬ *Chapter 1.6.2.6.2.1.1 “Option boards for AC500-eCo V3 processor modules” on page 2410* .

Serial interface

RS-232 communication interface is available by using option board:

- TA5141-RS232I (isolated)
  ↬ *Chapter 1.6.3.3.1.2.6 “TA5141-RS232I - Option board for COMx serial communication” on page 2502*

RS-485 communication interface is available by using option boards:

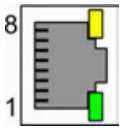
- TA5142-RS485I (isolated)
  ↬ *Chapter 1.6.3.3.1.2.7 “TA5142-RS485I - Option board for COMx serial communication” on page 2504*
- TA5142-RS485 (non isolated)
  ↬ *Chapter 1.6.3.3.1.2.8 “TA5142-RS485 - Option board for COMx serial communication” on page 2510*

Ethernet

Ethernet is also used for Modbus TCP connection.

Ethernet interface

The Ethernet interface is carried out via a RJ45 jack. The pin assignment of the Ethernet interface:

Interface	Pin	Description	
	1	Tx+	Transmit Data +
	2	Tx-	Transmit Data -
	3	Rx+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	Rx-	Receive data -
	7	NC	Not connected

Interface	Pin	Description	
	8	NC	Not connected
	Shield	Cable shield	Functional earth

Modbus RTU connection details

The Modbus RTU protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Available serial interfaces can work as Modbus interfaces simultaneously.

The Modbus client operating mode of an interface is set with the function block COM_MOD_MAST.

Technical data

Table 594: Description of the Modbus protocol

Parameter	Value
Supported standard	See *Serial interface* ↳ *Chapter 1.6.6.2.14.1 "Configuring Modbus RTU on serial interface" on page 3793*
Number of connection points	1 client Max. 1 server with RS-232 interface Max. 31 servers with RS-485
Protocol	Modbus
Operating mode	Client/server
Address	Server only
Data transmission control	CRC16
Data transmission speed	From 9,600 bits/s to 115,200 bits/s (see *Serial interface* ↳ *Chapter 1.6.6.2.14.1 "Configuring Modbus RTU on serial interface" on page 3793*)
Encoding	1 start bit 8 data bits 1 or 2 stop bits 1 parity bit (see *Serial interface* ↳ *Chapter 1.6.6.2.14.1 "Configuring Modbus RTU on serial interface" on page 3793*)
Max. cable length for RS-485 on serial interface option board used on the CPU.	1.200 m at 19.200 baud

Bus topology

Point-to-point with RS-232 or bus topology with RS-485. Modbus is a master-slave protocol.

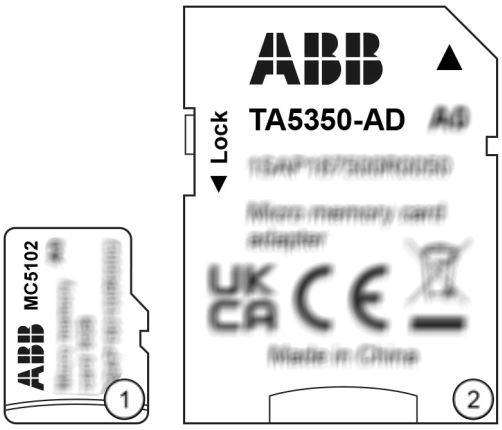
For further information on Modbus see chapter ↳ *Chapter 1.6.5.1.10 "Communication with Modbus RTU" on page 3542*.

1.6.4.5.5 Handling of accessories

This section only describes accessories that are frequently used for system assembly, connection and construction. A description of all additional accessories that can be used to supplement AC500 system can be found in the Hardware PLC device description.

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

> *The MC5102 micro memory card has no write protect switch.*
> *The TA5350-AD micro memory card adapter has a write protect switch.*
> *In the position "LOCK", the inserted micro memory card can only be read.*

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

> *The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.*

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

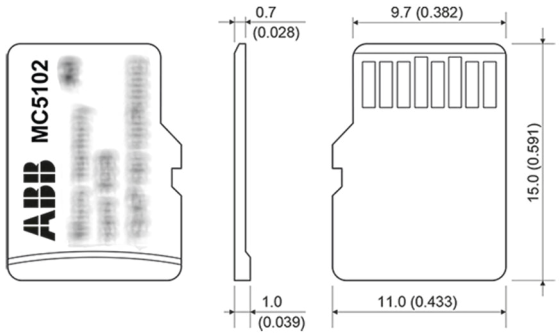
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

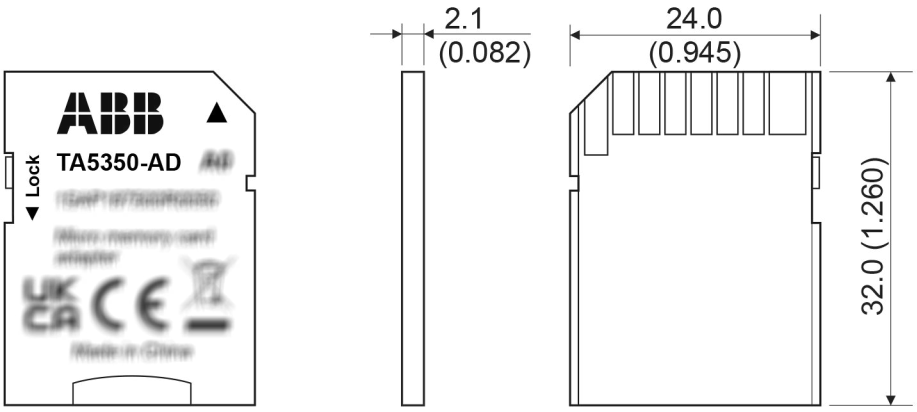
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V3

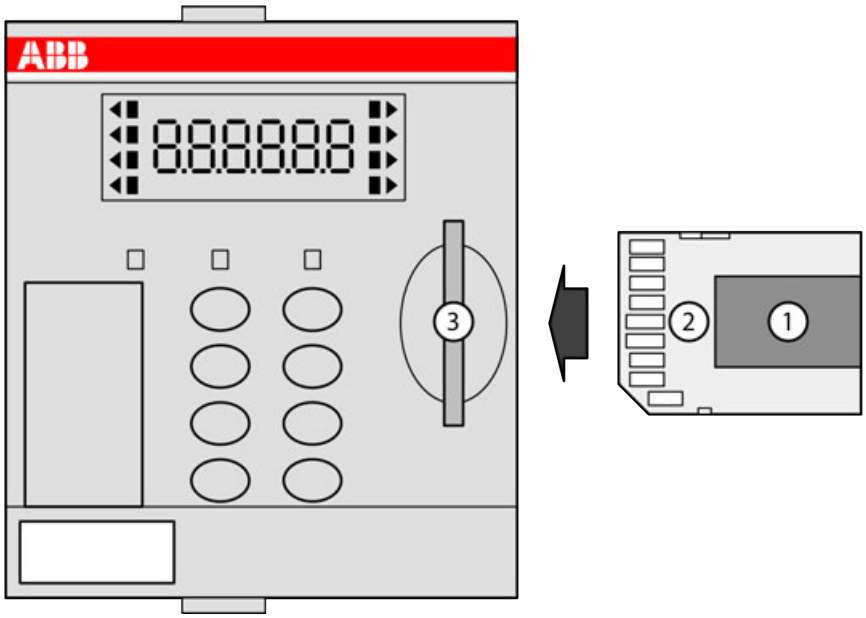
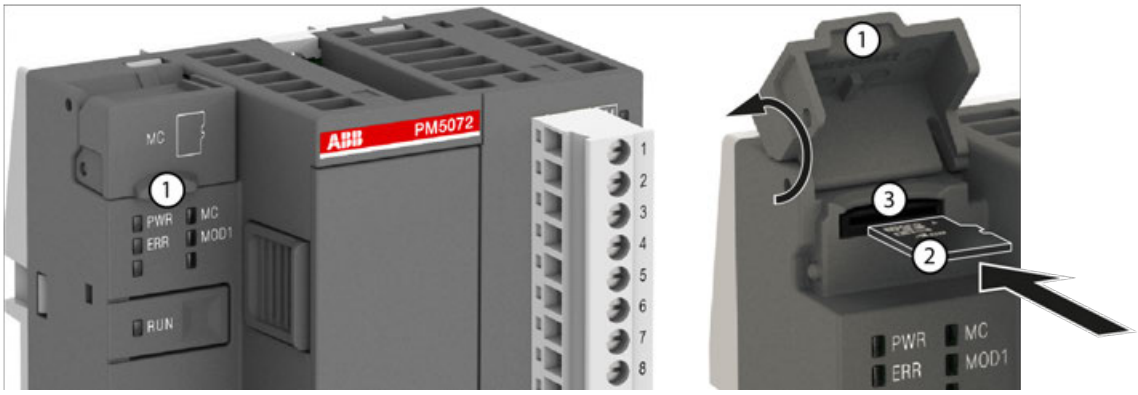


Fig. 277: Insert micro memory card into PM56xx

1 Micro memory card
2 TA5350-AD micro memory card adapter
3 Memory card slot

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Carefully insert the micro memory card into the micro memory card slot as far as it will go. Observe orientation of card.
3. Close the micro memory card slot cover by turning it downwards.

Remove the micro memory card

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

AC500 V3: Remove the micro memory card with micro memory card adapter only when no black square (■) is shown next to MC in the display.

AC500-eCo V3: Remove the micro memory card only when the MC LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

AC500 V3

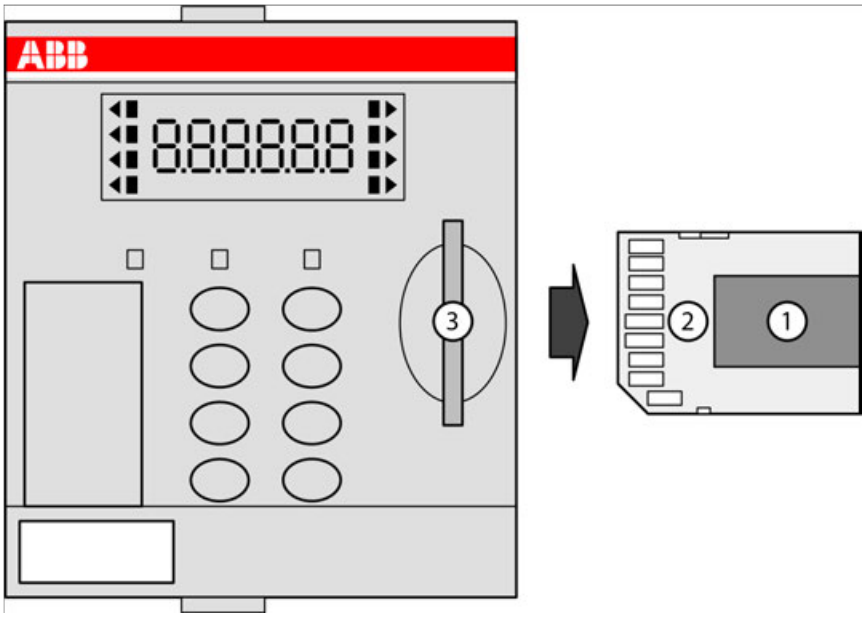
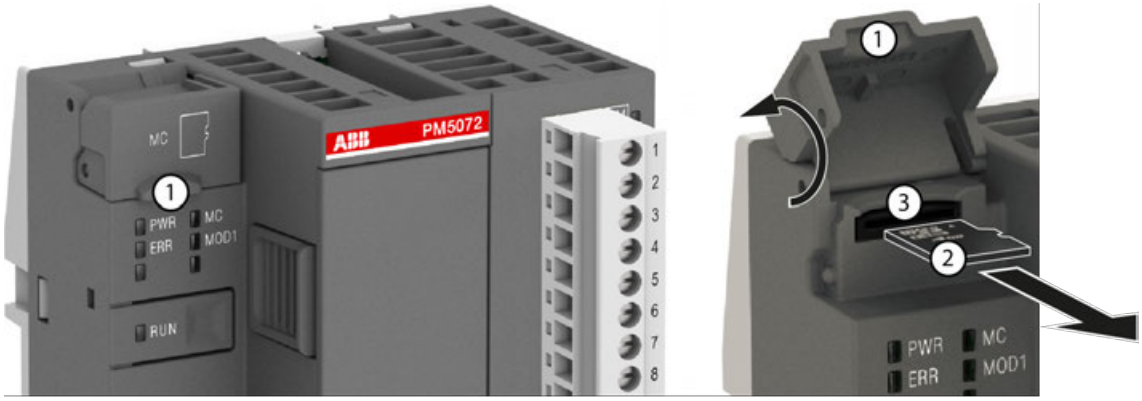


Fig. 278: Remove micro memory card from PM56xx

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Micro memory card can be removed from the micro memory card slot by gripping and pulling with two fingers.
3. Close the micro memory card slot cover by turning it downwards.

Technical data

Parameter		Value
Memory capacity		8 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter ↳ *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA52xx(-x) - Terminal block sets

Intended purpose

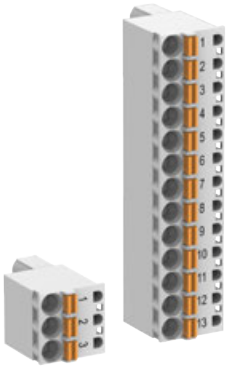
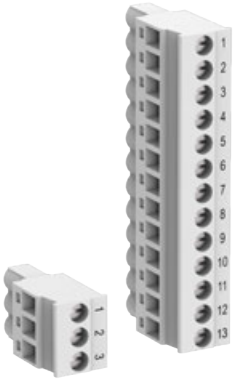
Removable terminal blocks are used for power supply and for I/O connectors on AC500-eCo V3 processor modules PM50x2.

For option boards there are different removable terminal blocks in spring version.

For the AC500-eCo V3 **Basic CPUs** a 3-pin terminal block for power supply and a 13-pin terminal block for I/O connectors are used.

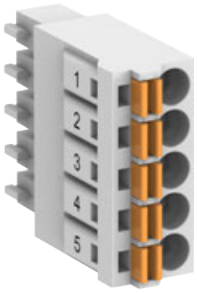
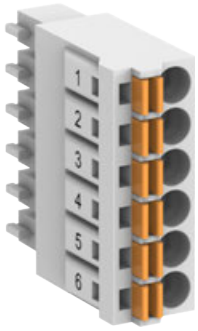
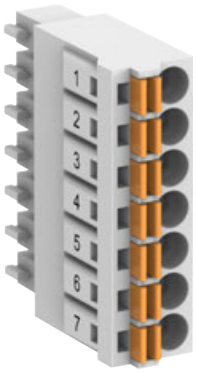
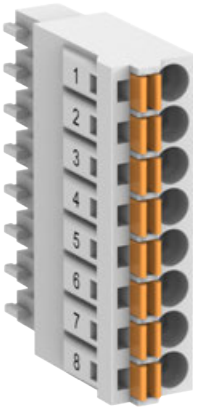
For the AC500-eCo V3 **Standard CPUs** and **Pro CPUs** a 3-pin terminal block for power supply, a 13-pin terminal block and a 12-pin terminal block for I/O connectors are used.

For all CPUs there is a screw and a spring variant available.

Basic CPU		Standard and Pro CPUs	
Spring type TA5211-TSPF-B	Screw type TA5211-TSCL-B	Spring type TA5212-TSPF	Screw type TA5212-TSCL
			

Various removable spring-type terminal blocks are available for option boards.

The following spare parts are available (depending on the number of pins).

Spring type			
TA5220-SPF5	**TA5220-SPF6**	**TA5220-SPF7**	**TA5220-SPF8**
			

CAUTION!

Risk of injury and damaging the product!

Improper installation and maintenance may result in injury and can damage the product!

- Installation and maintenance have to be performed according to the technical rules, codes and relevant standards, e.g. EN 60204-1.
- Read product documentation carefully before wiring. Improper wiring or wrong terminal block from other devices can damage the product!
- Only by qualified personnel.

CAUTION!

Risk of injury and damaging the processor module when using unapproved terminal blocks!

Only use terminal blocks approved by ABB to avoid injury and damage to the processor module.

Terminal block set for PM50x2

Processor modules PM50x2 CPU are not delivered with terminal blocks.

Screw type terminal block set:

- *TA5211-TSCL-B (1SAP187400R0001) for PM5012-x-ETH*
- *TA5212-TSCL (1SAP187400R0004) for PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH(W)*

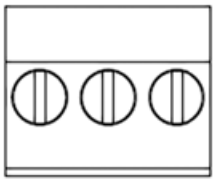
Spring type terminal block set:

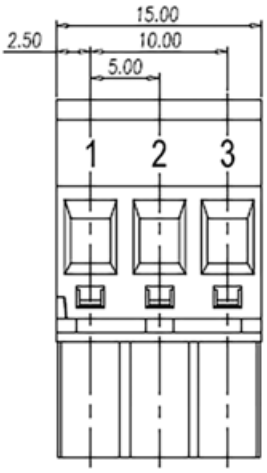
- *TA5211-TSPF-B (1SAP187400R0002) for PM5012-x-ETH*
- *TA5212-TSPF (1SAP187400R0005) for PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH(W)*

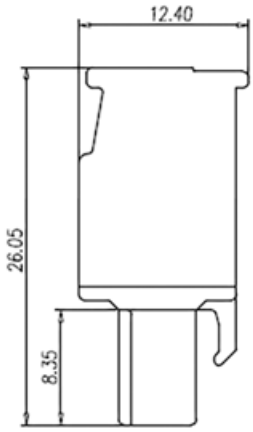
Dimensions

3-pin terminal block for power supply

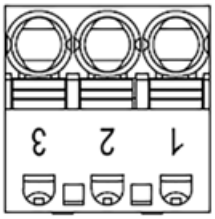
Screw type

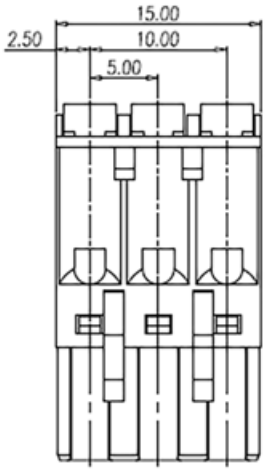


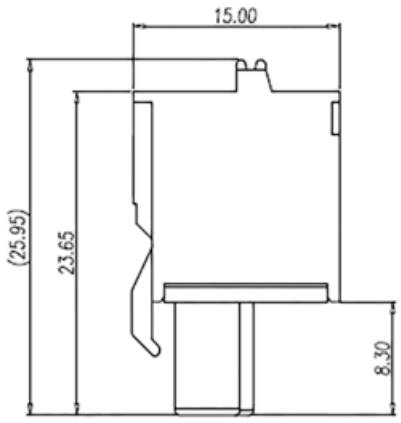




Spring type

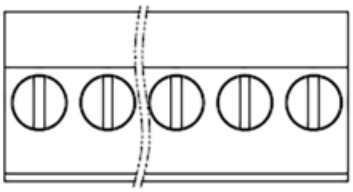


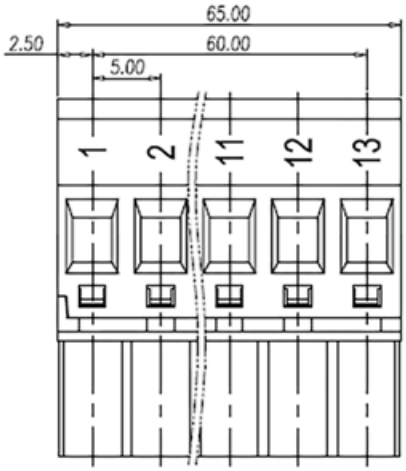


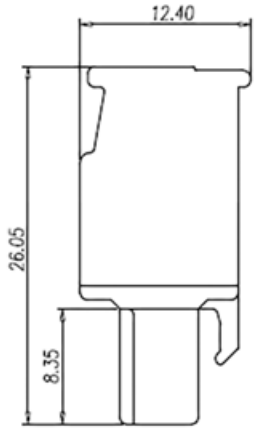


13-pin terminal block for I/O connectors

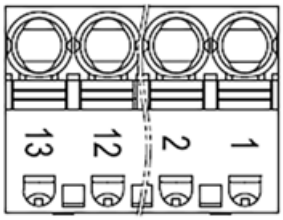
Screw type

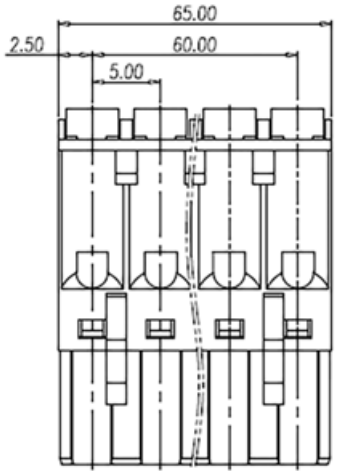


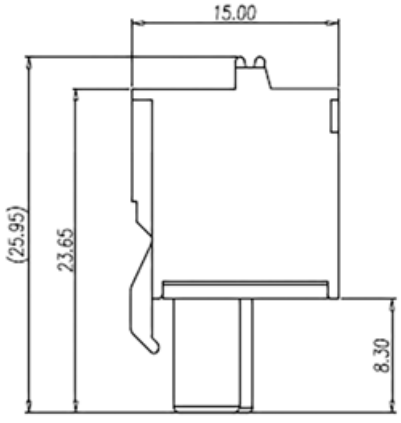




Spring type

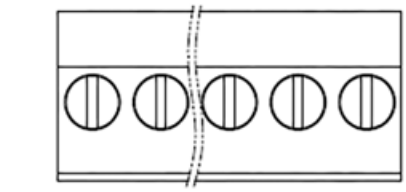


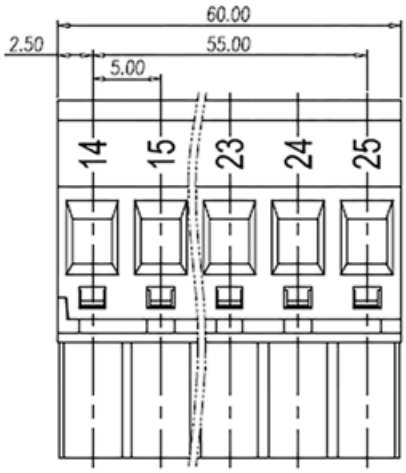


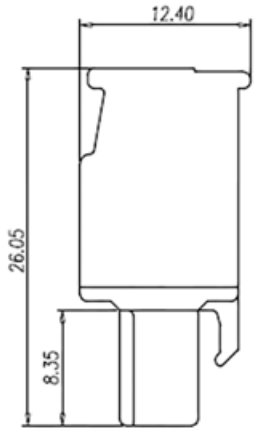


12-pin terminal block for I/O connectors

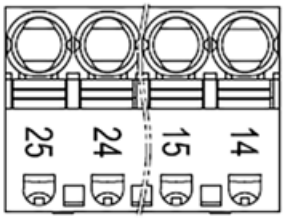
Screw type

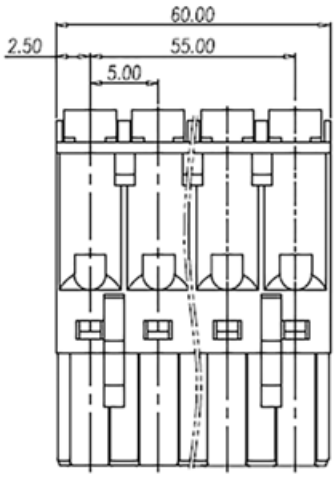


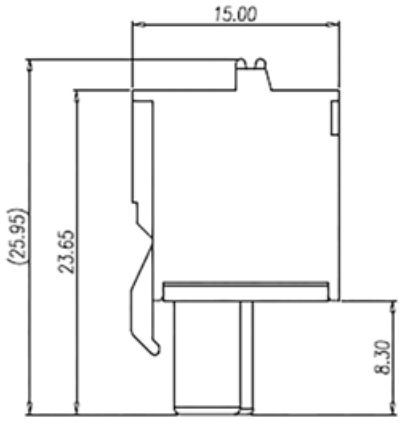




Spring type

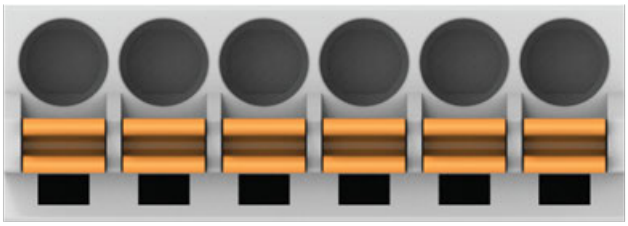


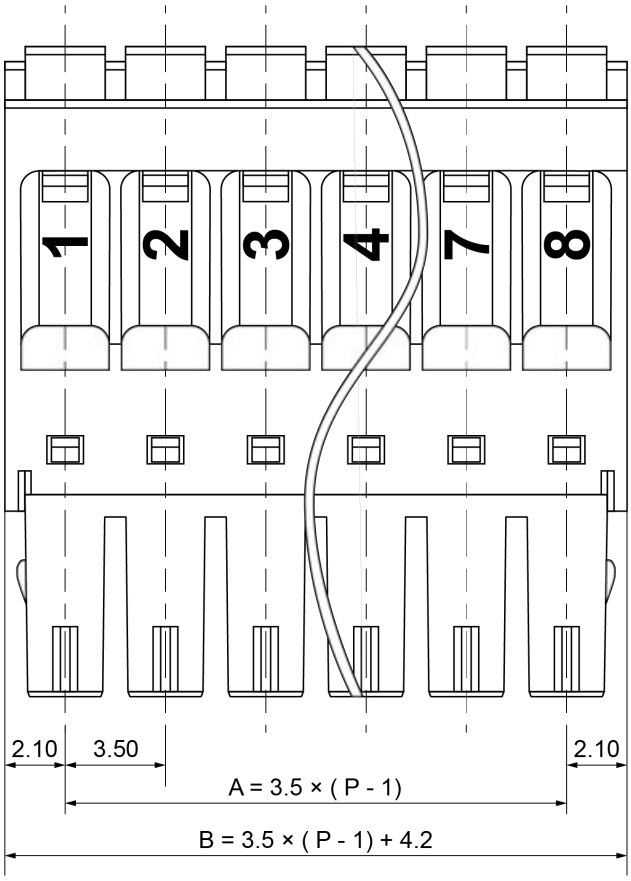


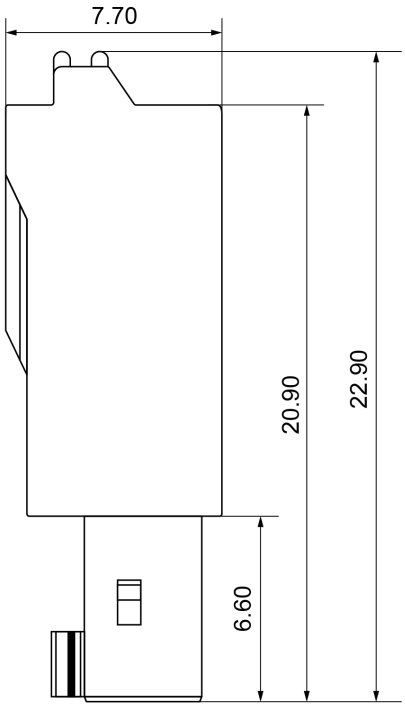


x-PIN terminal blocks for option boards

> *Only these x-pin blocks are available for the option boards.*
> *TA5220-SPF**x**, with **x** = 5...8*



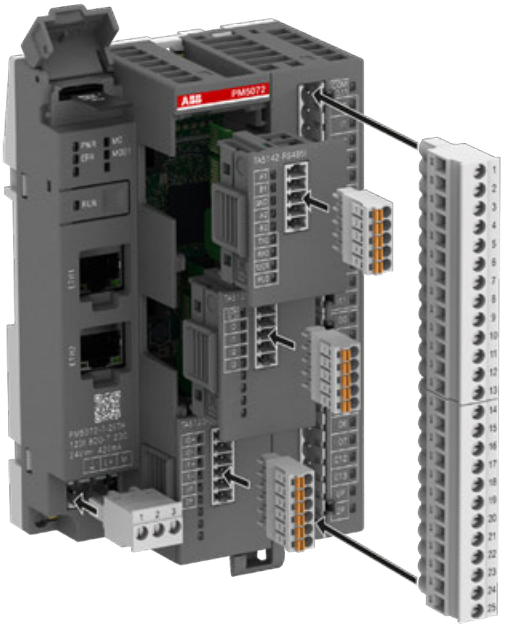




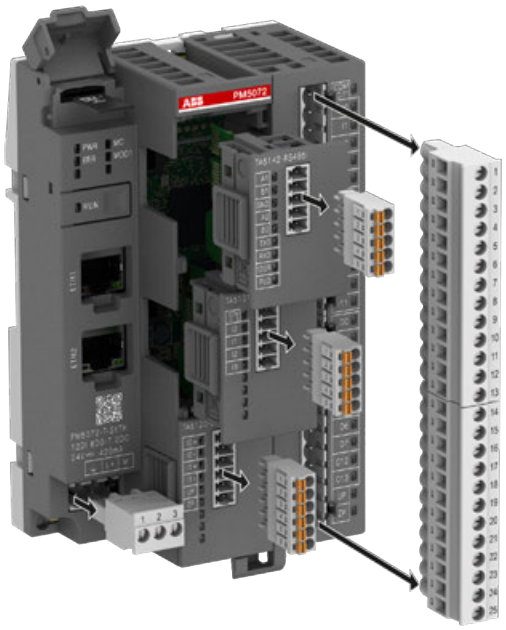
This results in these dimensions for the available spring terminal blocks.

Description	Pin	Length [mm]	Wide [mm]	Height [mm]
TA5220-SPF5	5	18.2	7.7	22.9
TA5220-SPF6	6	21.7	7.7	22.9
TA5220-SPF7	7	25.2	7.7	22.9
TA5220-SPF8	8	28.7	7.7	22.9

Assembly



Disassembly



Technical data

Table 595: Screw type terminal block for power supply

Parameter		Value
Type		
	TA5211-TSCL-B	Removable 3-pin terminal block:
	TA5212-TSCL	screw front/cable side 5.00 mm pitch
Usage		Power supply for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		7 mm

Parameter		Value
Fastening torque		0.5 Nm
Dimensions		
	3-pin terminal block	15 mm x 12.4 mm x 26.05 mm
Weight		
	TA5211-TSCL-B	150 g (2 terminal blocks)
	TA5212-TSCL	200 g (3 terminal blocks)

Table 596: Spring type terminal block for power supply

Parameter		Value
Type		
	TA5211-TSPF-B	Removable 3-pin terminal block:
	TA5212-TSPF	spring front/cable front 5.00 mm pitch
Usage		Power supply for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		11 mm
Dimensions		
	3-pin terminal block	15 mm x 15 mm x 25.95 mm
Weight		
	TA5211-TSPF-B	150 g (2 terminal blocks)
	TA5212-TSPF	200 g (3 terminal blocks)

Table 597: Screw type terminal block for onboard I/Os

Parameter		Value
Type		
	TA5211-TSCL-B	Removable 13-pin terminal block: screw front/cable side 5.00 mm pitch
	TA5212-TSCL	Removable 13-pin and 12-pin terminal block: screw front/cable side 5.00 mm pitch
Usage		Onboard I/Os for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		7 mm
Fastening torque		0.5 Nm
Dimensions		
	13-pin terminal block	65 mm x 12.4 mm x 26.05 mm
	12-pin terminal block	60 mm x 12.4 mm x 26.05 mm
Weight		

Parameter		Value
	TA5211-TSCL-B	150 g (2 terminal blocks)
	TA5212-TSCL	200 g (3 terminal blocks)

Table 598: Spring type terminal block for onboard I/Os

Parameter		Value
Type		
	TA5211-TSPF-B	Removable 13-pin terminal block: spring front/cable front 5.00 mm pitch
	TA5212-TSPF	Removable 13-pin and 12-pin terminal block: spring front/cable front 5.00 mm pitch
Usage		Onboard I/Os for AC500-eCo V3 processor modules
Conductor cross section		
	Solid (copper)	0.5 mm^2...2.5 mm^2
	Flexible (copper)	0.5 mm^2...2.5 mm^2
Stripped conductor end		11 mm
Dimensions		
	13-pin terminal block	65 mm x 15 mm x 25.95 mm
	12-pin terminal block	60 mm x 15 mm x 25.95 mm
Weight		
	TA5211-TSPF-B	150 g (2 terminal blocks)
	TA5212-TSPF	200 g (3 terminal blocks)

Table 599: Spring type terminal block for option boards

Parameter		Value
Type		
	TA5220-SPF5	Removable 5-pin terminal block: spring front, cable front 3.50 mm pitch
	TA5220-SPF6	Removable 6-pin terminal block: spring front, cable front 3.50 mm pitch
	TA5220-SPF7	Removable 7-pin terminal block: spring front, cable front 3.50 mm pitch
	TA5220-SPF8	Removable 8-pin terminal block: spring front, cable front 3.50 mm pitch
Usage		Connectors for AC500-eCo V3 option boards
Conductor cross section		
	Solid (copper)	0.2 mm^2...1.5 mm^2
	Flexible (copper)	0.2 mm^2...1.5 mm^2
Stripped conductor end		8 mm...10 mm
Dimensions		
	TA5220-SPF5	18.2 mm x 7.7 mm x 22.9 mm
	TA5220-SPF6	21.7 mm x 7.7 mm x 22.9 mm

Parameter		Value
	TA5220-SPF7	25.2 mm x 7.7 mm x 22.9 mm
	TA5220-SPF8	28.7 mm x 7.7 mm x 22.9 mm
Weight		
	TA5220-SPF5	150 g
	TA5220-SPF6	170 g
	TA5220-SPF7	180 g
	TA5220-SPF8	200 g

Ordering data

Part no.	Description
1SAP 187 400 R0001	TA5211-TSCL-B: screw terminal block set for AC500-eCo V3 CPU Basic screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors
1SAP 187 400 R0002	TA5211-TSPF-B: spring terminal block set for AC500-eCo V3 CPU Basic spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors

Part no.	Description
1SAP 187 400 R0004	TA5212-TSCL: screw terminal block set for AC500-eCo V3 Standard and Pro CPU screw front, cable side 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors
1SAP 187 400 R0005	TA5212-TSPF: spring terminal block set for AC500-eCo V3 Standard and Pro CPU spring front, cable front 5.00 mm pitch ● 1 removable 3-pin terminal block for power supply ● 1 removable 13-pin terminal block for I/O connectors ● 1 removable 12-pin terminal block for I/O connectors

Part no.	Description
Spare parts	
1SAP 187 400 R0012	TA5220-SPF5: spring terminal block, removable, 5-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0013	TA5220-SPF6: spring terminal block, removable, 6-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0014	TA5220-SPF7: spring terminal block, removable, 7-pin, spring front, cable front, 6 pieces per packing unit
1SAP 187 400 R0015	TA5220-SPF8: spring terminal block, removable, 8-pin, spring front, cable front, 6 pieces per packing unit

TA5300-CVR - Option board slot cover

Intended purpose

TA5300-CVR option board slot covers for PM50xx processor modules are necessary to protect not used option board slots.

CAUTION!

Risk of injury and damaging the product!

Always plug in the option board slot cover when the option board is not inserted.

If the option board slot cover is lost, please order the replacement TA5300-CVR (1SAP187500R0001).

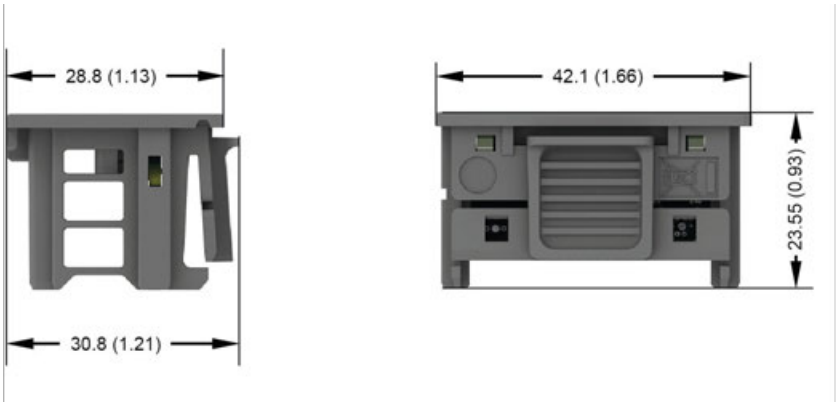
Never power up the CPU with uncovered option board slot, otherwise it may cause serious injury and/or damage the product.

The AC500-eCo V3 processor modules are delivered with option board slot cover(s).

The option board slot cover has to be removed before inserting an option board.

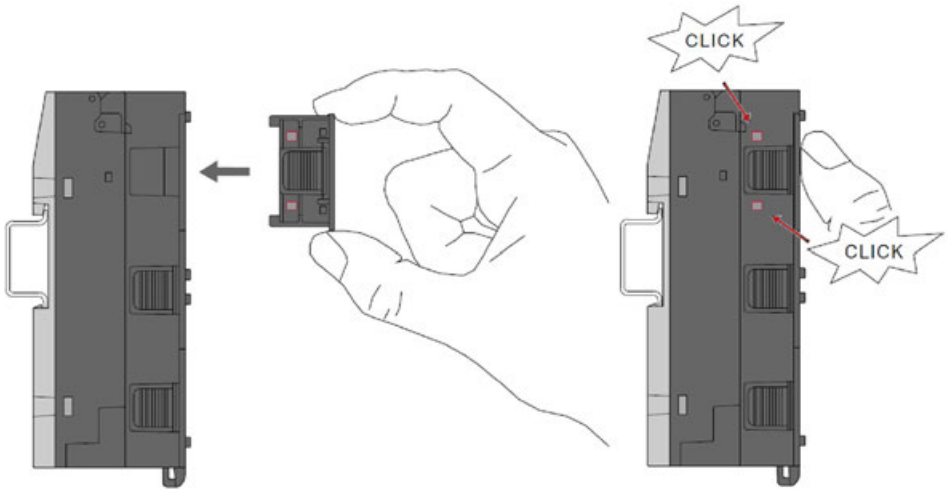
The TA5300-CVR option board slot covers are available as spare parts.

Dimensions



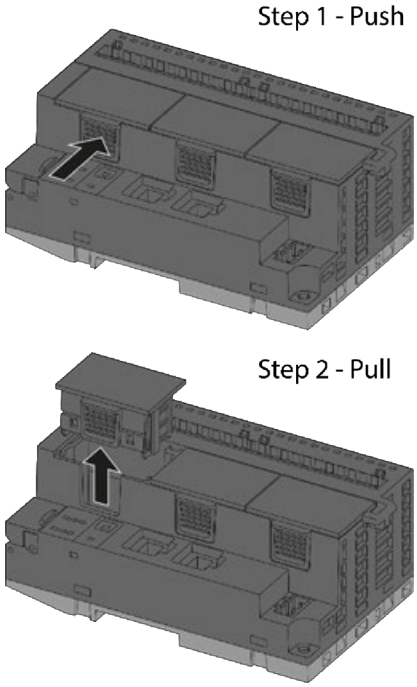
The dimensions are in mm and in brackets in inch.

Inserting of the option board slot cover



1. Press on the option board slot cover to insert it in the not used option board slot of the processor module PM50xx.
2. The option board slot cover must click into the not used option board slot.

Removing of the option board slot cover



1. Press the side of the inserted option board slot cover.
2. At the same time, pull the option board slot cover out of the option board slot of the processor module PM50xx.

Technical data

The system data of AC500-eCo V3 apply ↳ *Chapter 1.6.4.5.1 “System data AC500-eCo V3” on page 3352*

Only additional details are therefore documented below.

Parameter	Value
Weight	47 g
Dimensions	42.1 mm x 30.8 mm x 23.55

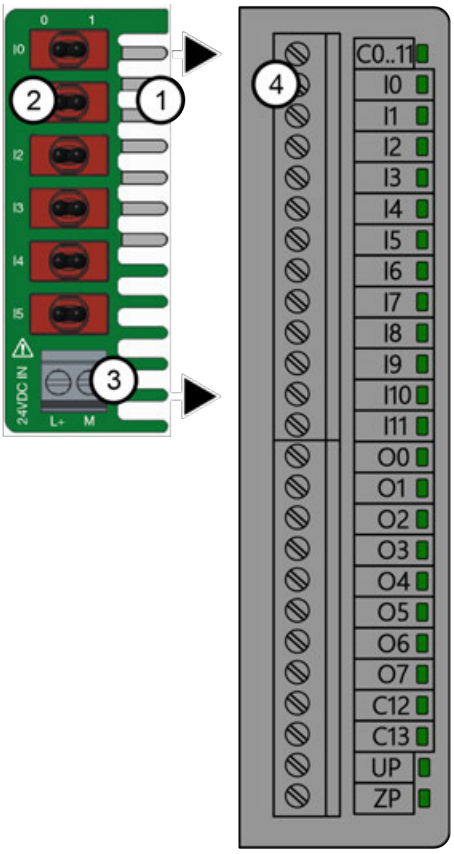
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 500 R0001	TA5300-CVR: option board slot cover, removable plastic part, 6 pieces per packing unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA5400-SIM - Input simulator

- TA5400-SIM input simulator for 6 digital inputs 24 V DC
- For usage with AC500-eCo V3 processor modules



1 Contacts for connecting the input simulator to the terminal block for I/O connectors
2 6 switches for the digital inputs DI0 ... DI5 (0 means opened switch, 1 means closed switch)
3 Screw terminal block for power supply
4 Screw terminal block(s) for I/O connectors

Intended purpose

TA5400-SIM

The TA5400-SIM input simulator is only intended for testing and training purposes for AC500-eCo V3 processor modules PM50x2.

Continuous operation in a productive system is not permitted.

The TA5400-SIM input simulator may only be used with screw-type terminal blocks.

The TA5400-SIM input simulator must not be used with spring-type terminal blocks.

Environmental conditions for testing and training purposes

In order not to impair the functionality of the product, avoid any kind of disturbing environmental influences:

- *mechanical disturbances*
- *climatic influences*

Make sure that the parameters are within the normal range:

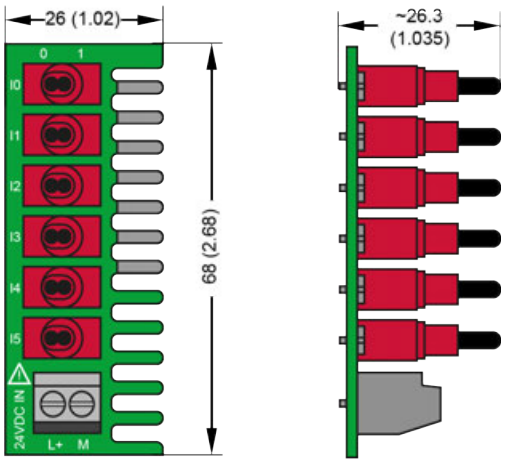
- *temperature*
- *air pressure*
- *humidity*
- *altitude*

The TA5400-SIM input simulator can simulate 6 digital 24 V DC input signals to the digital inputs I0...I5 of onboard I/Os.

With the TA5400-SIM input simulator, the digital 24 V DC inputs I0...I5 can be turned OFF and ON separately:

- If the lever of the switch is on the right side (1), the input is ON.
- If the lever of the switch is on the left side (0), the input is OFF.

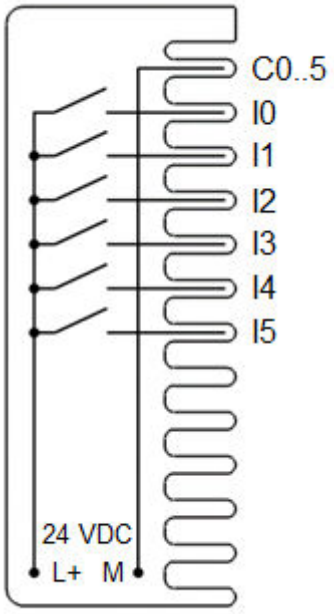
Dimensions



The dimensions are in mm and in brackets in inch.

Electrical diagram

The diagram below shows the connection of the TA5400-SIM input simulator.



NOTICE!

Risk of damage to the TA5400-SIM input simulator!

Do not remove the terminal block while the TA5400-SIM input simulator is connected.

Do not apply mechanical forces to the input simulator when it is connected to the terminal block.

In both cases the input simulator could be damaged.

Assembly

Insertion of the input simulator

1. Make sure that the power supply of the processor module is turned off.

CAUTION!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overvoltages and short circuits.

Make sure, that all voltage sources (supply and process voltage) are switched off before you start working on the system.

Never connect voltages > 24 V DC to the terminal block of the TA5400-SIM input simulator.

CAUTION!

Risk of damaging the input simulator and/or PLC modules!

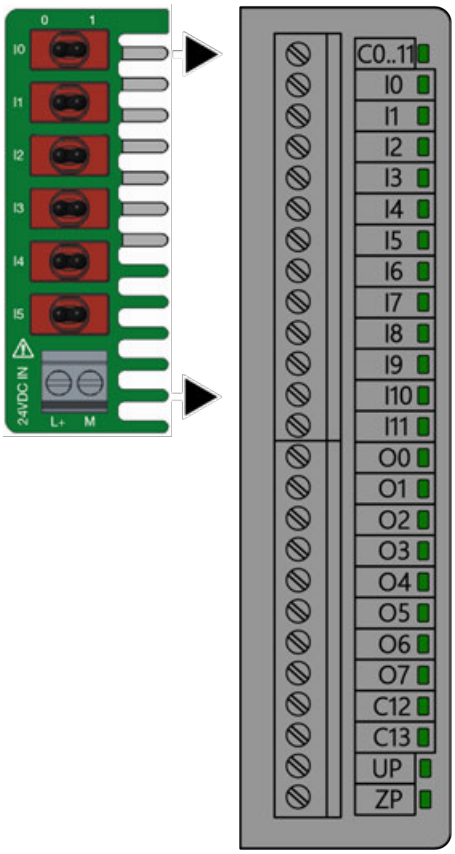
The TA5400-SIM input simulator may only be used with AC500-eCo V3 processor modules PM50x2.

Never use the input simulator with other devices.

The input simulator may only be used with screw-type terminal blocks.

The input simulator is only intended for testing and training purposes. Never use it within productive sytems.

2. Make sure that all clamps of the onboard I/Os are totally open.
3. Insert the TA5400-SIM input simulator into the screw terminal block as shown in the figure.



4. Tighten all screws of the onboard I/O clamps.
5. Make sure all switches are in OFF state (0).
6. Connect 24 V DC to the power supply of the TA5400-SIM (L+ and M). Tighten the screws.
7. Connect the processor module power supply wires (24 V DC). See PM50xx ↳ *"Pin assignment" on page 3371*.

Disassembly

Removal of the input simulator

1. Make sure that the power supply of the processor module is turned off.

> **CAUTION!**
> **Risk of damaging the PLC modules!**
> The PLC modules can be damaged by overvoltages and short circuits.
> Make sure that all voltage sources (supply and process voltage) are switched off before you start working on the system.

2. Disconnect the TA5400-SIM power supply wires (24 V DC) with a flat-blade screwdriver from the terminal block for power supply (L+ and M).
3. Loosen all screws of the onboard I/Os.
4. Remove the input simulator by pulling it to the left side.

Technical data

The system data of AC500-eCo V3 apply ↳ *Chapter 1.6.4.5.1 "System data AC500-eCo V3" on page 3352*

Only additional details are therefore documented below.

Table 600: Technical data of the module

Parameter		Value
Process supply voltage		
	Connections	Terminal (L+) for +24 V DC and terminal (M) for 0 V DC
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
Galvanic isolation		Yes (on processor module PM50xx)
Isolated Groups		1 (6 channels per group)
Weight		18 g
Mounting position		Horizontal or vertical

Table 601: Technical data of the inputs

Parameter	Value
Number of channels per module	6 digital input channels (+24 V DC)
Distribution of the channels into groups	1 (6 channels per group)
Connections of channels I0 to I5	Terminals 2...7
Reference potential for the channels I0 to I5	Terminal 1 (negative pole of the process supply voltage, signal name C0...5)
Input current per active channel (at input voltage +24 V DC) The current is given through the used processor module.	Typ. 5 mA
Inrush current per active channel The current is given through the used processor module.	Typ. 5 mA

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 187 600 R0001	TA5400-SIM, input simulator for PM50x2	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

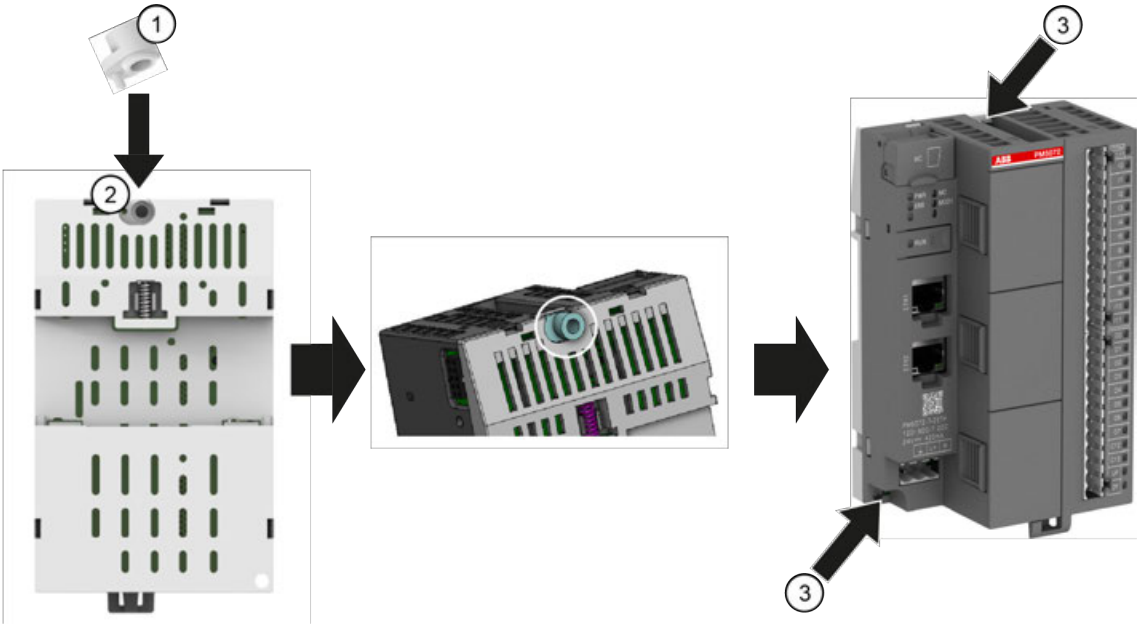
TA543 - Screw mounting accessory

Intended purpose

The TA543 screw mounting accessory is used for mounting the processor module PM50xx without DIN rail.

Handling instruction

TA543 must be snapped on the backside of PM50xx *Chapter 1.6.4.5.3.1.3 “Mounting a processor module on a metal plate” on page 3362*.



1 Screw mounting accessory TA543
2 Slot for screw mounting accessory TA543
3 2 holes for screw mounting

Technical data

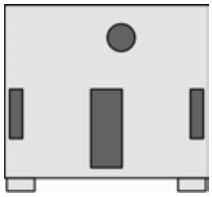
Parameter	Value
Weight	5 g
Dimensions	12 mm x 8.5 mm x 10 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 800 R0001	TA543, screw mounting accessory for PM50x2	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA566 - Wall mounting accessory



Intended purpose

The TA566 wall mounting accessory is used for mounting S500-eCo I/O modules without DIN rail.

Handling instruction

The TA566 is snapped into the back side of the device's housing ↳ *"Mounting I/O modules on a metal plate" on page 3367*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	29 mm x 28 mm x 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3107	TA566, wall mounting accessory, 100 pieces	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.4.6 AC500 (Standard)

1.6.4.6.1 System data AC500

Environmental conditions

Table 602: Process and supply voltages

Parameter		Value
24 V DC		
	Voltage	24 V (-15 %, +20 %)
	Protection against reverse polarity	Yes
120 V AC		
	Voltage	120 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
230 V AC		
	Voltage	230 V AC (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
120 V AC...240 V AC wide-range supply		
	Voltage	120 V...240 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
Allowed interruptions of power supply, according to EN 61131-2		
	DC supply	Interruption < 10 ms, time between 2 interruptions > 1 s, PS2
	AC supply	Interruption < 0.5 periods, time between 2 interruptions > 1 s

NOTICE!

Exceeding the maximum power supply voltage for process or supply voltages could lead to unrecoverable damage of the system. The system might be destroyed.

NOTICE!
Improper voltage level or frequency range which cause damage of AC inputs:
– AC voltage above 264 V
– Frenquency below 47 Hz or above 62.4 Hz

NOTICE!
Improper connection leads cause overtemperature on terminals.
PLC modules may be destroyed by using wrong cable type, wire size and cable temperature classification.

Parameter		Value
Temperature		
	Operating	0 °C...+60 °C: Horizontal mounting of modules. 0 °C...+40 °C: Vertical mounting of modules. Output load reduced to 50 % per group.
	Storage	-40 °C...+70 °C
	Transport	-40 °C...+70 °C
Humidity		Max. 95 %, without condensation
Air pressure		
	Operating	> 800 hPa / < 2000 m
	Storage	> 660 hPa / < 3500 m
Ingress protection		IP20

Creepage distances and clearances

The creepage distances and clearances meet the requirements of the overvoltage category II, pollution degree 2.

Insulation test voltages, routine test

According to EN 61131-2

Parameter	Value	
230 V circuits against other circuitry	2500 V	1.2/50 µs
120 V circuits against other circuitry	1500 V	1.2/50 µs
120 V...240 V circuits against other circuitry	2500 V	1.2/50 µs
24 V circuits (supply, 24 V inputs/outputs, analog inputs/outputs), if they are galvanically isolated against other circuitry	500 V	1.2/50 µs

Parameter	Value	
COM interfaces, galvanically isolated	500 V	1.2/50 µs
Ethernet	500 V	1.2/50 µs
230 V circuits against other circuitry	1350 V	AC 2 s
120 V circuits against other circuitry	820 V	AC 2 s
120 V...240 V circuits against other circuitry	1350 V	AC 2 s
24 V circuits (supply, 24 V inputs/outputs, analog inputs/outputs), if they are galvanically isolated against other circuitry	350 V	AC 2 s
COM interfaces, galvanically isolated	350 V	AC 2 s
	Not applicable	Not applicable
Ethernet	350 V	AC 2 s

According to IEC 61010-2-201

The content of the following table is only valid for PM56xx and TB56xx.

Table 603: Insulation, test voltages and continuous voltages

	Insulation	Test Voltage	Continuous Voltage
COM interfaces, galvanically isolated	1.1 mm	1216 V DC (60 s) 1500 V (1.2/50µs)	75 V
CAN interface, galvanically isolated	1.1 mm	1216 V DC (60 s) 1500 V (1.2/50µs)	75 V
Ethernet	1.1 mm	1500 V rms (50-60 Hz, 60 s) 2400 V (1.2/50µs)	On request

Power supply units

For the supply of the modules, power supply units according to SELV or PELV specifications must be used.

Safety Extra Low Voltage (SELV) and Protective Extra Low Voltage (PELV)

To ensure electrical safety of AC500/AC500-eCo extra low voltage circuits, 24 V DC supply, communication interfaces, I/O circuits, and all connected devices must be powered from sources meeting requirements of SELV, PELV, class 2, limited voltage or limited power according to applicable standards.

WARNING!

Improper installation can lead to death by touching hazardous voltages!

To avoid personal injury, safe separation, double or reinforced insulation and separation of the primary and secondary circuit must be observed and implemented during installation.

- Only use power converters for safety extra-low voltages (SELV) with safe galvanic separation of the primary and secondary circuit.
- Safe separation means that the primary circuit of mains transformers must be separated from the secondary circuit by double or reinforced insulation. The protective extra-low voltage (PELV) offers protection against electric shock.

Electromagnetic compatibility

Table 604: Range of use

Parameter	Value
Industrial applications	Yes
Domestic applications	No

Table 605: Immunity against electrostatic discharge (ESD), according to IEC 61000-4-2, zone B, criterion B

Parameter	Value
Electrostatic voltage in case of air discharge	8 kV
Electrostatic voltage in case of contact discharge	4 kV, in a closed switchgear cabinet 6 kV [1])
ESD with communication connectors	In order to prevent operating malfunctions, it is recommended, that the operating personnel discharge themselves prior to touching communication connectors or perform other suitable measures to reduce effects of electrostatic discharges.
ESD with connectors of terminal bases	The connectors between the Terminal Bases and processor modules or Communication Modules must not be touched during operation. The same is valid for the I/O bus with all modules involved.

[1]) High requirement for shipping classes are achieved with additional specific measures (see specific documentation).

Table 606: Immunity against the influence of radiated (CW radiated), according to IEC 61000-4-3, zone B, criterion A

Parameter	Value
Test field strength	10 V/m

Table 607: Immunity against fast transient interference voltages (burst), according to IEC 61000-4-4, zone B, criterion B

Parameter	Value
Supply voltage units (DC)	2 kV
Supply voltage units (AC)	2 kV
Digital inputs/outputs (24 V DC)	1 kV
Digital inputs/outputs (120 V AC...240 V AC)	2 kV
Analog inputs/outputs	1 kV
CS31 bus	1 kV
Serial RS-485 interfaces (COM)	1 kV
Serial RS-232 interfaces (COM, not for PM55x and PM56x)	1 kV
Ethernet	1 kV
I/O supply (DC-out)	1 kV

Table 608: Immunity against the influence of line-conducted interferences (CW conducted), according to IEC 61000-4-6, zone B, criterion A

Parameter		Value
Test voltage		3V zone B, 10 V is also met.
High energy surges		According to IEC 61000-4-5, zone B, criterion B
	Power supply DC	1 kV CM / 0.5 kV DM [2])
	DC I/O supply	0.5 kV CM / 0.5 kV DM [2])
	Communication Lines, shielded	1 kV CM [2])
	AC I/O unshielded [3])	2 kV CM / 1 kV DM [2])
	I/O analog, I/O DC unshielded [3])	1 kV CM / 0.5 kV DM [2])
Radiation (radio disturbance)		According to IEC 55011, group 1, class A

[2]) CM = Common Mode, DM = Differential Mode

[3]) When DC I/O inputs are used with AC voltage, external filters limiting high energy surges to 1 kV CM / 0.5 DM are required to meet requirements according IEC 61131-2.

Mechanical data

Parameter	Value
Mounting	Horizontal
Degree of protection	IP 20
Housing	Classification V-2 according to UL 94

Parameter	Value
Vibration resistance acc. to EN 61131-2	all three axes 2 Hz...8.4 Hz, continuous 3.5 mm 8.4 Hz...150 Hz, continuous 1 g (higher values on request)
Shock test	All three axes 15 g, 11 ms, half-sinusoidal
Mounting of the modules:	
DIN rail according to DIN EN 50022	35 mm, depth 7.5 mm or 15 mm
Mounting with screws	Screws with a diameter of 4 mm
Fastening torque	1.2 Nm

Approvals and certifications

Information on approvals and certificates can be found in the corresponding chapter of the *Main catalog, PLC Automation*.

1.6.4.6.2 Mechanical dimensions

Switchgear cabinet assembly

Information on EMC-conforming assembly and construction is provided within the overall functions section ↳ Chapter 1.6.4.4.4 “EMC-conforming assembly and construction” on page 3345.

PLC enclosure

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

To protect PLCs against:

- unauthorized access,
- dusting and pollution,
- moisture and wetness and
- mechanical damage,

switchgear cabinet IP54 for common dry factory floor environment is suitable.

Maintain spacing from:

- enclosure walls
- wireways
- adjacent equipment

Allow a minimum of 20 mm clearance on all sides. This provides ventilation and galvanic isolation.

It is recommended to mount the modules on an grounded mounting plate, or an grounded DIN rail, independent of the mounting location.

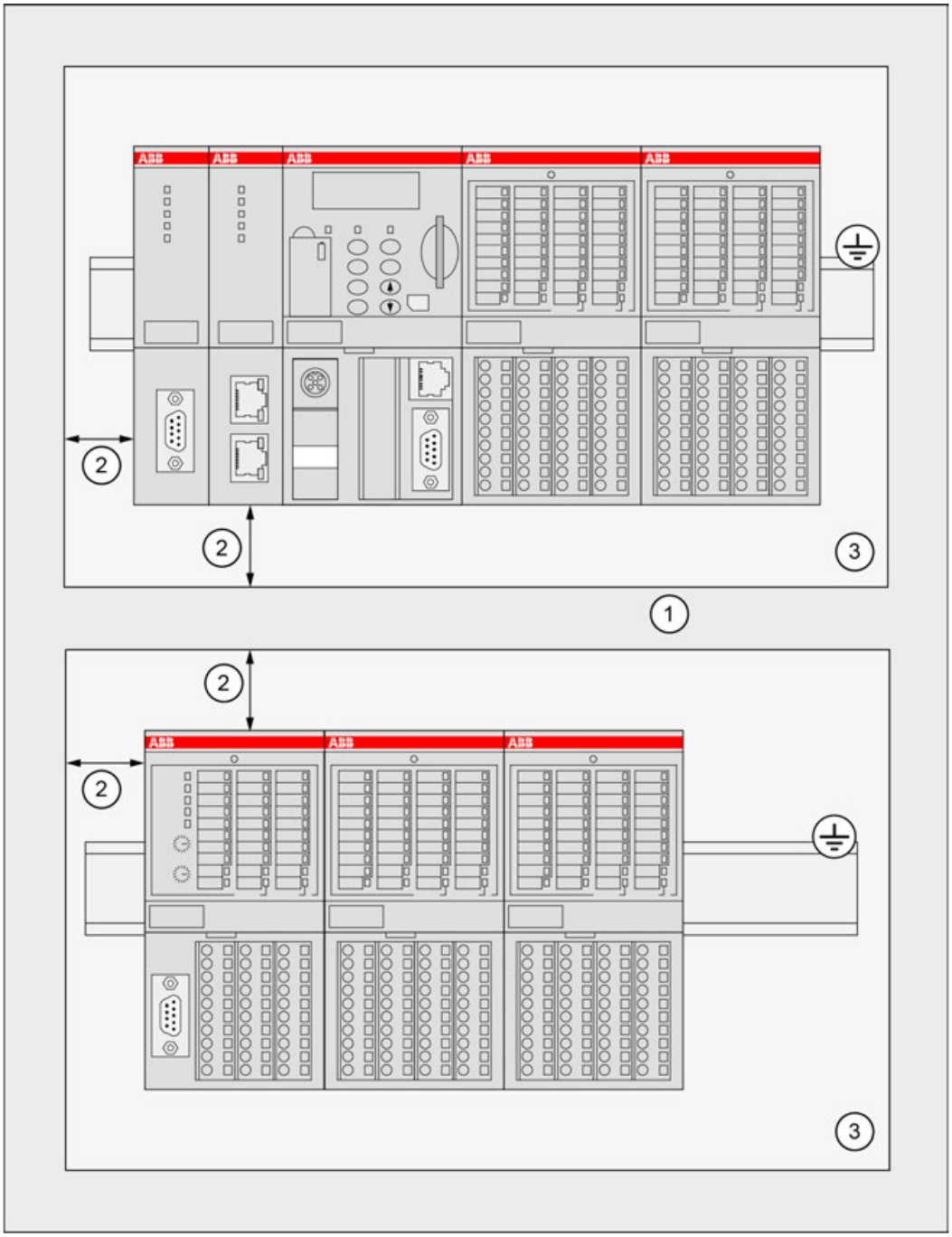


Fig. 279: Installation of AC500/S500 modules in a switchgear cabinet

1 Cable duct
2 Distance from cable duct ≥20 mm
3 Mounting plate, grounded

NOTICE!

Horizontal mounting is highly recommended.

Vertical mounting is possible, however, derating consideration should be made to avoid problems with poor air circulation and overheating (see *Chapter 1.6.4.6.1.1 “Environmental conditions” on page 3398*).

When vertically mounted, always place an end-stop terminal block (e.g. type BADL, P/N: 1SNA399903R0200) on the bottom and on the top of the modules to properly secure the modules.

With high vibration applications and horizontal mounting, we also recommend to place end-stop terminals at the right and left side of the device to properly secure the modules, e.g. type BADL, P/N: 1SNA399903R0200.

Mechanical dimensions AC500

Dimensions: terminal bases

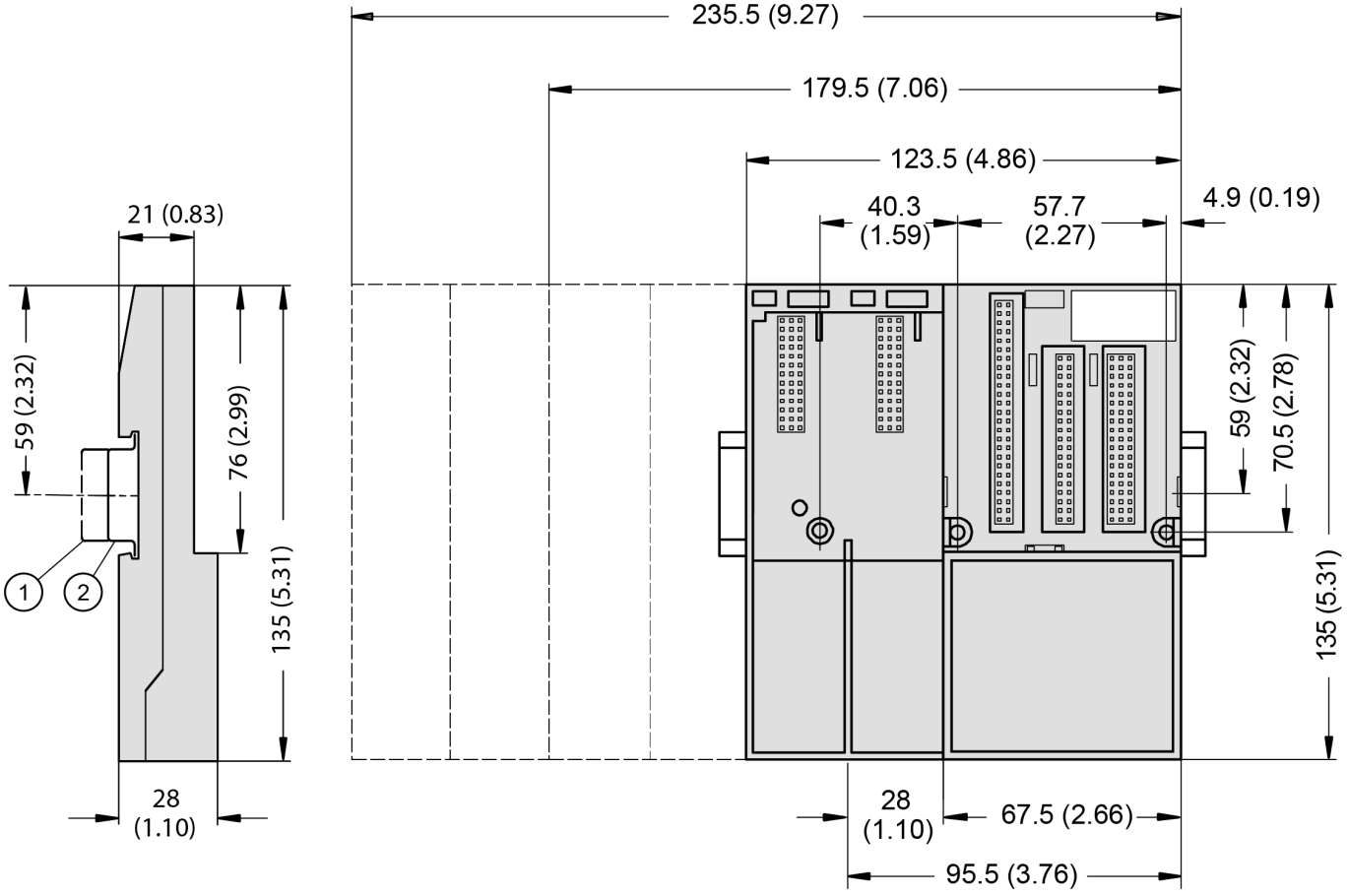


Fig. 280: Terminal bases, side view and front view

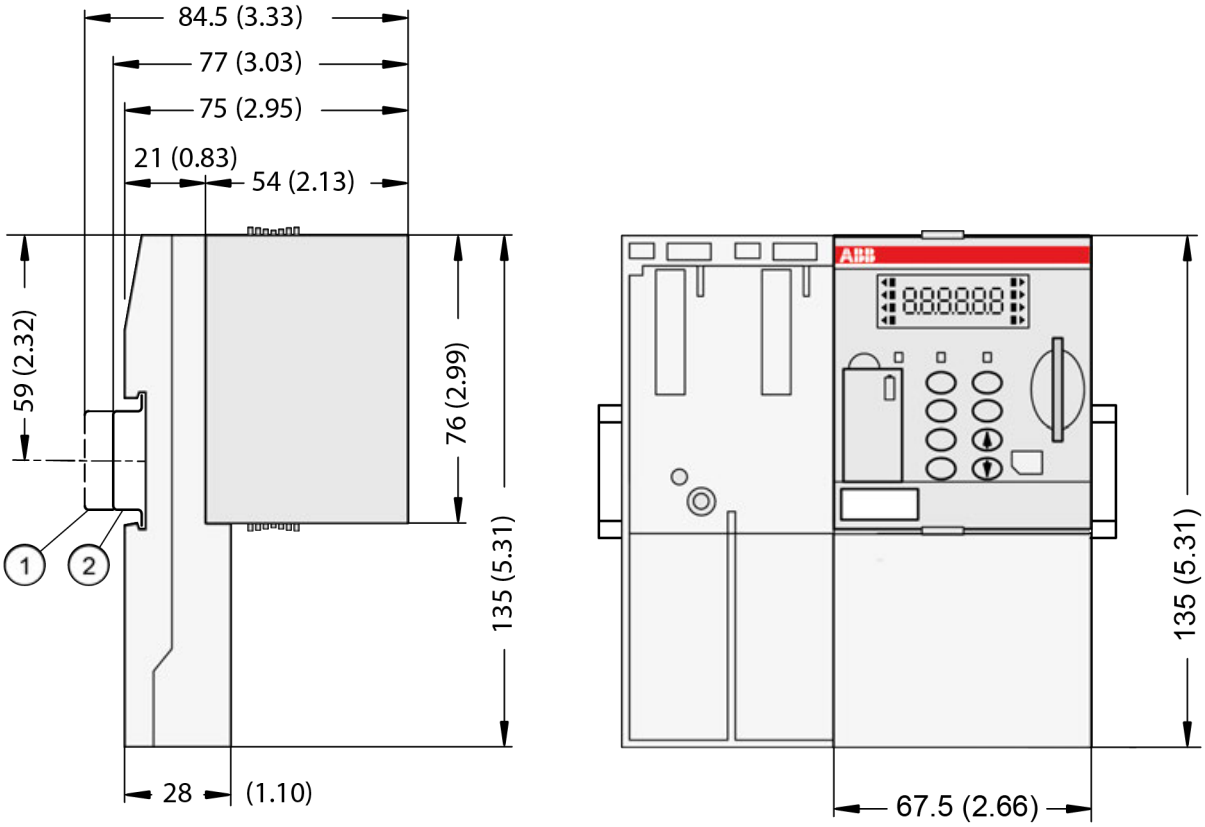


Fig. 281: Terminal bases with processor modules, side view and front view

Mechanical dimensions S500

Dimensions:
Terminal units

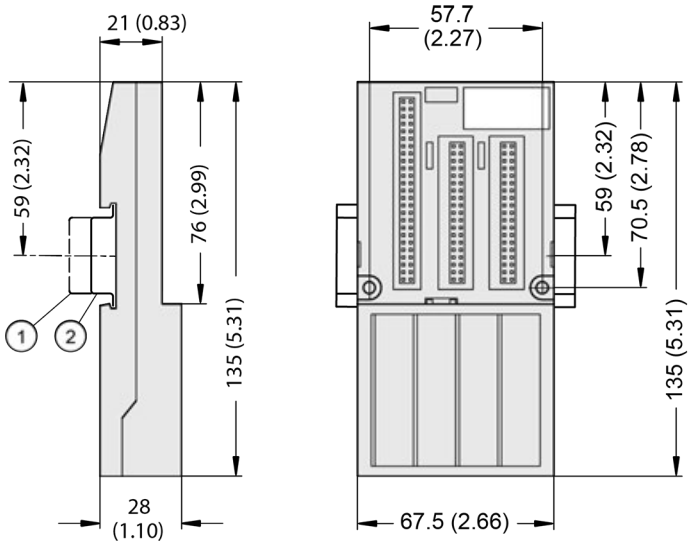


Fig. 282: Terminal units, side view and front view

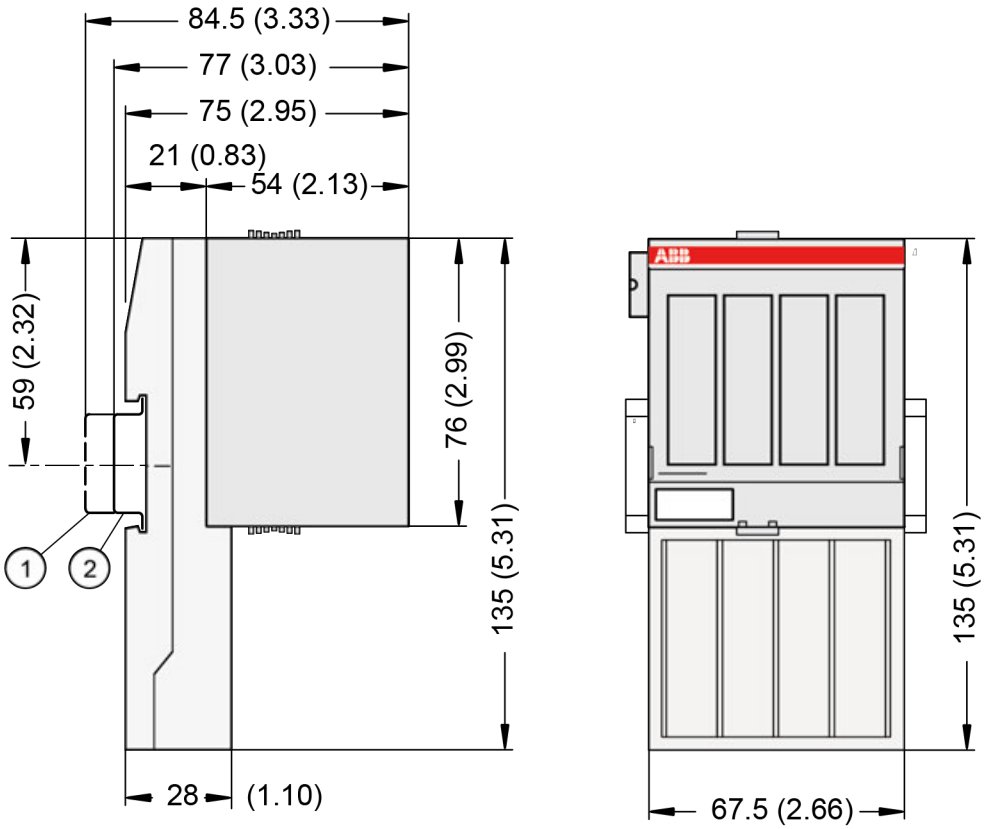


Fig. 283: Terminal units and S500 modules, side view and front view

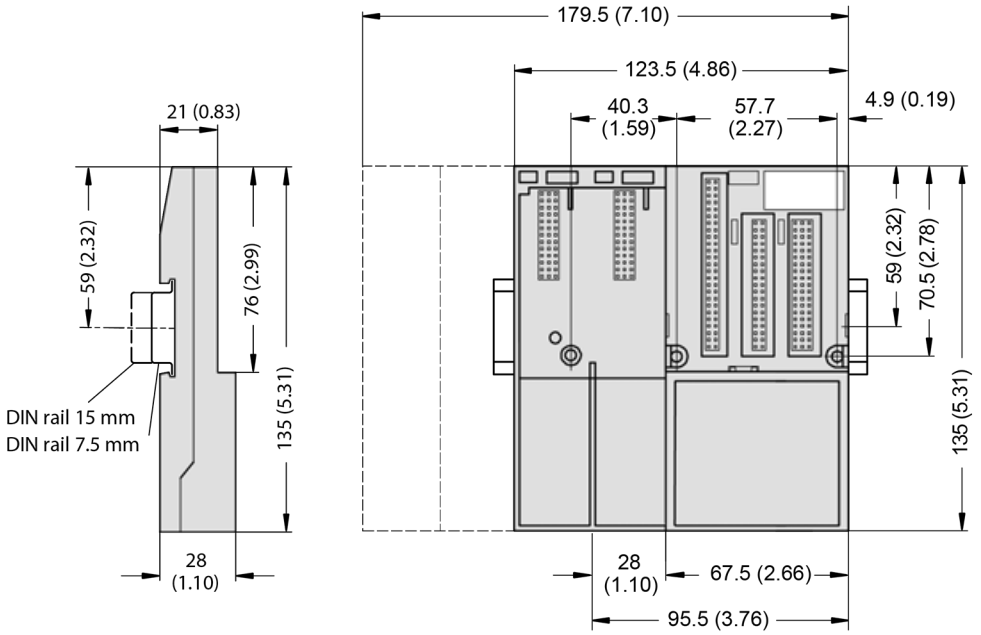


Fig. 284: Terminal base (for comparison)

All dimensions are in mm (in.). Hole spacing tolerance: ±0.4 mm (0.016 in.)

1.6.4.6.3 Mounting and demounting

The control system is designed to be mounted to a well-grounded mounting surface such as a metal panel. Additional grounding connections from the mounting tabs or DIN rail (if used), are not required unless the mounting surface cannot be grounded.

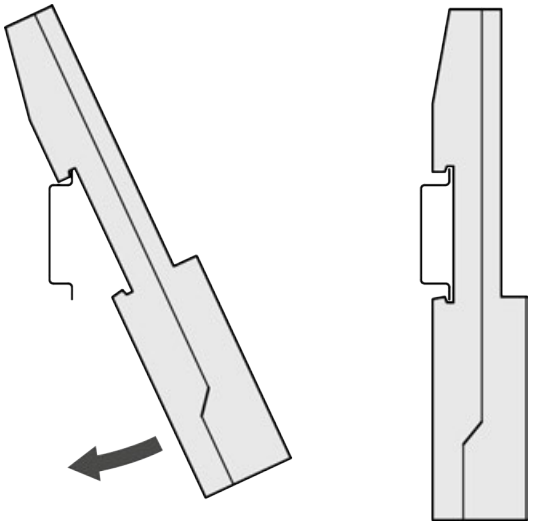
During panel or DIN rail mounting of all devices, be sure that all debris (metal chips, wire strands, etc.) is kept from falling into the controller. Debris that falls into the controller could cause damage while the controller is energized.

All devices are grounded through the DIN rail to chassis ground. Use zinc plated yellow-chromate stell DIN rail to assure proper grounding. The use of other DIN rail materials (e.g. aluminium, plastic, etc.) that can corrode, oxidize, or are poor conductors, can result in improper or intermittent grounding.

Mounting/Demounting terminal bases and function module terminal bases

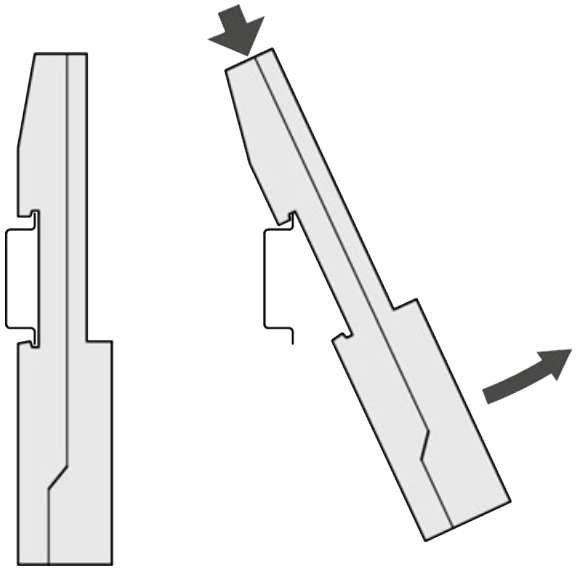
Demounting on DIN rail

1. Mount DIN rail 7.5 mm or 15 mm.
2. Mount the terminal base/function module terminal base:



⇨ The terminal base is put on the DIN rail above and then snapped-in below.

3. The demounting is carried out in a reversed order.



Mounting with screws

If the terminal base should be mounted with screws, wall mounting accessories TA526 ↳ *Chapter 1.6.4.6.5.5 “TA526 - Wall mounting accessory” on page 3445* must be inserted at the rear side first. These plastic parts prevent bending of the terminal base while screwing on. TB560x and TB561x need one TA526, TB562x, TB564x and TB566x need two TA526.

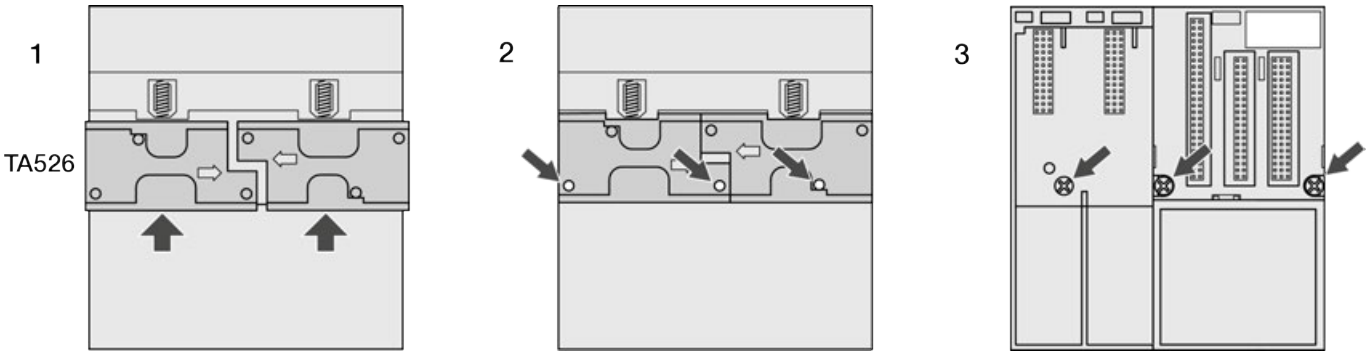


Fig. 285: Terminal bases, Fastening with screws

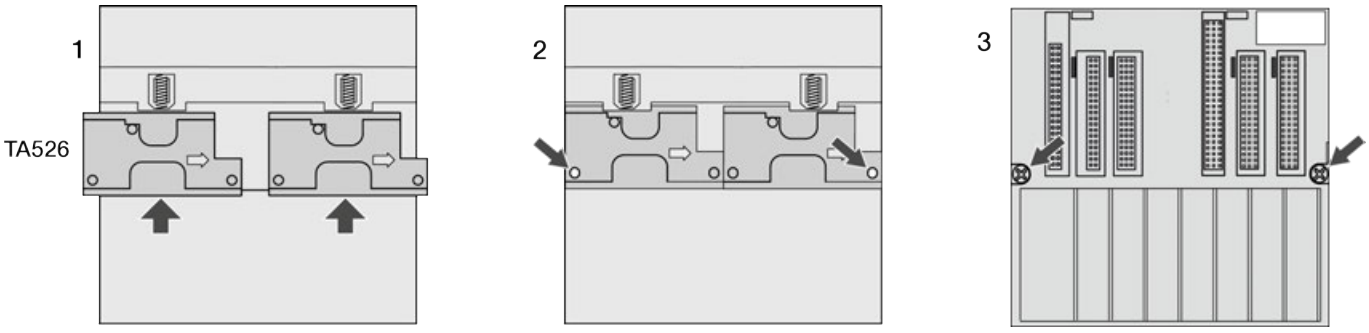


Fig. 286: Function module terminal bases, Fastening with screws

By wall mounting, the terminal base is grounded through the screws. It is necessary that

- *the screws have a conductive surface (e.g. steel zinc-plated or brass nickel-plated)*
- *the mounting plate is grounded*
- *the screws have a good electrical contact to the mounting plate*

Practical tip

The following procedure allows you to use the mounted modules as a template for drilling holes in the panel. Due to module mounting hole tolerance, it is important to follow these procedures:

1. On a clean work surface, mount no more than 3 modules (e.g. one terminal base and two terminal units).
2. Using the mounted modules as a template, carefully mark the center of all module-mounting holes on the panel.
3. Return the mounted modules to the clean work surface, including any previously mounted modules.
4. Drill and tap the mounting holes for the screws (M4 or #8 recommended).
5. Place the modules back on the panel and check for proper hole alignment.
6. Attach the modules to the panel using the mounting screws.

> *If mounting more modules, mount only the last one of this group and put the others aside. This reduces remounting time during drilling and tapping of the next group.*

7. Repeat the steps for all remaining modules.

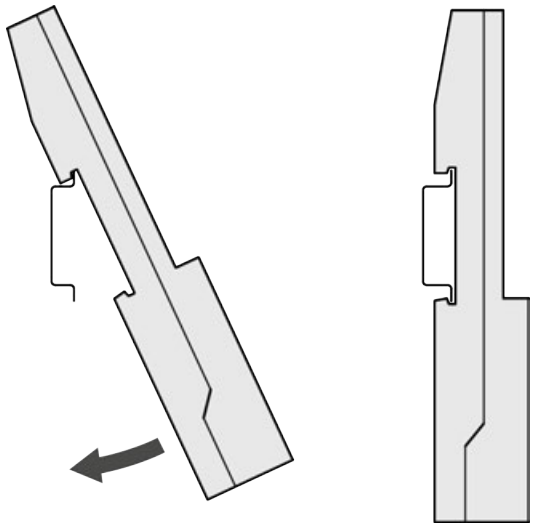
Mounting/Demounting the terminal unit

Mounting on DIN rail

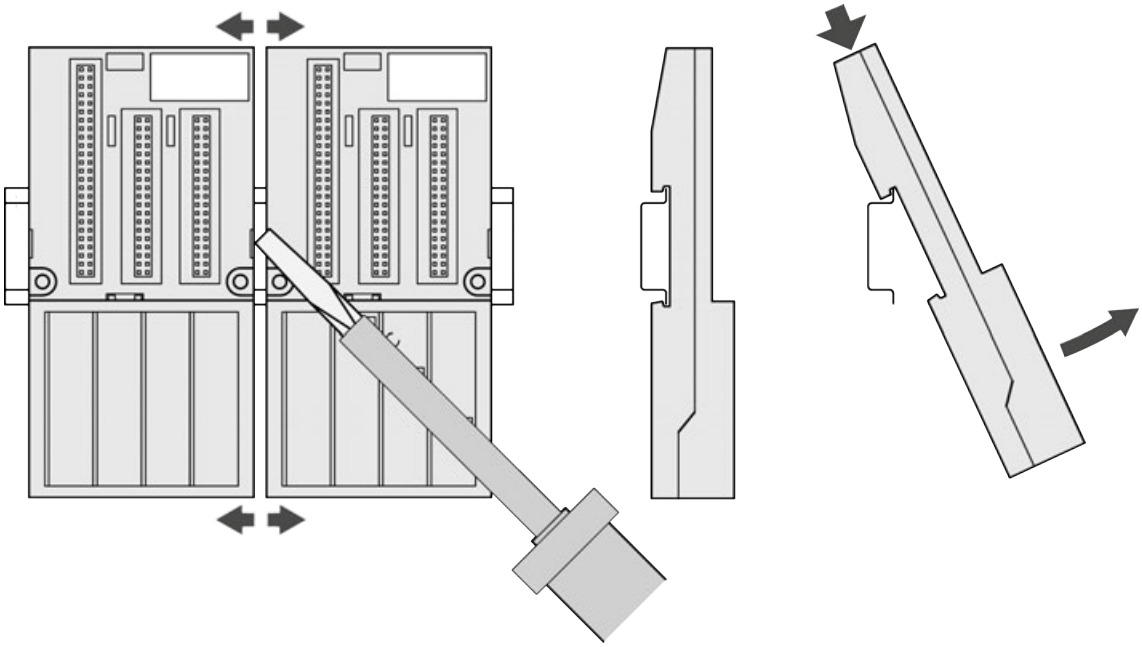
1. Mount DIN rail 7.5 mm or 15 mm.
2. Mount the terminal unit.

   The terminal unit is snapped into the DIN rail in the same way as the Terminal Base. Once secured to the DIN rail, slide the terminal unit to the left until it fully locks into place creating a solid mechanical and connection.

> *When attaching the devices, make sure the bus connectors are securely locked together to ensure proper connection. Max. 10 terminal units can be attached.*



3. Demounting: A screwdriver is inserted in the indicated place to separate the terminal units.



Mounting with screws

If the terminal unit should be mounted with screws, wall mounting accessories TA526 ↳ *Chapter 1.6.4.6.5.5 “TA526 - Wall mounting accessory” on page 3445* must be inserted at the rear side first. These plastic parts prevent bending of the Terminal Base while screwing on.

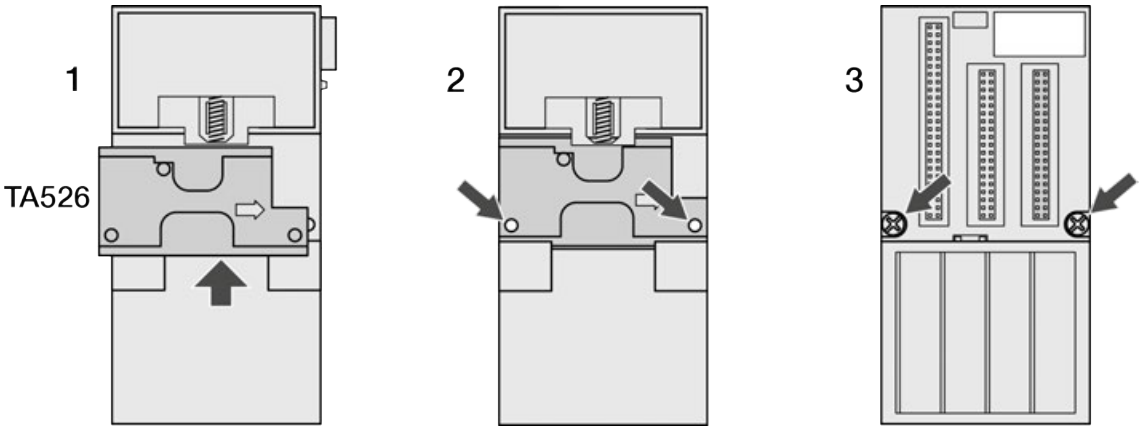


Fig. 287: Fastening with screws

By wall mounting, the terminal unit is grounded through the screws. It is necessary that

- *the screws have a conductive surface (e.g. steel zinc-plated or brass nickel-plated)*
- *the mounting plate is grounded*
- *the screws have a good electrical contact to the mounting plate*

Practical tip

The following procedure allows you to use the mounted modules as a template for drilling holes in the panel. Due to module mounting hole tolerance, it is important to follow these procedures:

1. On a clean work surface, mount no more than 3 modules (e.g. one terminal base and two terminal units).
2. Using the mounted modules as a template, carefully mark the center of all module-mounting holes on the panel.

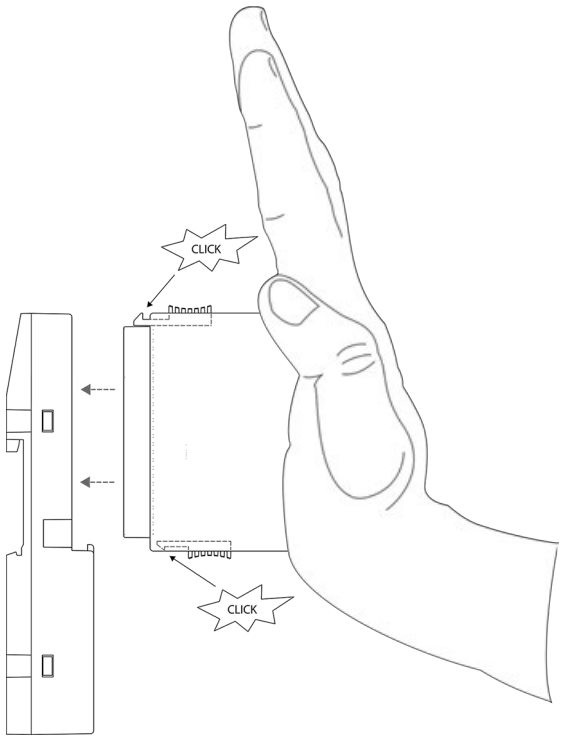
3. Return the mounted modules to the clean work surface, including any previously mounted modules.
4. Drill and tap the mounting holes for the screws (M4 or #8 recommended).
5. Place the modules back on the panel and check for proper hole alignment.
6. Attach the modules to the panel using the mounting screws.

> *If mounting more modules, mount only the last one of this group and put the others aside. This reduces remounting time during drilling and tapping of the next group.*

7. Repeat the steps for all remaining modules.

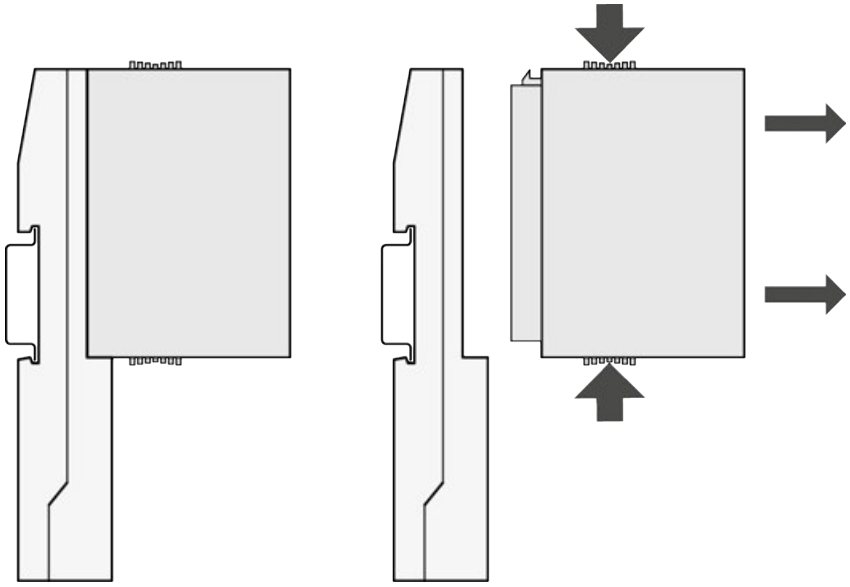
Mounting processor modules PM57x, PM58x, PM59x and PM56xx

1. After mounting the Terminal Base on the DIN rail, mount the processor module.



2. Press the processor module into the Terminal Base until it locks in place.

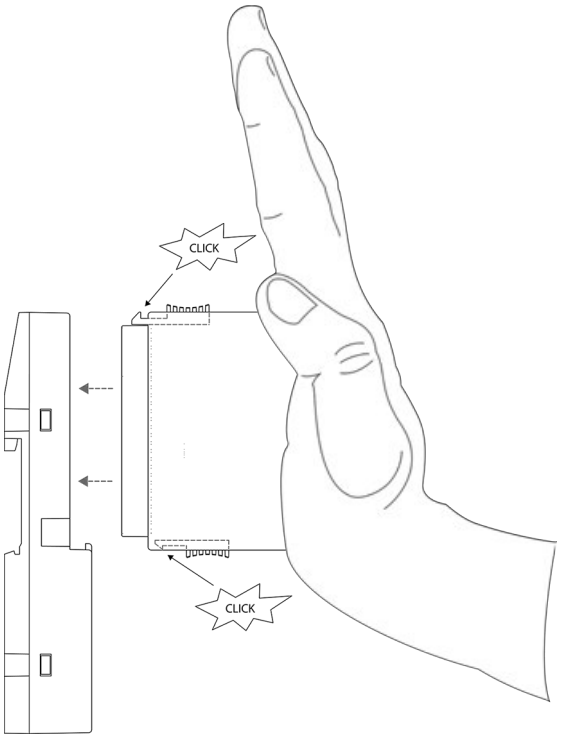
3. The demounting is carried out in a reversed order. Press above and below, then remove the processor module.



Mounting/Demounting the I/O modules

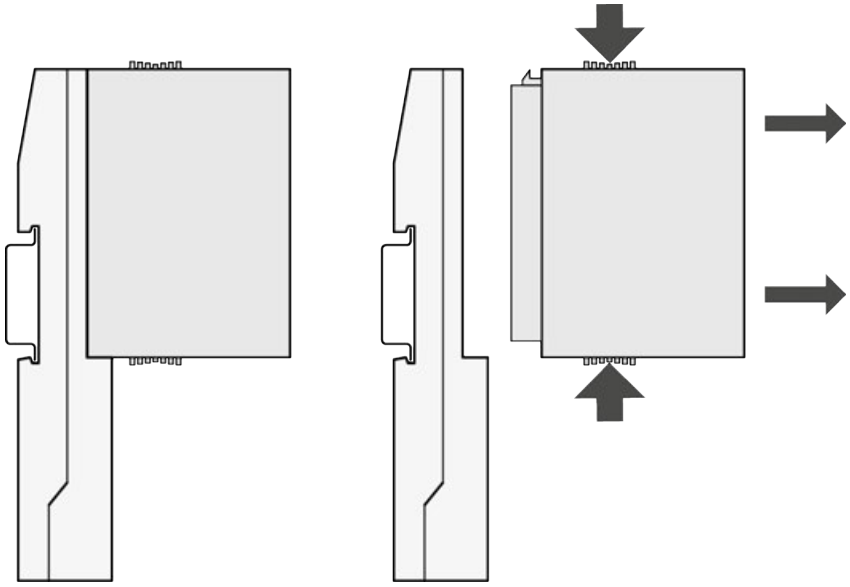
After mounting the terminal unit, mount the I/O modules.

1. Press the I/O module into the terminal unit until it locks in place.



2. The demounting is carried out in a reversed order.

   Press above and below, then remove the module.



Mounting/Demounting the communication modules

Communication modules are mounted on the left side of the processor module on the same terminal base. The connection is established automatically when mounting the communication module.

NOTICE!

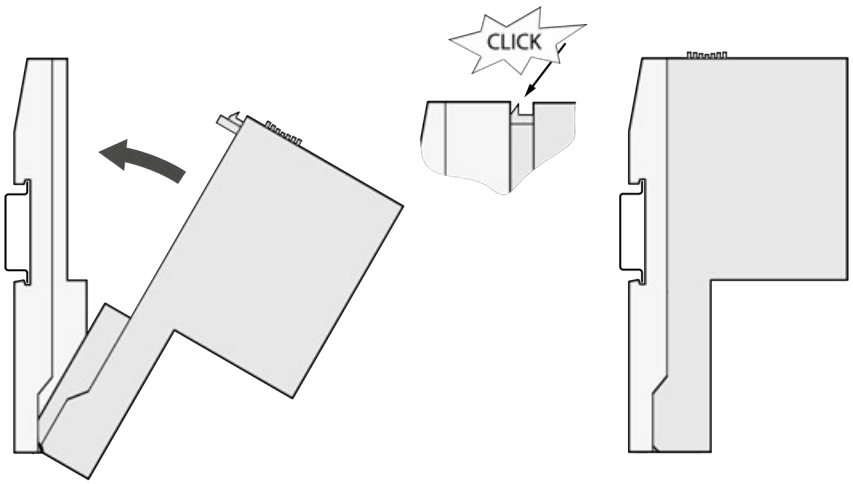
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

After mounting the terminal base, mount the communication modules.

1. First insert the bottom nose of the communication module into the dedicated holes of the terminal base. Then, rotate the communication module on the dedicated terminal base slot until it is locked in place.



⇨

NOTICE!

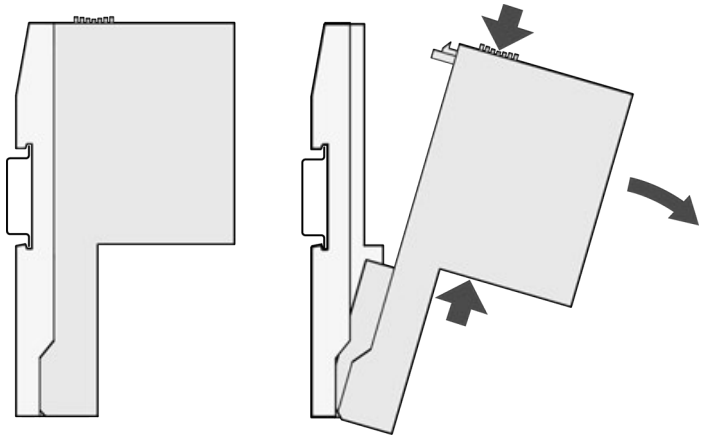
Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↬ *Chapter 1.6.4.6.5.6 “TA524 - Dummy communication module” on page 3446*.
– I/O bus connectors must not be touched during operation.

2. The demounting is carried out in a reversed order.

   Press above and below, then rotate the communication module and remove it.



Mounting/Demounting the accessories

Additional components such as batteries, cables, etc. are required for commissioning the PLC system. Information on assembly, replacement or basic use of the orderable components can be found in the description of the respective accessory.

↬ *Chapter 1.6.4.6.5 “Handling of accessories” on page 3428*

Hardware details can be found in the device specifications of the accessory.

↬ *Chapter 1.6.3.8 “Accessories” on page 3288*

1.6.4.6.4 Connection and wiring

For detailed information such as technical data of your mounted devices (AC500 product family) refer to the hardware device description of the appropriate device.

NOTICE!

Attention:

The devices should be installed by experts who are trained in wiring electronic devices. In case of bad wiring, the following problems could occur:

- On the terminal base, the terminals L+ and M are doubled. If the power supply is badly connected, a short circuit could happen and lead to a destruction of the power supply or its fuse. If no suitable fuse exists, the terminal base itself might be destroyed.
- The terminal bases and all electronic modules and terminal units are protected against reverse polarity.
- All necessary measures should be carried out to avoid damages to modules and wiring. Notice the wiring plans and connection examples.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

NOTICE!

Attention:

Due to possible loss of communcation, the communication cables should be fixed with cable duct or bracket or clamp during application.

Power supply

AC500 system power supply

As soon as the power supply of the processor module (CPU) is higher than the minimum Process and supply voltage (see *Chapter 1.6.4.6.1.1 “Environmental conditions” on page 3398*), the power supply detection is activated and the processor module is started. Power supply of processor module and I/O modules should be powered on the same time, otherwise the processor module will not switch to run after startup.

When during operation the power supply is going down lower than the minimum Process and supply voltage (see *Chapter 1.6.4.6.1.1 “Environmental conditions” on page 3398*) for more than 10 ms, the processor module is switched to safety mode (display shows “AC500”). A restart of the processor module only occurs by switching the power supply off and on again.

If an I/O module is disconnected during normal operation from power supply while processor module is still powered, the processor module will continue its normal operation on all other powered peripherals (I/O modules, communication modules and communication interfaces), but freezes the input image. After recovery of I/O Module power supply it will continue normal operation and inputs and outputs were updated.

Logic Controller Supply: AC500 logic controller power supply is provided through terminals L+ / M.

Process Power Supply: S500 process power supply is provided through terminals UP / ZP.

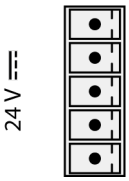
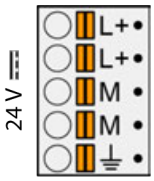
Logic Controller Supply is galvanic isolated from Process Power Supply.

As system power supply for AC500/S500, the ABB CP power supply series can be used.

Power supply for processor modules

The supply voltage of 24 V DC is connected to a removable 5-pin terminal block. L+/M exist twice. It is therefore possible to feed e.g. external sensors (up to 8 A max. with 1.5 mm^2 conductor) via these terminals.

Pin assignment

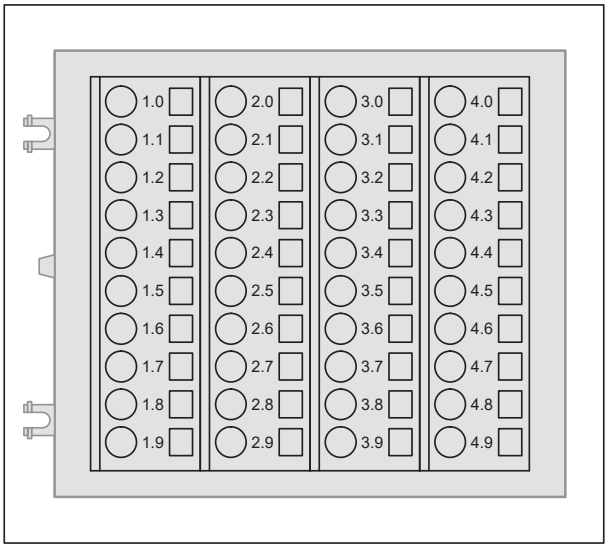
Pin Assignment		Label	Function	Description
 Terminal block removed	 Terminal block inserted	L+	+24 V DC	Positive pin of the power supply voltage
		L+	+24 V DC	Positive pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		⏚	FE	Functional earth

Terminals for power supply and the COM1 interface

Terminal type: Spring terminal

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 1.5 mm^2
1	Flexible	0.08 mm^2 to 1.5 mm^2
1 with wire-end ferrule (without plastic sleeve)	Flexible	0.25 mm^2 to 1.5 mm^2
1 with wire-end ferrule (with plastic sleeve)	Flexible	0.25 mm^2 to 0.5 mm^2
1 (TWIN wire end ferrule)	Flexible	0.5 mm^2

Terminals at the terminal unit



Terminal type: Screw-type terminal

Front terminal, conductor connection vertically with respect to the printed circuit board.

Parameter	Value
Type	Front terminal
Degree of protection	IP 20
Stripped conductor end	9 mm, min. 8 mm
Fastening torque	0.6 Nm
Needed tool	Slotted screwdriver
Dimensions	Blade diameter 3.5 mm

Terminal units with product index < C0 e. g. 1SAP 212 200 R0001 B0

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 2.5 mm^2
1	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule	Flexible	0.25 mm^2 to 1.5 mm^2
2	Solid	Not intended
2	Flexible	Not intended
2 with TWIN wire end ferrule (length 10 mm) with plastic sleeve	Flexible	2 x 0.25 mm^2 or 2 x 0.5 mm^2 or 2 x 0.75 mm^2, with square cross-section of the wire-end ferrule also 2 x 1.0 mm^2

Terminal units with product index ≥ C0 e. g. 1SAP 212 200 R0001 C0

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 2.5 mm^2
1	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule without plastic sleeve	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule with plastic sleeve	Flexible	0.14 mm^2 to 1.5 mm^2
2	Solid	0.08 mm^2 to 1.5 mm^2
2	Flexible	0.08 mm^2 to 1.5 mm^2
2 with TWIN wire end ferrule (length 10 mm) with plastic sleeve	Flexible	2 x 0.5 mm^2 to 2 x 1.0 mm^2
2 with separate wire-end ferrule without plastic sleeve	Flexible	0.08 mm^2 to 0.75 mm^2

Terminal type: Spring terminal

Front terminal, conductor connection vertically with respect to the printed circuit board.

Parameter	Value
Type	Front terminal
Degree of protection	IP 20
Stripped conductor end	9 mm, min. 8 mm
Needed tool	Slotted screwdriver
Dimensions	2.5 x 0.4 to 3.5 x 0.5 mm, screwdriver must be at least 15 mm free of insulation at the tip

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 2.5 mm^2
1	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule	Flexible	0.25 mm^2 to 1.5 mm^2
2	Solid	Not intended
2	Flexible	Not intended
2 with TWIN wire end ferrule (length 10 mm) with plastic sleeve	Flexible	2 x 0.25 mm^2 or 2 x 0.5 mm^2 or 2 x 0.75 mm^2, with square cross-section of the wire-end ferrule also 2 x 1.0 mm^2

Connection of wires at the spring terminals

Connection

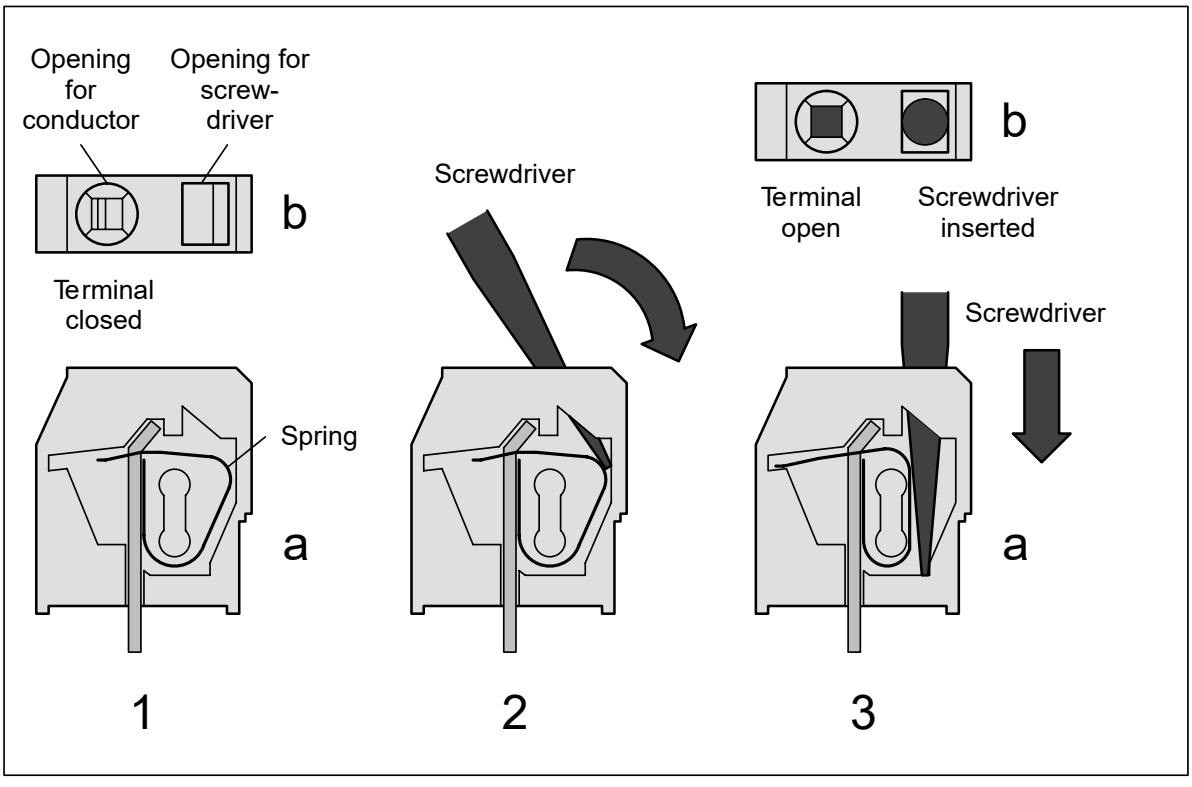


Fig. 288: Connect the wire to the spring terminal (steps 1 to 3)

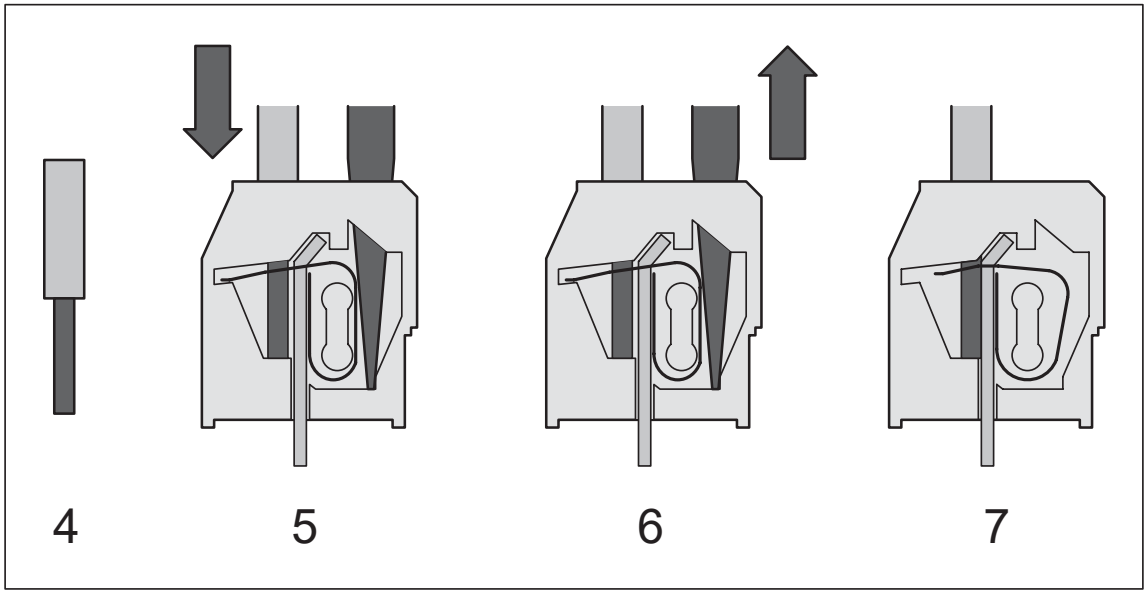


Fig. 289: Connect the wire to the spring terminal (steps 4 to 7)

1. Side view (open terminal drawn for illustration)
2. The top view shows the openings for wire and screwdriver
3. Insert screwdriver (2.5 x 0.4 to 3.5 x 0.5 mm) at an angle, screwdriver must be at least 15 mm free of insulation at the tip
4. While erecting the screwdriver, insert it until the stop (requires a little strength)
5. Screwdriver inserted - terminal open
6. Strip the wire for 7 mm (and put on wire-end ferrule)
7. Insert wire into the open terminal
8. Done

Disconnection

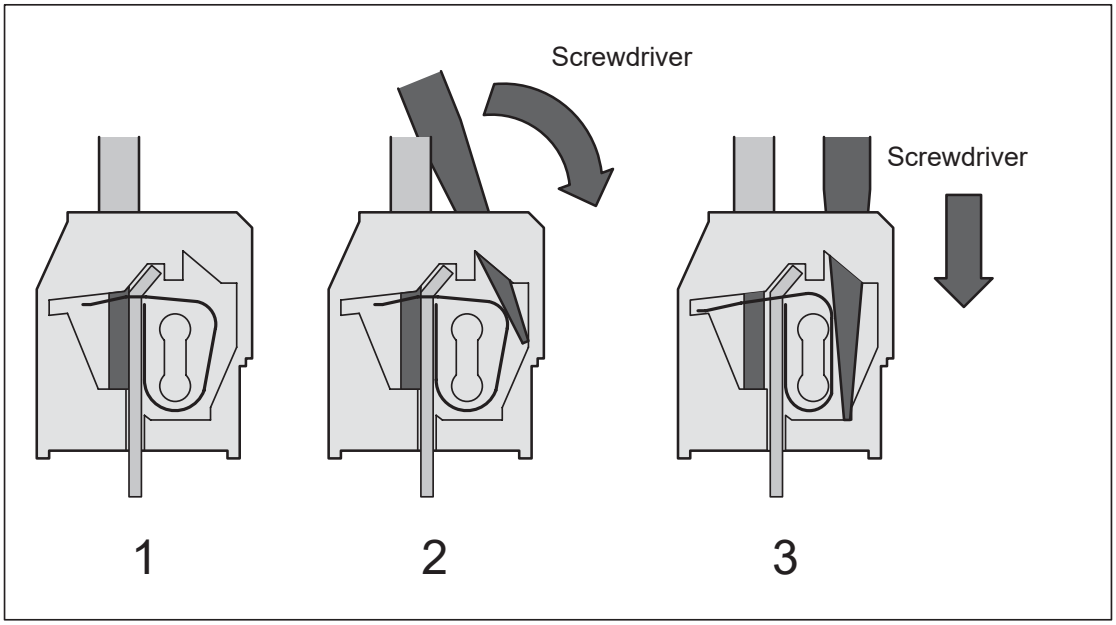


Fig. 290: Disconnect wire from the spring terminal (steps 1 to 3)

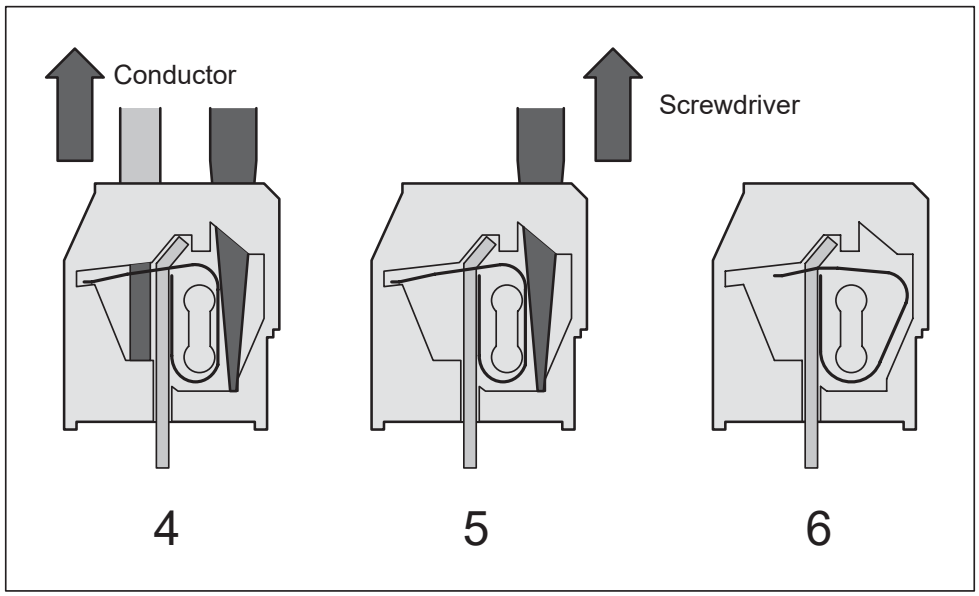


Fig. 291: Disconnect wire from the spring terminal (steps 4 to 6)

1. Terminal with wire connected
2. Insert screwdriver (2.5 x 0.4 to 3.5 x 0.5 mm) at an angle, screwdriver must be at least 15 mm free of insulation at the tip
3. While erecting the screwdriver, insert it until the stop (requires a little strength) - terminal is now open
4. Remove wire from the open terminal
5. Done

Terminals for CANopen/DeviceNet communication modules

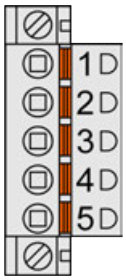


Fig. 292: Combicon, 5-pole, female, removable plug with spring terminals

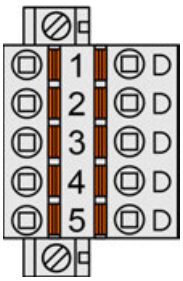


Fig. 293: Combicon, 5-pole, female, removable plug with spring terminals

Terminal type: Spring terminal

Number of cores per terminal	Conductor type	Cross section	Stripped conductor end
1	solid	0.2 mm^2 to 2.5 mm^2	10 mm
1	flexible	0.2 mm^2 to 2.5 mm^2	10 mm
1 with wire-end ferrule (without plastic sleeve)	flexible	0.25 mm^2 to 2.5 mm^2	10 mm
1 with wire-end ferrule (with plastic sleeve)	flexible	0.25 mm^2 to 2.5 mm^2	10 mm

CANopen field bus

Types of bus cables

For CANopen, only bus cables with characteristics as recommended in ISO 11898 are to be used. The requirements for the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	Bus cable (shielded, twisted pair)			Max. transmission rate [kbit/s]
	Conductor cross section [mm^2]	Line resistance [Ω/km]	Wave impedance [Ω]	
0...40	0.25...0.34 / AWG23, AWG22	70	120	1000 at 40 m
40...300	0.34...0.60 / AWG22, AWG20	< 60	120	< 500 at 100 m
300...600	0.50...0.60 / AWG20	< 40	120	< 100 at 500 m
600...1000	0.75...0.80 / AWG18	< 26	120	< 50 at 1000 m

NOTICE!

Risk of telegram and data errors!

The use of wrong cable type and quality could lead to limitations in cable length, causing telegram and data errors.

NOTICE!

Risk of damaging the terminating resistor!

A bus-line short-circuit to the 24 V DC power supply can cause damage by exceeding the power rating of the terminating resistor.

NOTICE!

Risk of telegram and data errors!

Miss- or unterminated data lines can cause reflections on the bus, leading to telegram and data errors. For maximum cable length and transmission rate, the bus must always be terminated on both ends with the characteristic impedance of the cable type.

NOTICE!

Verification of termination (Make sure the power supply on all CAN nodes is turned off)!

To verify the termination, the DC resistance between CAN_H and CAN_L can be measured. The value should be between 50 Ω and 70 Ω.

Check for correct resistor values, short circuits and correct number of terminating resistors, if the measurement is showing deviations.

Installation hint

Ensure that the termination and FE connection will not be removed when removing CAN modules from the bus.

Branches are not allowed in a CAN network. Stubs should be avoided or kept as short as possible (< 0.3 m).

When connecting the cable take care to use one dedicated twisted pair for the CAN signals (CAN_L and CAN_H) and another free wire for CAN_GND. CAN_GND must be connected as reference, to avoid common mode problems causing telegram errors.

Keep the CAN bus wiring away from electrical disturbance and close to earth potential to minimize interference.

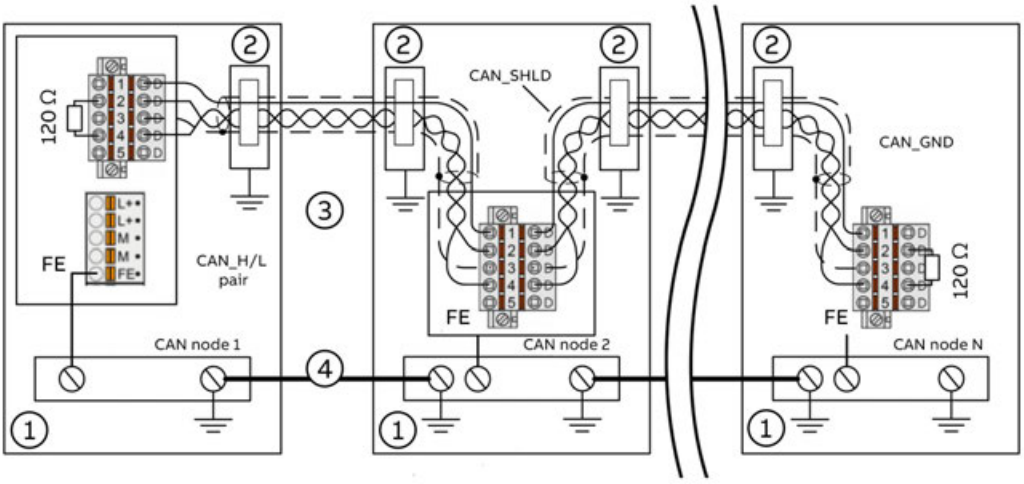


Fig. 294: CAN bus, connection and wiring

1 Cabinet
2 Direct earthing of shields when entering the cabinet
3 CAN bus segment
4 Current-carrying connection

Ethernet connection details

Ethernet is also used for PROFINET, EtherCAT and Modbus TCP connection.

Ethernet interface

Pin assignment

Interface	Pin	Signal	Description
8 RJ45 1	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used
	6	RxD-	Receive data -
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

See supported protocols and used Ethernet ports: *Chapter 1.6.5.1.7.1.2 “Ethernet protocols and ports for AC500 V3 products” on page 3515*.

See communication via Modbus TCP/IP: *Chapter 1.6.5.1.11 “Communication with Modbus TCP/IP” on page 3558*.

See communication via Modbus RTU: *Chapter 1.6.5.1.10 “Communication with Modbus RTU” on page 3542*.

Wiring

Cable length restrictions

For the maximum possible cable lengths within an Ethernet network, various factors have to be taken into account. Twisted pair cables (TP cables) are used as transmission medium for 10 Mbit/s Ethernet (10Base-T) as well as for 100 Mbit/s (Fast) Ethernet (100Base-TX). For a transmission rate of 10 Mbit/s, cables of at least category 3 (IEA/TIA 568-A-5 Cat3) or class C (according to European standards) are allowed. For fast Ethernet with a transmission rate of 100 Mbit/s, cables of category 5 (Cat5) or class D or higher have to be used. The maximum length of a segment, which is the maximum distance between two network components, is restricted to 100 m due to the electric properties of the cable.

Furthermore, the length restriction for one collision domain has to be observed. A collision domain is the area within a network which can be affected by a possibly occurring collision (i.e. the area the collision can propagate over). This, however, only applies if the components operate in half-duplex mode since the CSMA/CD access method is only used in this mode. If the components operate in full-duplex mode, no collisions can occur. Reliable operation of the collision detection method is important, which means that it has to be able to detect possible collisions even for the smallest possible frame size of 64 bytes (512 bits). But this is only guaranteed if the first bit of the frame arrives at the most distant subscriber within the collision domain before the last bit has left the transmitting station. Furthermore, the collision must be able to propagate to both directions at the same time. Therefore, the maximum distance between two ends must not be longer than the distance corresponding to the half signal propagation time of 512 bits. Thus, the resulting maximum possible length of the collision domain is 2000 m for a transmission rate of 10 Mbit/s and 200 m for 100 Mbit/s. In addition, the bit delay times caused by the passed network components also have to be considered.

The following table shows the specified properties of the respective cable types per 100 m.

Table 609: Specified cable properties:

Parameter	10Base-T [10 MHz]	100Base-TX [100 MHz]
Attenuation [dB / 100m]	10.7	23.2
NEXT [dB / 100m]	23	24
ACR [dB / 100m]	N/A	4
Return loss [dB / 100m]	18	10
Wave impedance [Ohms]	100	100
Category	3 or higher	5
Class	C or higher	D or higher

TP cable

The TP cable has eight wires arranged in four pairs of twisted wires. Different color codes exist for the coding of the wires, the coding according to EIA/TIA 568, version 1, being the one most commonly used. In this code, the individual pairs are coded with blue, orange, green and brown color. One wire of a pair is unicolored and the corresponding second wire is striped, the respective color alternating with white. For shielded cables, a distinction is made between cables that have one single shield around all pairs of wires and cables that have an additional individual shield for each pair of wires. The following table shows the different color coding systems for TP cables:

Table 610: Color coding of TP cables:

Pairs	EIA/TIA 568 Version 1		EIA/TIA 568 Version 2		DIN 47100		IEC 189.2	
Pair 1	white/ blue	blue	green	red	white	brown	white	blue
Pair 2	white/ orange	orange	black	yellow	green	yellow	white	orange

Pairs	EIA/TIA 568 Version 1		EIA/TIA 568 Version 2		DIN 47100		IEC 189.2	
Pair 3	white/ green	green	blue	orange	grey	pink	white	green
Pair 4	white/ brown	brown	brown	slate	blue	red	white	brown

Two general variants are distinguished for the pin assignment of the normally used RJ45 connectors: EIA/TIA 568 version A and version B. The wiring according to EIA/TIA 568 version B is the one most commonly used.

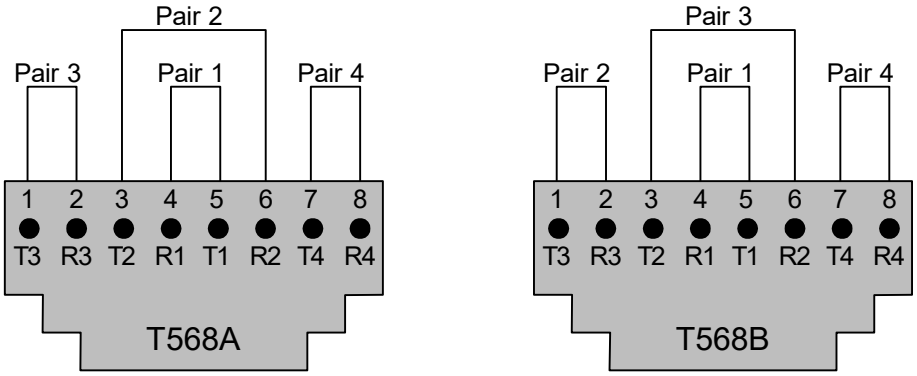


Fig. 295: Pin assignment of RJ45 sockets

Cable types

Crossover cable

> ***Particular use***
> *Crossover cables are needed only for a direct Ethernet connection without crossover functionality. In particular for AC500 modules in product life cycle phase "Classic".*

Crossover cables are for a direct Ethernet connection of two terminal devices as the simplest variant of a network. From transmission lines of the first station to the reception lines of the second station.

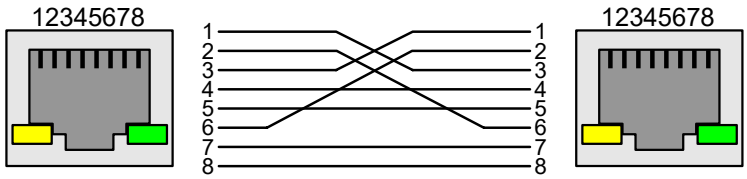


Fig. 296: Wiring of a crossover cable

Straight-through cable

For networks with more than two subscribers, hubs or switches have to be used additionally for distribution. These active devices already have the crossover functionality implemented which allows a direct connection of the terminal devices using straight-through cables.

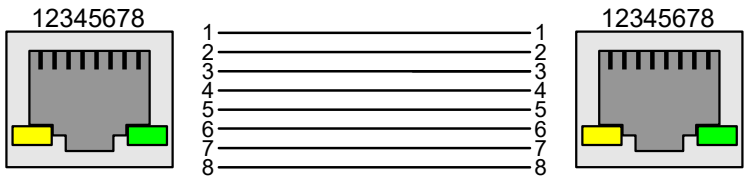


Fig. 297: Wiring of a straight-through cable

CAUTION!
Risk of communication faults!
When using inappropriate cables, malfunctions in communication may occur.

Only use network cables of the categories 5 (Cat 5, Cat 5e, Cat 6 or Cat 7) or higher within PROFINET networks.

Modbus RTU connection details

The Modbus RTU protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Available serial interfaces can work as Modbus interfaces simultaneously.

The Modbus client operating mode of an interface is set with the function block COM_MOD_MAST.

Technical data

The Modbus operating mode and the interface parameters are set in the ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*.

Table 611: Description of the Modbus protocol

Parameter	Value
Supported standard	See ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*
Number of connection points	1 client Max. 1 server with RS-232 interface Max. 31 servers with RS-485
Protocol	Modbus
Operating mode	Client/server
Address	Server only
Data transmission control	CRC16
Data transmission speed	From 9,600 bits/s to 115,200 bits/s ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*
Encoding	1 start bit 8 data bits 1 or 2 stop bits 1 parity bit ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*)
Max. cable length for RS-485 on COM1 for AC500 CPU	1.200 m at 19.200 baud

Bus topology Point-to-point with RS-232 or bus topology with RS-485. Modbus is a master-slave protocol.

For further information on Modbus see chapter *Chapter 1.6.5.1.10 “Communication with Modbus RTU” on page 3542*.

1.6.4.6.5 Handling of accessories

This section only describes accessories that are frequently used for system assembly, connection and construction. A description of all additional accessories that can be used to supplement AC500 system can be found in the Hardware PLC device description.

MC502 - Memory card

- Solid state flash memory storage

1 MC502 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

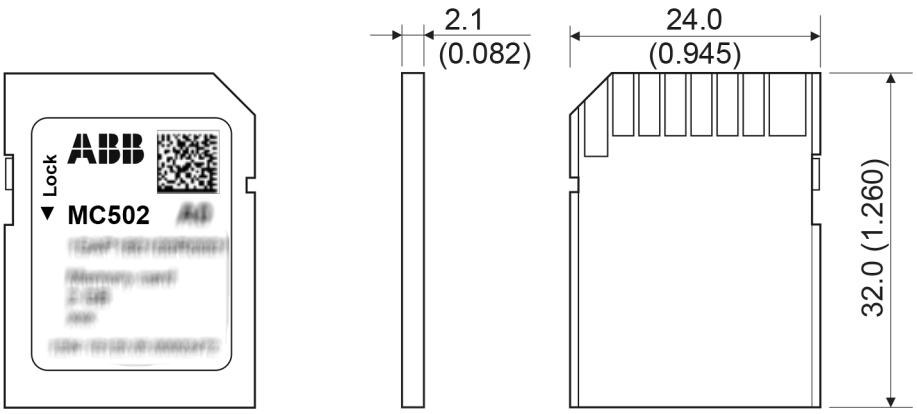
Processor modules are supplied without (micro) memory card. It must be ordered separately.

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V3

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

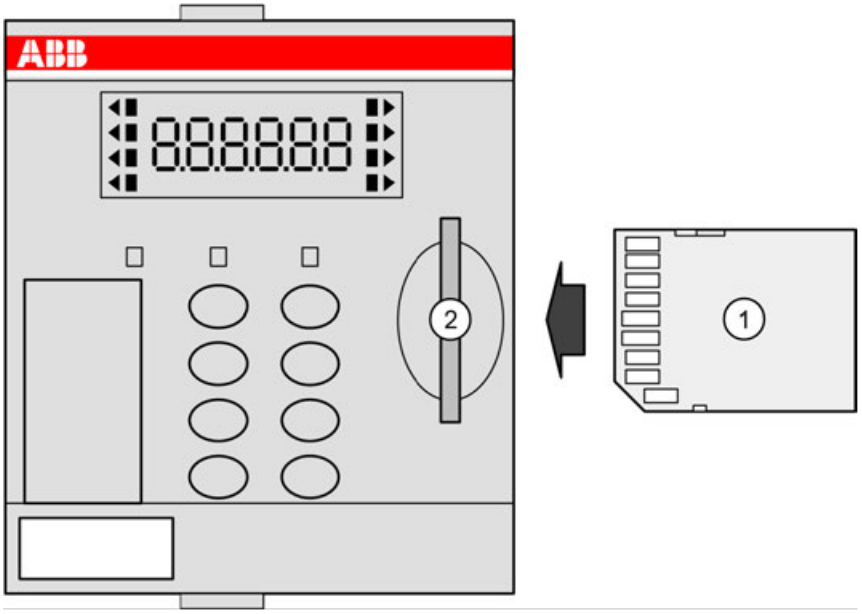


Fig. 298: Insert memory card into PM56xx

1 Memory card
2 Memory card slot

Remove the memory card

AC500 V3

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when no black square (■) is shown next to MC in the display.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

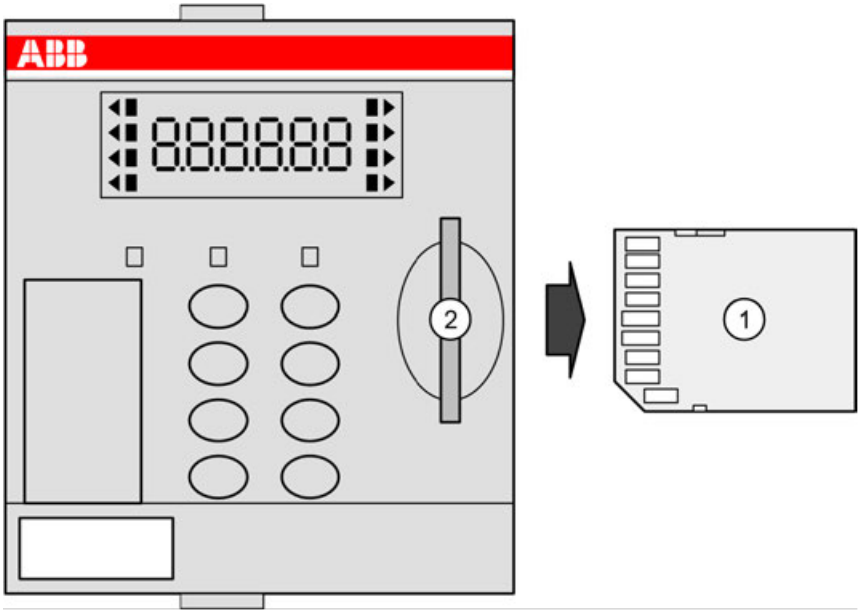


Fig. 299: Remove memory card from PM56xx

1 Memory card
2 Memory card slot

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter
Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999.

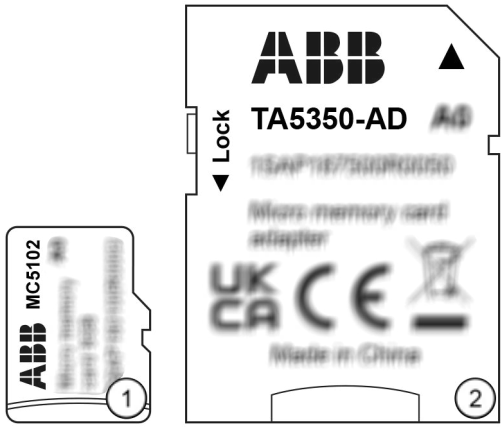
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0001	MC502, memory card	Classic

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.

The TA5350-AD micro memory card adapter has a write protect switch.

In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

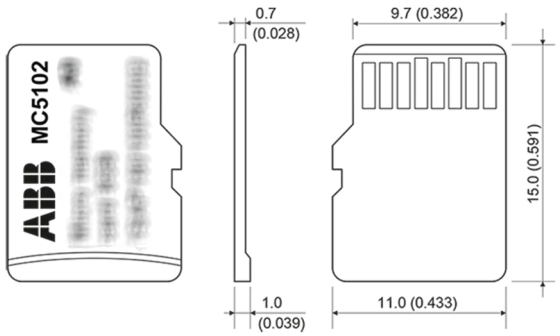
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

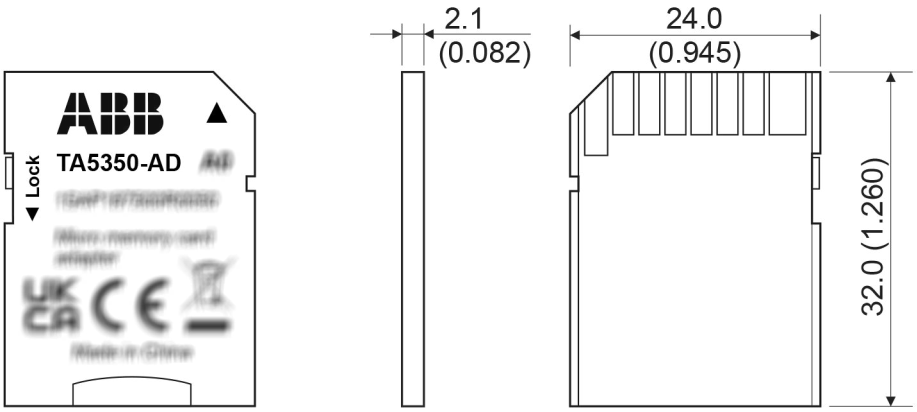
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V3

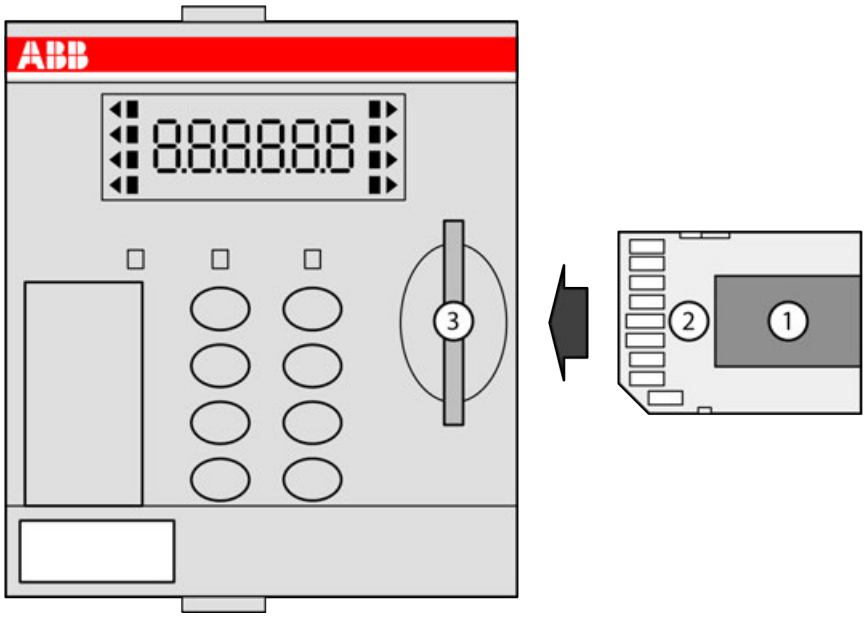
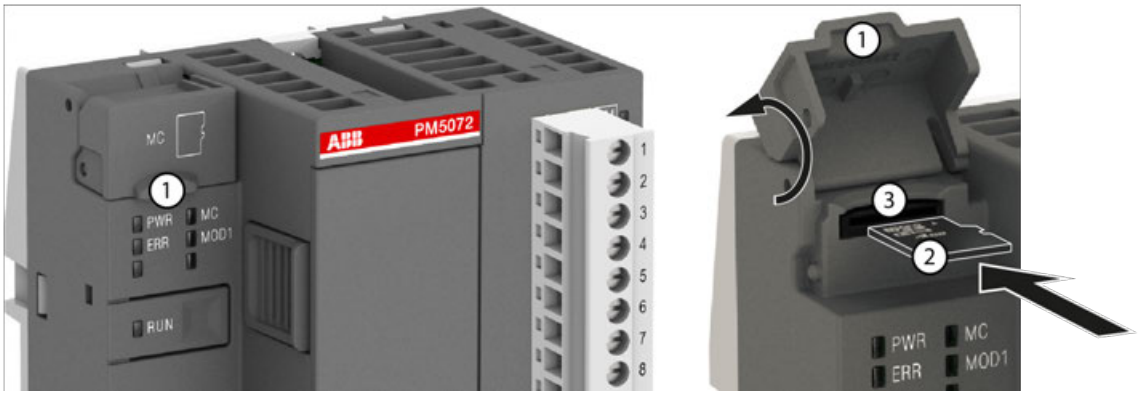


Fig. 300: Insert micro memory card into PM56xx

1 Micro memory card
2 TA5350-AD micro memory card adapter
3 Memory card slot

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Carefully insert the micro memory card into the micro memory card slot as far as it will go. Observe orientation of card.
3. Close the micro memory card slot cover by turning it downwards.

Remove the micro memory card

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

AC500 V3: Remove the micro memory card with micro memory card adapter only when no black square (■) is shown next to MC in the display.

AC500-eCo V3: Remove the micro memory card only when the MC LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

AC500 V3

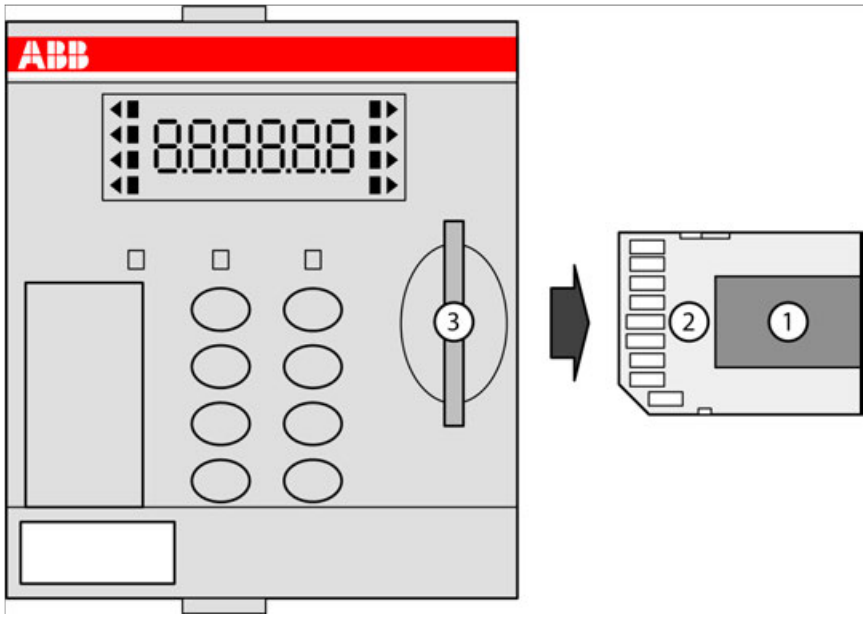
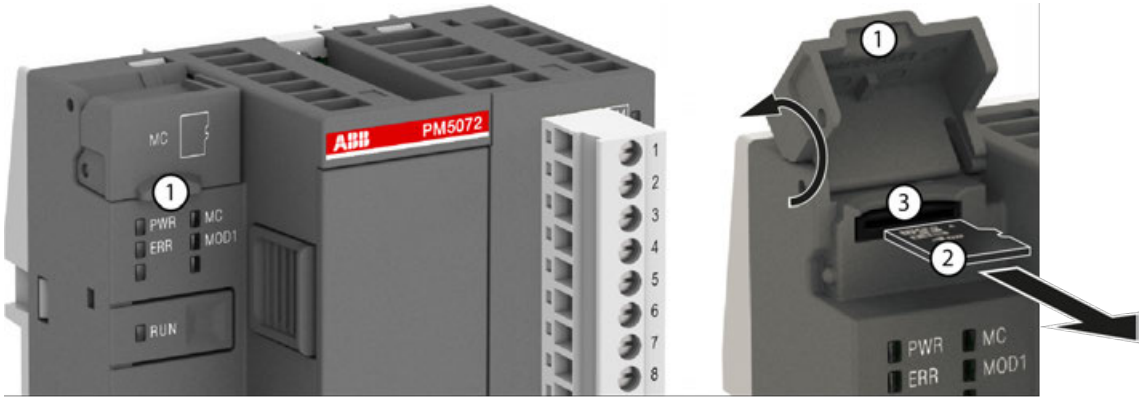


Fig. 301: Remove micro memory card from PM56xx

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

AC500-eCo V3



1 Micro memory card slot cover
2 Micro memory card
3 Micro memory card slot

1. Open the micro memory card slot cover by turning it upwards.
2. Micro memory card can be removed from the micro memory card slot by gripping and pulling with two fingers.
3. Close the micro memory card slot cover by turning it downwards.

Technical data

Parameter	Value
Memory capacity	8 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter ↳ *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC5141 - Memory card

- Solid state flash memory storage

1 MC5141 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

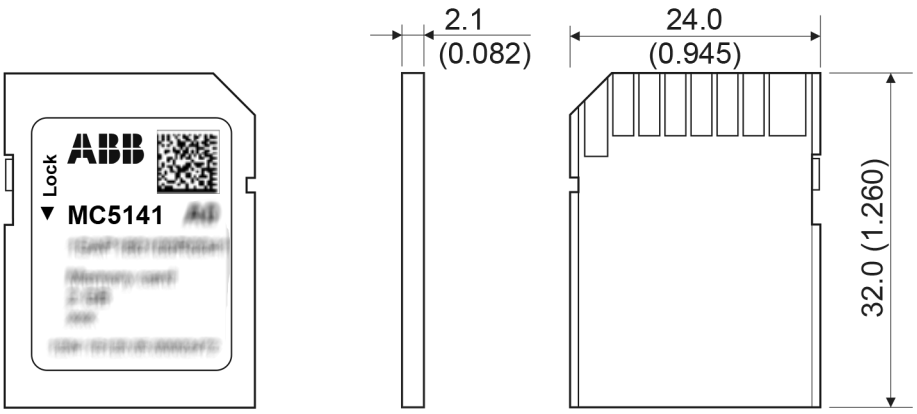
Processor modules are supplied without (micro) memory card. It must be ordered separately.

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V3

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

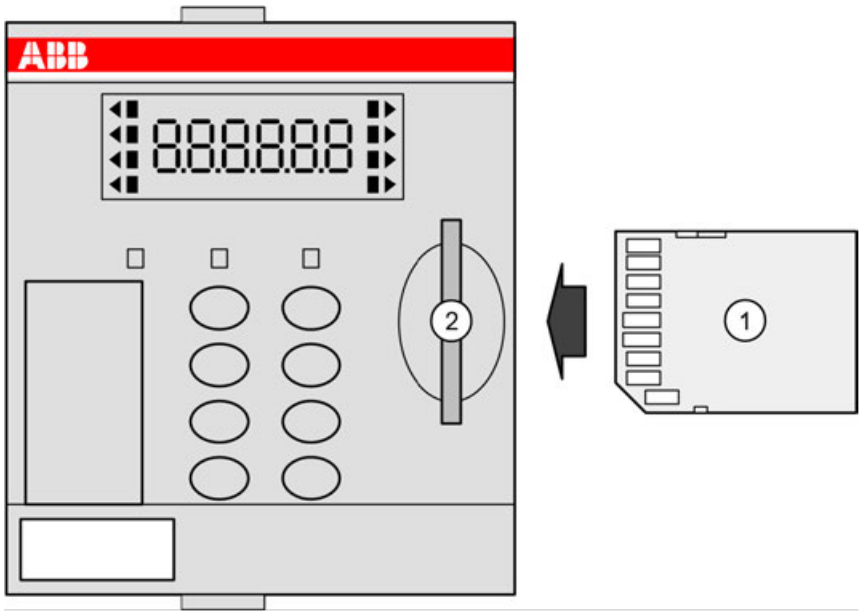


Fig. 302: Insert memory card into PM56xx

1 Memory card
2 Memory card slot

Remove the memory card

AC500 V3

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when no black square (■) is shown next to MC in the display.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

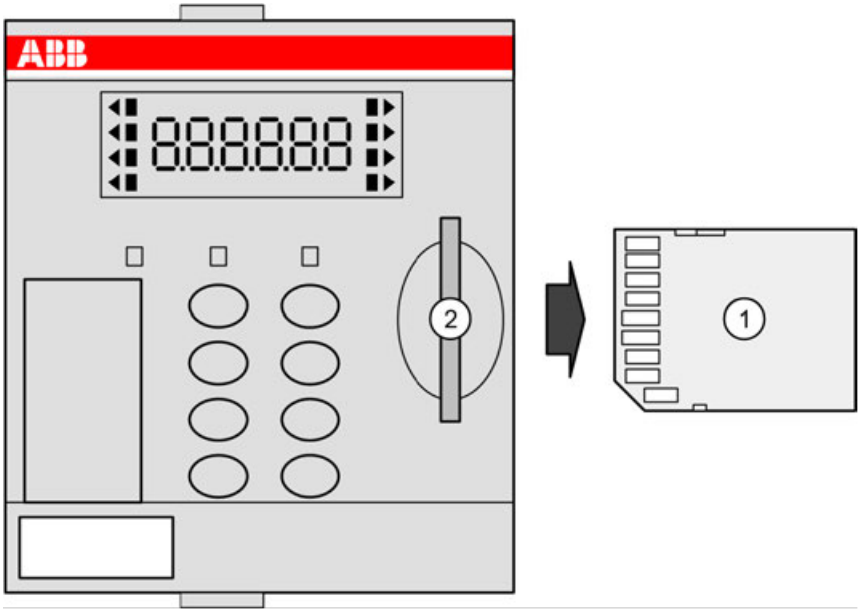


Fig. 303: Remove memory card from PM56xx

1 Memory card
2 Memory card slot

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

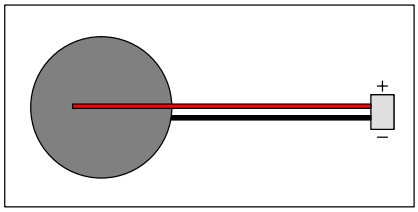
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0041	MC5141, memory card	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA521 - Battery

- Manganese dioxide lithium battery, 3 V, 560 mAh
- Non-rechargeable



Purpose

The TA521 battery is the only applicable battery for the AC500 processor modules *Chapter 1.6.3.3.2.1 “PM56xx-2ETH for AC500 V3 products” on page 2516*. It cannot be recharged.

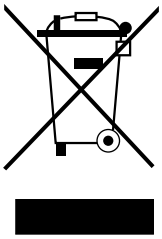
The processor modules are supplied without lithium battery. It must be ordered separately. The TA521 lithium battery is used for data (SRAM) and RTC buffering while the processor module is not powered.

See system technology - AC500 battery. *Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479*

The CPU monitors the discharge degree of the battery. A warning is issued before the battery condition becomes critical (about 2 weeks before). Once the warning message appears, the battery should be replaced as soon as possible.

Handling instructions

- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Replace the battery with supply voltage ON in order not to risk data being lost.
- Recycle exhausted batteries meeting the environmental standards.



Battery lifetime

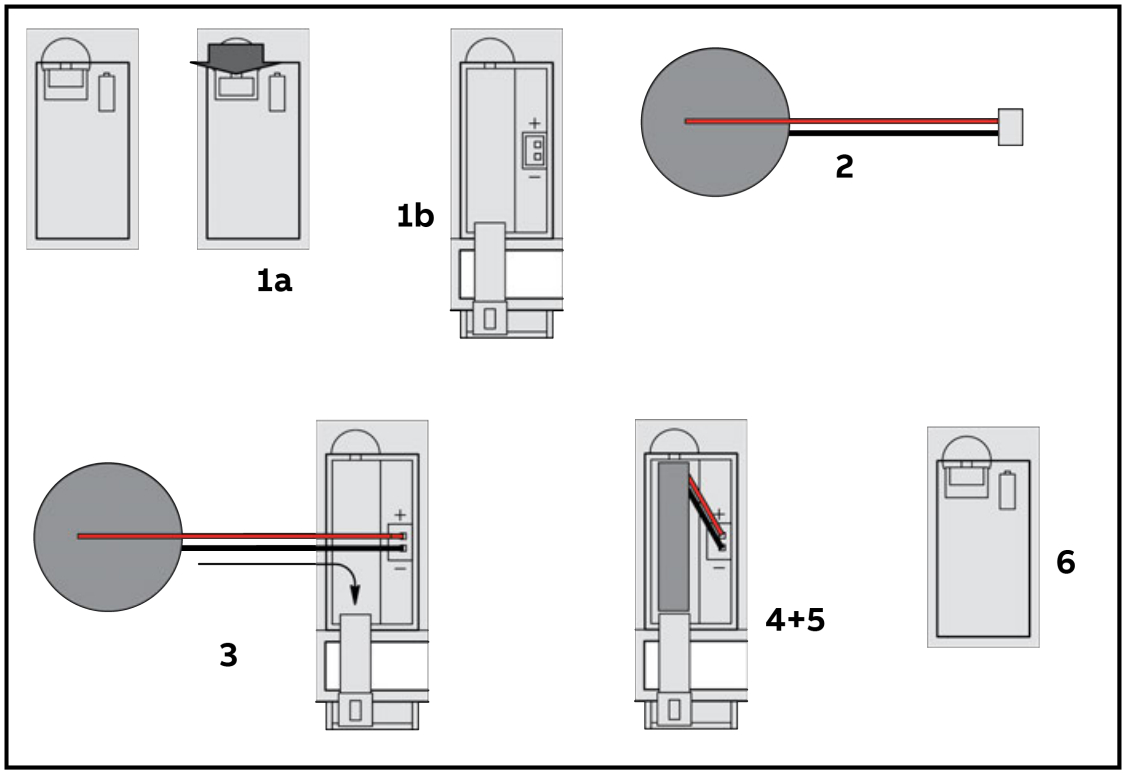
The battery lifetime is the time, the battery can store data while the processor module is not powered. As long as the processor module is powered, the battery will only be discharged by its own leakage current.

To avoid a short battery discharge, the battery should always be inserted or replaced while the process module is under power, then the battery is correctly recognized and will not shortly discharged.

Insertion

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

WARNING!
Risk of fire or explosion!
Use of incorrect Battery may cause fire or explosion.



1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front face of the processor module and cannot be removed.
2. Remove the TA521 battery from its package and hold it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.
3. Insert the battery connector into the small connector port of the compartment. The connector is keyed to find the correct polarity (red = positive pole = above).
4. Insert first the cable and then the battery into the compartment, push it until it reaches the bottom of the compartment.
5. Arrange the cable in order not to inhibit the door to close.
6. Pull-up the door and press until the locking mechanism snaps.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Replacement of the battery

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front view of the processor module and cannot be removed.
2. Remove the old TA521 battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.

3. Follow the previous instructions to insert a new battery.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Technical data

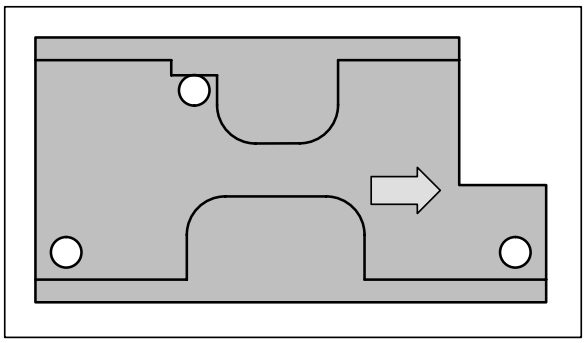
Parameter	Value
Nominal voltage	3 V
Nominal capacity	560 mAh
Temperature range (index below C0)	Operating: 0 °C...+60 °C Storage: -20 °C...+60 °C Transport: -20 °C...+60 °C
Temperature range (index C0 and above)	Operating: -40 °C...+70 °C Storage: -40 °C...+85 °C Transport: -40 °C...+85 °C
Battery lifetime	Typ. 3 years at 25 °C
Self-discharge	2 % per year at 25 °C 5 % per year at 40 °C 20 % per year at 60 °C
Protection against reverse polarity	Yes, by mechanical coding of the plug.
Insulation	The battery is completely insulated.
Connection	Red = positive pole = above at plug, black = negative pole,
Weight	7 g
Dimensions	Diameter of the button cell: 24.5 mm Thickness of the button cell: 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 300 R0001	TA521, lithium battery	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA526 - Wall mounting accessory



Purpose

If a terminal base TB5xx or a terminal unit TU5xx should be mounted with screws, the wall mounting accessories TA526 must be inserted at the rear side first. This plastic parts prevent bending of terminal bases and terminal units while screwing up.

Handling instructions

Handling of the wall mounting accessory is described in detail in the section *Mounting and disassembling the terminal unit* *“Mounting with screws” on page 3411* and *Mounting/Disassembling Terminal Bases and Function Module Terminal Bases* *“Mounting with screws” on page 3409*.

Technical data

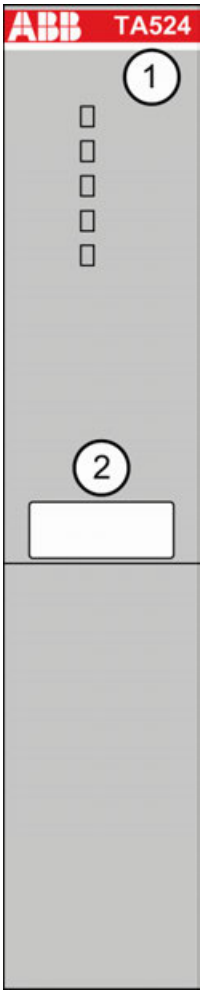
Parameter	Value
Weight	5 g
Dimensions	67 mm x 35 mm x 5,5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 800 R0001	TA526, wall mounting accessory	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA524 - Dummy communication module



1 Type
2 Label

Purpose

TA524 is used to cover an unused communication module slot of a terminal base ↳ *Chapter 1.6.3.2.1 “TB56xx for AC500 V3 products” on page 2430*. It protects the terminal base from dust and inadvertent touch.

Handling instructions

TA524 is mounted in the same way as a common communication module ↳ *Chapter 1.6.4.6.3.5 “Mounting/Demounting the communication modules” on page 3414*.

Technical data

Parameter	Value
Weight	50 g
Dimensions	135 mm x 28 mm x 62 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 600 R0001	TA524, dummy communication module	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CP-E - Economic range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

- Wide-range input voltage
- Mounting on DIN rail
- High efficiency of up to 90 %
- Low power dissipation and low heating
- Wide ambient temperature range from -40 °C...+70 °C
- No-load-proof, overload-proof, continuous short-circuit-proof
- Power factor correction (depending on the type)
- Approved in accordance with all relevant international standards

Table 612: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR427030R0000	CP-E 24/0.75	100-240 V AC or 120-370 V DC	24 V DC, 0.75 A	-	22.5
1SVR427031R0000	CP-E 24/1.25	100-240 V AC or 90-375 V DC	24 V DC, 1.25 A	-	40.5
1SVR427032R0000	CP-E 24/2.5	100-240 V AC or 90-375 V DC	24 V DC, 2.5 A	-	40.5
1SVR427034R0000	CP-E 24/5.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 5 A	-	63.2
1SVR427035R0000	CP-E 24/10.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 10 A	-	83
1SVR427036R0000	CP-E 24/20.0	115-230 V AC or 120-370 V DC	24 V DC, 20 A	-	175

CP-C.1 - High performance range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

The CP-C.1 power supplies are ABB's high performance and most advanced range. With excellent efficiency, high reliability and innovative functionality it is prepared for the most demanding industrial applications. These power supplies have a 50 % integrated power reserve and operate at an efficiency of up to 94 %. They are equipped with overheat protection and active power factor correction. Combinded with a broad AC and DC input range and extensive worldwide approvals the CP-C.1 power supplies are the preferred choice for professional DC applications.

- Typical efficiency of up to 94 %
- Power reserve design delivers up to 150 % of the nominal output current
- Signaling outputs for DC OK and power reserve mode
- High power density leads to very compact and small devices
- No-load-proof, overload-proof, continuous short-circuit-proof
- Active power factor correction (PFC)

Table 613: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR360563R1001	CP-C.1 24/5.0	110-240 V AC or 90-300 V DC	24 V DC, 5 A	+50 %	40
1SVR360663R1001	CP-C.1 24/10.0	110-240 V AC or 90-300 V DC	24 V DC, 10 A	+50 %	60
1SVR360763R1001	CP-C.1 24/20.0	110-240 V AC or 90-300 V DC	24 V DC, 20 A	+30 %	82

1.6.4.7 AC500-XC

1.6.4.7.1 System data AC500-XC

Assembly, construction and connection of devices of the variant AC500-XC is identical to AC500 (standard) ↬ Chapter 1.6.4.6 “AC500 (Standard)” on page 3398. The following description provides information on general technical data of AC500-XC system.

Environmental conditions

Table 614: Process and supply voltages

Parameter		Value
24 V DC		
	Voltage	24 V (-15 %, +20 %)
	Protection against reverse polarity	Yes
120 V AC...240 V AC wide-range supply		
	Voltage	120...240 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
Allowed interruptions of power supply		
	DC supply	Interruption < 10 ms, time between 2 interruptions > 1 s, PS2

NOTICE!
Exceeding the maximum power supply voltage for process or supply voltages could lead to unrecoverable damage of the system. The system might be destroyed.

NOTICE!
For the supply of the modules, power supply units according to PELV or SELV specifications must be used.

The creepage distances and clearances meet the requirements of the overvoltage category II, pollution degree 2.

Parameter		Value
Temperature		
	Operating	-40 °C...+70 °C -40 °C...-30 °C: Proper start-up of system; technical data not guaranteed -40 °C...0 °C: Due to the LCD technology, the display might respond very slowly. -40 °C...+40 °C: Vertical mounting of modules possible, output load limited to 50 % per group +60 °C...+70 °C with the following deratings: ● System is limited to max. 2 communication modules per terminal base ● Applications certified for cULus up to +60 °C ● Digital inputs: maximum number of simultaneously switched on input channels limited to 75 % per group (e.g. 8 channels => 6 channels) ● Digital outputs: output current maximum value (all channels together) limited to 75 % per group (e.g. 8 A => 6 A) ● Analog outputs only if configured as voltage output: maximum total output current per group is limited to 75 % (e.g. 40 mA => 30 mA) ● Analog outputs only if configured as current output: maximum number of simultaneously used output channels limited to 75 % per group (e.g. 4 channels => 3 channels)
	Storage / Transport	-40 °C...+85 °C
Humidity		Operating / Storage: 100 % r. H. with condensation
Air pressure		Operating: -1000 m....4000 m (1080 hPa...620 hPa) > 2000 m (< 795 hPa): ● max. operating temperature must be reduced by 10 K (e.g. 70 °C to 60°C) ● I/O module relay contacts must be operated with 24 V nominal only
Immunity to corrosive gases		Operating: Yes, according to: ISA S71.04.1985 Harsh group A, G3/GX IEC 60721-3-3 3C2 / 3C3
Immunity to salt mist		Operating: Yes, horizontal mounting only, according to IEC 60068-2-52 severity level: 1

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.3.8.3.4 “TA535 - Protective caps for XC devices” on page 3333*

Table 615: Electromagnetic compatibility

Parameter		Value
Device suitable for:		
	Industrial applications	Yes
	Domestic applications	No
Radiated emission (radio disturbances)		Yes, according to: CISPR 16-2-3
Conducted emission (radio disturbances)		Yes, according to: CISPR 16-2-1, CISPR 16-1-2
Electrostatic discharge (ESD)		Yes, according to: IEC 61000-4-2, zone B, criterion B
Fast transient interference voltages (burst)		Yes, according to: IEC 61000-4-4, zone B, criterion B
High energy transient interference voltages (surge)		Yes, according to: IEC 61000-4-5, zone B, criterion B
Influence of radiated disturbances		Yes, according to: IEC 61000-4-3, zone B, criterion A
Influence of line-conducted interferences		Yes, according to: IEC 61000-4-6, zone B, criterion A
Influence of power frequency magnetic fields		Yes, according to: IEC 61000-4-8, zone B, criterion A

In order to prevent malfunctions, it is recommended, that the operating personnel discharge themselves prior to touching communication connectors or perform other suitable measures to reduce effects of electrostatic discharges.

NOTICE!

Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↳ *Chapter 1.6.4.6.5.6 “TA524 - Dummy communication module” on page 3446*.
– I/O bus connectors must not be touched during operation.

Mechanical data

Parameter	Value
Wiring method	Spring terminals
Degree of protection	IP 20
Vibration resistance	Yes, according to: IEC 61131-2 IEC 60068-2-6 IEC 60068-2-64
Shock resistance	Yes, according to: IEC 60068-2-27
Assembly position	Horizontal Vertical (no application in salt mist environment)
Assembly on DIN rail	
DIN rail type	According to IEC 60715 35 mm, depth 7.5 mm or 15 mm
Assembly with screws	
Screw diameter	4 mm
Fastening torque	1.2 Nm

Environmental tests

Parameter	Value
Storage	IEC 60068-2-1 Test Ab: cold withstand test -40 °C / 16 h IEC 60068-2-2 Test Bb: dry heat withstand test +85 °C / 16 h
Humidity	IEC 60068-2-30 Test Db: Cyclic (12 h / 12 h) damp-heat test 55 °C, 93 % r. H. / 25 °C, 95 % r. H., 6 cycles IEC 60068-2-78, stationary humidity test: 40 °C, 93 % r. H., 240 h
Insulation Test	IEC 61131-2

Parameter	Value
Vibration resistance	IEC 61131-2 / IEC 60068-26: 5 Hz...500 Hz, 2 g (with memory card inserted) IEC 60068-2-64: 5 Hz...500 Hz, 4 g rms
Shock resistance	IEC 60068-2-27: all 3 axes 15 g, 11 ms, half-sinusoidal

Table 616: EMC immunity

Parameter	Value
Electrostatic discharge (ESD)	Electrostatic voltage in case of air discharge: 8 kV Electrostatic voltage in case of contact discharge: 6 kV
Fast transient interference voltages (burst)	Supply voltage units (DC): 4 kV Digital inputs/outputs (24 V DC): 2 kV Analog inputs/outputs: 2 kV Communication lines shielded: 2 kV I/O supply (DC-out): 2 kV
High energy transient interference voltages (surge)	Supply voltage units (DC): 1 kV CM *) / 0.5 kV DM *) Digital inputs/outputs (24 V DC): 1 kV CM *) / 0.5 kV DM *) Digital inputs/outputs (AC): 4 kV Analog inputs/outputs: 1 kV CM *) / 0.5 kV DM *) Communication lines shielded: 1 kV CM)* I/O supply (DC-out): 0,5 kV CM *) / 0.5 kV DM *)
Influence of radiated disturbances	Test field strength: 10 V/m
Influence of line-conducted interferences	Test voltage: 10 V
Power frequency magnetic fields	30 A/m 50 Hz 30 A/m 60 Hz

*) CM = Common Mode, * DM = Differential Mode

1.6.4.8 AC500-S

The AC500-S safety user manual must be read and understood before using safety configuration and programming tools of Automation Builder / PS501 Control Builder Plus. Only qualified personnel shall be allowed to work with AC500-S safety PLCs.

In order to have always the latest version and due to a different lifecycle compared to Automation Builder help, the *AC500-S safety user manual* is only available on our website.

The AC500-S safety PLC includes the following safety-relevant hardware components.

- SM560-S / SM560-S-FD-1 / SM560-S-FD-4
- DI581-S
- DX581-S
- AI581-S
- TU582-S

1.6.5 System technology for AC500 V3 products

This chapter provides advanced information on the system technology of AC500 control systems from a general perspective. It provides information to link the details from the hardware descriptions (provided in the device specifications section) with detailed information on configuring/programming a corresponding library (provided in the individual library sections).

Configuration of a specific device with Automation Builder is described in the PLC configuration section.

1.6.5.1 System technology of CPU and overall system

1.6.5.1.1 Handling of remanent variables for AC500 V3 products

The retain / persistent memory must be buffered by a battery TA521 for the PLCs PM56xx-2ETH. Following described functionalities are only working if a battery is inserted. Take care about the handling for TA521 battery.

Chapter 1.6.4.6.5.4 “TA521 - Battery” on page 3441

The AC500-eCo V3 PLCs, PM50xx-ETH PLCs don't need a battery.

All operands supported by CODESYS are described in *Chapter 1.4.1.7 “Configuring I/O Links” on page 213*. For the memory sizes of the different CPUs, see for AC500-eCo V3 *“AC500-eCo V3 processor modules” on page 3457* and for AC500 V3 *“AC500 V3 processor modules” on page 3457*.

This part of the documentation describes the declaration of remanent variables for AC500 V3 products.

Different handling of remanent variables in AC500 FW ≥V3.0.2

- *No more %R memory area (use instead %M with {no_init} Chapter 1.6.5.1.1.5 “Initialization of %M variables” on page 3461)*
- *Creating of addresses for "VAR RETAIN PERSISTENT" variables automatically by IEC Compiler*

It is **NOT** possible to change the structure (e.g. add, delete, change order, ..) of retain / persisten variables of a project and update the project via memory card.

Up to version of SystemFW 3.4.x the boot project will be deleted (renamed into application.err) and the PLC will not load the boot project anymore. Also download of new/other project with Automation Builder failes.

Workaround:

- Automation Builder ➔ PLC Shell ➔ clearsram all ➔ retain persistent / retain area is deleted (not the %M area) ➔ application is running after reboot with initialized / retain / persistent data
- Automation Builder ➔ PLC Shell ➔ clearsram all ➔ retain persistent / retain area is deleted ➔ sram c m (clear %M) ➔ %M area is deleted ➔ reboot ➔ application is running with initialized /retain/persistent/%M data
- Automation Builder empty project ➔ Reset Origin Device ➔ retain persistent / retain area / %M area is deleted ➔ new update via SD card ➔ application is running after reboot with initialized data
- For midrange reboot without battery ➔ application is running after reboot with initialized data

The application is running with initialized persistent and/or retain after updating the application via memory card and rebooting the PLC.

In version of SystemFW 3.5.x the changed retain persistent / retain area is deleted. Application is running after reboot with initialized / retain / persistent data.

In case of trouble use the above described workaround.

Memory sizes

AC500-eCo V3 processor modules

PLC type	system RAM disk	userdisk PlcLogic ...	Retain, ProzM area	flash disk	memory card
PM5012-x-ETH	Dynamically /max. 7.6 MB	30 MB	8 kB Retain and persistent 4 kB (of which 88 byte are reserved for allocation table) ProzM 4 kB	None	see *Chapter 1.6.4.6.5.2 “MC5102 - Micro memory card with micro memory card adapter” on page 3432*
PM5032-x-ETH			32 kB Retain and persistent 16 kB (of which 88 byte are reserved for allocation table) ProzM 16 kB		
PM5052-x-ETH					
PM5072-T-2ETH(W)			100 kB Retain and persistent 36 kB (of which 88 byte are reserved for allocation table) ProzM 64 kB		

AC500 V3 processor modules

PLC type	system RAM disk	userdisk PlcLogic ...	SRAM Retain, ProzM area	flash disk	memory card
PM5630-2ETH	Dynamically /max. 7.6 MB	40 MB 30 MB (as of V3.4.0)	256 kB Retain and persistent 128 kB (of which 24 byte are reserved for allocation table) ProzM 128 kB	None	see *Chapter 1.6.4.6.5.1 “MC502 - Memory card” on page 3428* *Chapter 1.6.4.6.5.3 “MC5141 - Memory card” on page 3437* *Chapter 1.6.4.6.5.2 “MC5102 - Micro memory card with micro memory card adapter” on page 3432*
PM5650-2ETH	Dynamically /max. 16 MB	246 MB (as of V3.0.x) 381 MB (as of V3.1) 285.75 (as of V3.4.0)			
PM5670-2ETH	Dynamically /max. 69 MB	858 MB 643.50 MB (as of V3.4.0)	1536 MB 1 MB retain and persistent (of which 24 byte are reserved for allocation table) 512 kB ProzM		
PM5675-2ETH				8 GB	

> *It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.*

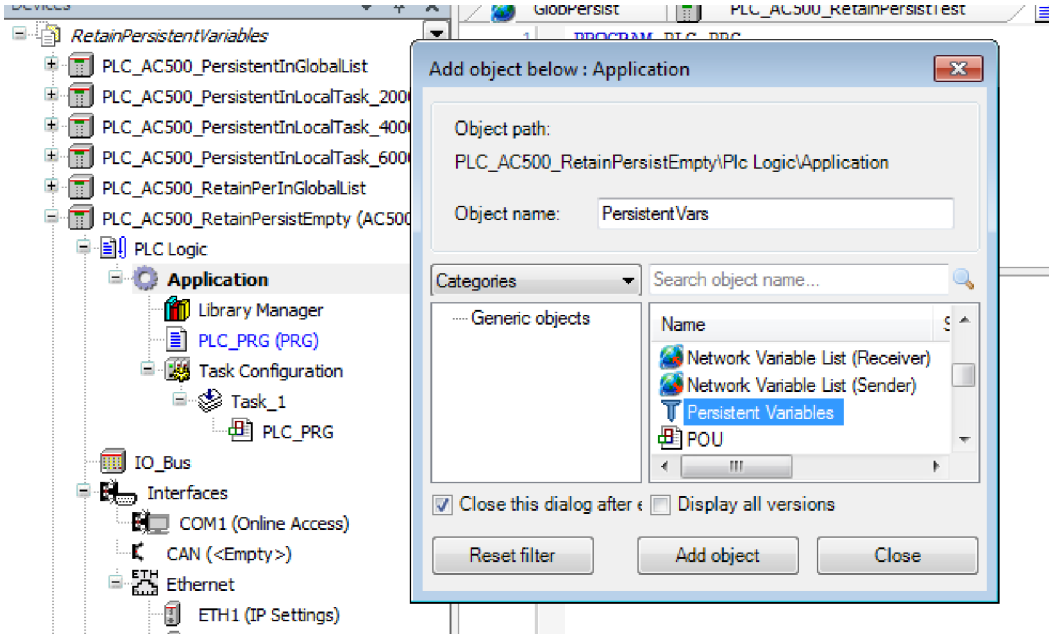
Adding a global list of persistent/retain variables

A global list of persistent variables will be added with the standard definition for persistent variables "VAR RETAIN PERSISTENT" (see Remanent variables ↳ *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537* ↳ *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*).

First steps:

1. Expand the object path of your PLC
2. After right click on *App* select *Add object* in the context menu.

   The window *Add object below: Application* appears.

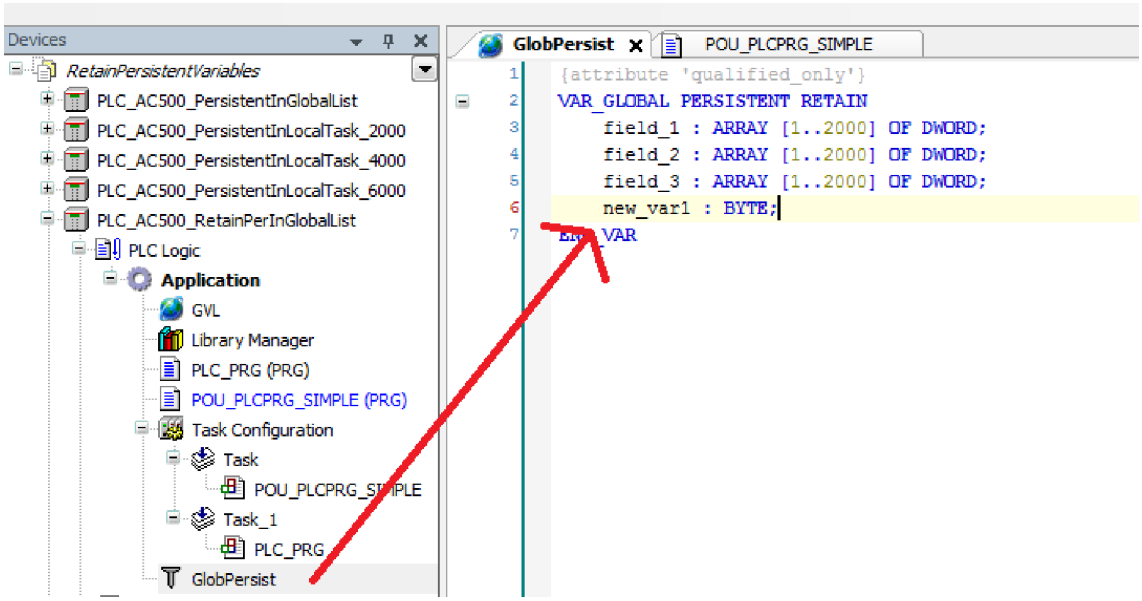


3. Select *Persistent Variables* and click *“Add object”*.

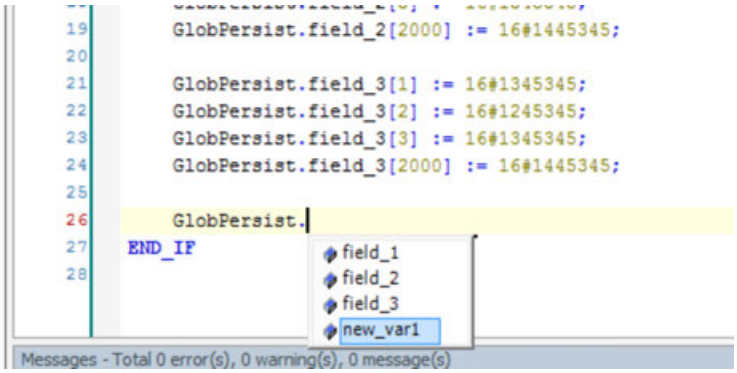
The object name can be chosen freely. In the application it will be reused to reference the persistent variables.

Declaring a new variable in global list

Declare a new variable in the window "*GlobPersist*".



Afterwards the variable can be selected in the program.



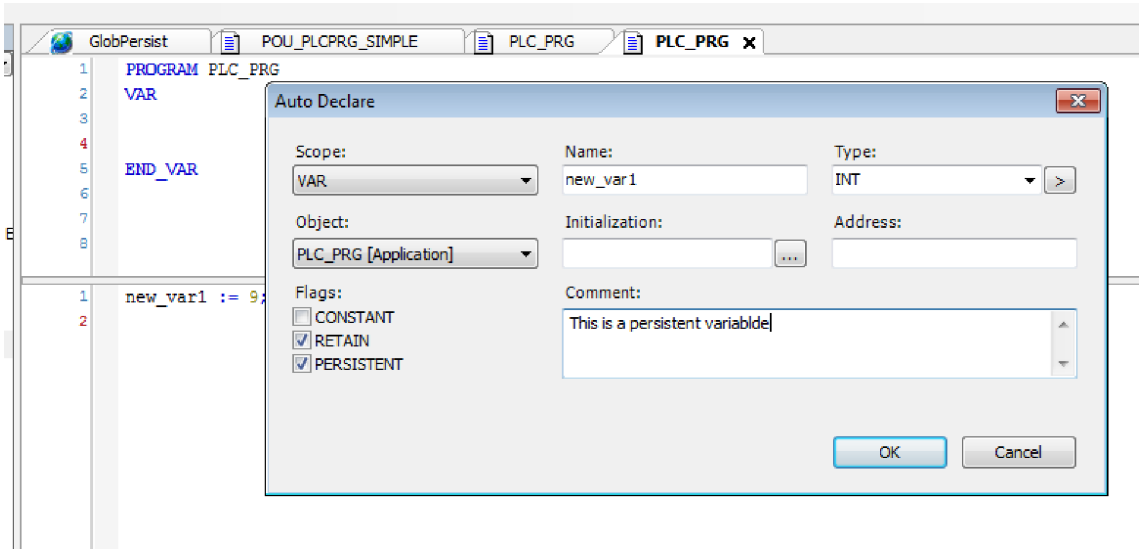
In this way the persistent variable can be accessed directly.

> *Do not use the same persistent variable in different IEC tasks, to avoid problems with consistency.*

Declaring a new persistent/retain variable in local POU

It is also possible to declare a persistent/retain variable in a local POU and not in the global list of persistent variables.

> *It is not recommended to declare a large number of persistent variables locally, due to the potentially effect to performance.*

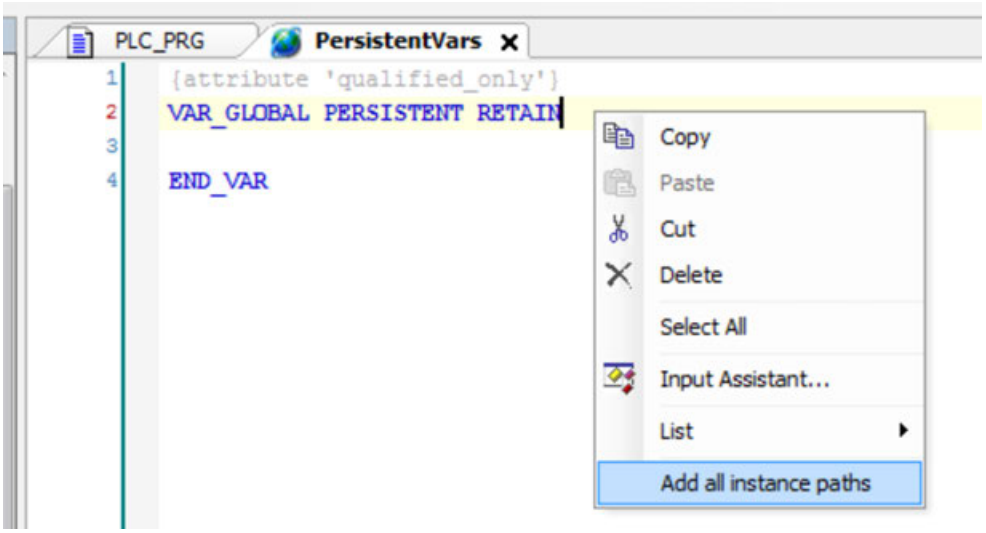


The auto-declare mechanism declares **always** a persistent variable locally and not in the global list. If the program will be executed, the following warning appears in the message window:

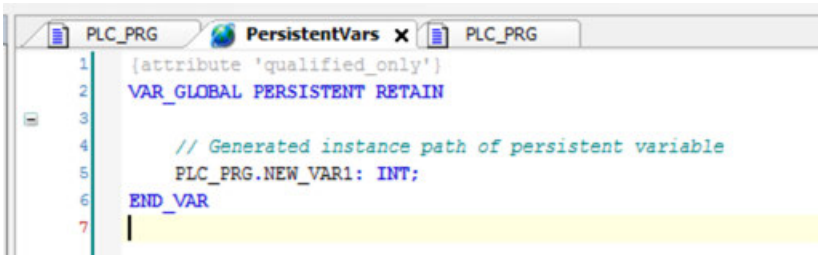
C0244: No VAR_PERSISTENT-list is part of the application to enter instance path for variable PLC_PRG.new_var1

The locally declared persistent variable has to be added to the global list.

> **NOTICE!**
> For the initialization of a Retain/Persistent variable the value of the global list is used NOT the value of the local declaration.
>
> For further information see *"RetainPersistentExample.project"*.



1. Right-click in window "*PersistentVars*".
2. Select option "*Add all instance paths*".
   ⇨ Afterwards the persistent variables are added.



The application can be downloaded to the PLC

> *It is NOT recommended to declare a new persistent variable in the application due to performance problems.*
>
> *For example PM5650-2ETH:*
>
> *1000 DWORD ≈ 600µs additional cycle time of task.*

Initialization of %M variables

After download or restart, all %M variables will be initialized to 0. This can be prevented by setting the "*no_init*" attribute.

In doing so the %M variables behave similar to the "*VAR RETAIN PERSISTENT*" variables.

```
cnt : DWORD := 0;
setvar : BOOL := FALSE;
{attribute 'no_init'}
ProzMivar AT %MW0: WORD;
ProzMivarField AT %MW1: ARRAY [1..1000] OF WORD;
ProzMivar1 AT %MW1001: WORD;
```

In the example above variable "*ProzMivar*" has the attribute "*no_init*". This variable will not be initialized and keeps its last value.

The attribute "*no_init*" is always and only valid for the next following variable (see *Chapter 1.4.1.19.6.2.30 “Attribute 'noinit'” on page 713*).

The following two variables "*Proz MivarField*" and "*Proz Mivar1*" will be further on initialized to 0.

Behavior of retain variables

The declaration of the retain variables strictly follows the 3S standard (see Remanent variables *Chapter 1.4.1.19.2.13 “Retain Variable - RETAIN” on page 537* *Chapter 1.4.1.19.2.12 “Persistent Variable - PERSISTENT” on page 535*).

For retain variables it does not matter if they are declared locally in a program or in the global variable list.

PLC shell command for import and export of retain/persistent variables

The syntax of the command is: *sram <direction><area><path>*

Supported options:	
Direction:	i=import, e=export
Area:	rp=Retain/Persistent, m=%M area
Path:	Any pathname

The file will be stored in the user partition of the PLC. This data can be imported or exported via the FTP-Server or the Files dialog in Automation Builder.

If no path is indicated, the files are saved under *"PlcLogic/<ApplicationName>/<ApplicationName>.ret or .prozm"*.

If a path is indicated, the files are saved under or accessed via *"<path>/<ApplicationName>.ret or .prozm"*.

A non-existing path is created with the exception of the memory card. The path for the memory card must be an existing path. On the memory card a non-existing path leads to an error message.

Data area	File extension	Path
Retain/Persistent	.ret	PlcLogic/<ApplicationName>/<Application-Name>.ret <path>/<ApplicationName>.ret
%M (memory area)	.prozm	PlcLogic/<ApplicationName>/<Application-Name>.prozm <path>/<ApplicationName>.prozm

Examples:

Application	Command	File
myApp	sram e rp	*PlcLogic/myApp/myApp.ret*
	sram e ep data	*data/myApp.ret*
Application	sram i m	*PlcLogic/Application/Application.prozm*
	sram i m data	*data/Application.prozm*

If the path "data" does not exist, tha path is created. The path for the memory card must be an existing path. The path "sdcard/data" leads to an error message if the path "data" does not exists on the memory card.

Only if the application uses Retain or Retain/Persistent variables the command generates an output file.

Attention!
It is recommended to execute the PLC shell command only while PLC is in state STOP, or it is ensured that there is no write access to the %M or the Retain/Persistent area.

Import and export of retain/persistent variables by library functions

It is also possible to import or export the Retain/Persistent variables and the %M markers via system function calls from the PLC Application. The required system functions are implemented in the IEC library *ABB_IntUtils_AC500.library*.

It provides the following Functions or function blocks:

- SRAM_IMPORT
- SRAM_EXPORT
- SRAM_CLEARED

SRAM_IMPORT

The function block *SRAM_IMPORT* is used to import the %M markers and the Retain/Persistent variables from the specified files in the userdisk.

Import only those %M markers and/or Retain/Persistent variables that are compatible to the application running in the PLC.

It is recommended to import only when the %M and/or the Retain/Persistent area is not accessed by the application.

Otherwise inconsistencies are possible.

For a complete description of the function block see *ABB_IntUtils_AC500.library*.

SRAM_EXPORT

The function block *SRAM_EXPORT* is used to export the %M markers and the Retain/Persistent variables to the specified files in the userdisk.

Export only those %M markers and/or Retain/Persistent variables that are compatible to the application running in the PLC.

It is recommended to export only when the %M and/or the Retain/Persistent area is not accessed by the application.

Otherwise inconsistencies are possible.

For a complete description of the function block see *ABB_IntUtils_AC500.library*.

SRAM_CLEARED

The Function SRAM_CLEARED is used to check if the *SRAM* was deleted.

For a complete description of the Function see *ABB_IntUtils_AC500.library*.

1.6.5.1.2 System processing

System start-up / Program processing

AC500-eCo processor modules do not have an integrated display and keyboard. All functions related to keyboard and display are not applied for those devices.

Definitions: PLC system start-up

Cold start

The AC500-eCo V3 does not use a battery for buffering the operand areas specified below, hence the “cold start” mode does not exist in this product.

- A cold start is performed by switching power OFF/ON if no battery is connected.
- All RAM memory modules are checked and erased (see *Chapter 1.4.1.20.3.6.10 “Command 'Reset Cold'” on page 1038*).
- If no user program is stored in the Flash EPROM, the default values (as set on delivery) are applied to the interfaces.
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

Warm start

- A warm start is performed by switching power OFF/ON with a battery connected.
- All RAM memory modules are checked and erased except of the buffered operand areas and the RETAIN variables (see *Chapter 1.4.1.20.3.6.11 “Command 'Reset Warm'” on page 1038*).
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

RUN -> STOP

- RUN -> STOP means pressing the RUN function key on the PLC while the PLC is in run mode (AC500 PLC display "run", AC500-eCo PLC "RUN LED" is ON).
- If a user program is loaded into RAM, execution is stopped.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP", AC500-eCo "RUN LED" changes from ON to OFF.

START -> STOP

- START -> STOP means stopping the execution of the user program in the PLC's RAM using the menu item "Online/Stop" in the programming system.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP".

Reset

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (exception: RETAIN variables) are set to their initialization values.
- Reset is performed using the menu item "Online/Reset" in the programming system or pressing the function key RUN for $\geq$ 5 s in STOP mode.

Reset (cold)

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (also RETAIN variables) are set to their initialization values.
- Reset (cold) is performed using the menu item "Online/Reset (cold)" in the programming system.

Reset (original)

- Resets the controller to its original state (deletion of Flash, SRAM (%M, area, %R area, RETAIN, RETAIN PERSISTENT), Communication Module configurations and user program!).
- Reset (original) is performed using the menu item "Online/Reset (original)" in the programming system.

STOP -> RUN

- STOP -> RUN means short pressing the RUN function key on the PLC while the PLC is in STOP mode (AC500 PLC display "StoP", AC500-eCo "RUN LED" is ON). "RUN LED" is OFF of the toggle switch of an AC500-eCo CPU.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo "RUN LED" changes from OFF to ON.

STOP -> START

- STOP -> START means continuing the execution of the user program in the PLC's RAM using the menu item "Online/Start" in the programming system.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo PLC "RUN LED" changes from OFF to ON.

Download

- Download means loading the complete user program into the PLC's RAM. This process is started by selecting the menu item "Online/Download" in the programming system or after confirming a corresponding system message when switching to online mode (menu item "Online/Login").
- Execution of the user program is stopped.
- In order to store the user program to the Flash memory, the menu item "Online/Create boot project" must be called after downloading the program.
- Variables are set to their initialization values according to the initialization table.
- RETAIN variables can have wrong values as they can be allocated to other memory addresses in the new project!
- A download is forced by the following:
  - changed PLC configuration
  - changed task configuration
  - changed library management
  - changed compile-specific settings (segment sizes)
  - execution of the commands "Project/Clean all" and "Project/Rebuild All".

Online change

- After a project has changed, only these changes are compiled when pressing the key <F11> or calling the menu item "Project/Build". The changed program parts are marked with a blue arrow in the block list.
- The term Online Change means loading the changes made in the user program into the PLC's RAM using the programming system (after confirming a corresponding system message when switching to online mode, menu item "Online/Login").
- Execution of the user program is not stopped. After downloading the program changes, the program is re-organized. During re-organization, no further online change command is allowed. The storage of the user program to the Flash memory using the command "Online/Create boot project" cannot be initiated until re-organization is completed.
- Online Change is not possible after:
  - changes in the PLC configuration
  - changes in the task configuration
  - changes in the library management
  - changed compile-specific settings (segment sizes)
  - performing the commands "Project/Clean all" and "Project/Rebuild All".

Data buffering

- Data buffering, i.e., maintaining data after power ON/OFF, is only possible, if a battery is connected for AC500 CPU and the buffering will take place in FLASH with AC500-eCo V3 CPU. The following data can be buffered completely or in parts:
  - Data in the addressable flag area (%M area)
  - RETAIN variable
  - PERSISTENT variable (number is limited, no structured variables)
  - PERSISTENT area (%R area)
- In order to buffer particular data, the data must be excluded from the initialization process (see *Chapter 1.6.5.1.1 “Handling of remanent variables for AC500 V3 products” on page 3456*).

Start of the user program

The user program (UP) is started according to the following table. It is assumed that a valid user program is stored to the Flash memory.

See *Chapter 1.6.7.1.4 “Storage device details” on page 3997*.

Action	No memory card with UP installed Auto run = ON	No memory card with UP installed Auto run = OFF	Memory card with UP installed Auto run = ON	Memory card with UP installed Auto run = OFF
Voltage ON or Warm start or Cold start	UP is loaded from Flash into RAM and started from Flash.	No UP is loaded from Flash. When logging in, the message "No program available in the controller ..." is displayed.	UP is loaded from the memory card into Flash memory and RAM and then started from RAM.	UP is loaded from the memory card to the Flash memory. RAM remains empty. When logging in, the message "No program available in the controller ..." is displayed.
STOP -> RUN	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.
STOP -> START	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.
Download [1)]	The UP currently stored in the CPU's RAM is stopped. The built UP is loaded from the PC into the PLC's RAM.	The built UP is loaded from the PC into the PLC's RAM.	The UP currently stored in the CPU's RAM is stopped. The built UP is loaded from the PC into the PLC's RAM.	The built UP is loaded from the PC into the PLC's RAM.
Online Change [2)]	Processing of the UP currently stored in the CPU's RAM is continued. The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized and processed.	The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized.	Processing of the UP currently stored in the CPU's RAM is continued. The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized and processed.	The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized.

Remarks:

[1)]: After the download is completed, the program is not automatically stored to the Flash memory. To perform this, create a boot project *Chapter 1.4.1.10.6 “Generating boot applications” on page 391*. If the UP is not stored to the Flash memory, the UP is reloaded from the Flash memory after voltage OFF/ON. Start the program either by pressing the RUN/STOP function key or using Automation Builder.

[2)]: After the online change process is completed, the program is not automatically stored to the Flash memory. For this, after reorganization is completed create a boot project. During reorganization and flashing, no further online change command is allowed. If the UP is not stored to the Flash memory, the UP is reloaded from the Flash memory after voltage OFF/ON.

[2)]: After the online change process is completed, the program is not automatically stored to the Flash memory. For this, after reorganization is completed create a boot project *Chapter 1.4.1.10.6 “Generating boot applications” on page 391*. During reorganization and flashing, no further online change command is allowed. If the UP is not stored to the Flash memory, the UP is reloaded from the Flash memory after voltage OFF/ON.

Task configuration

This statement is applicable to PM5032-x-ETH, PM5052-x-ETH and PM5072-T-2ETH(W).

If the main task cycle is faster than 10 ms, remove the onboard inputs I8..I11 and I/O channels of option boards or option boards for serial communication from the main task cycle, but use a separate task cycle.

The task model processes the following kind of tasks:

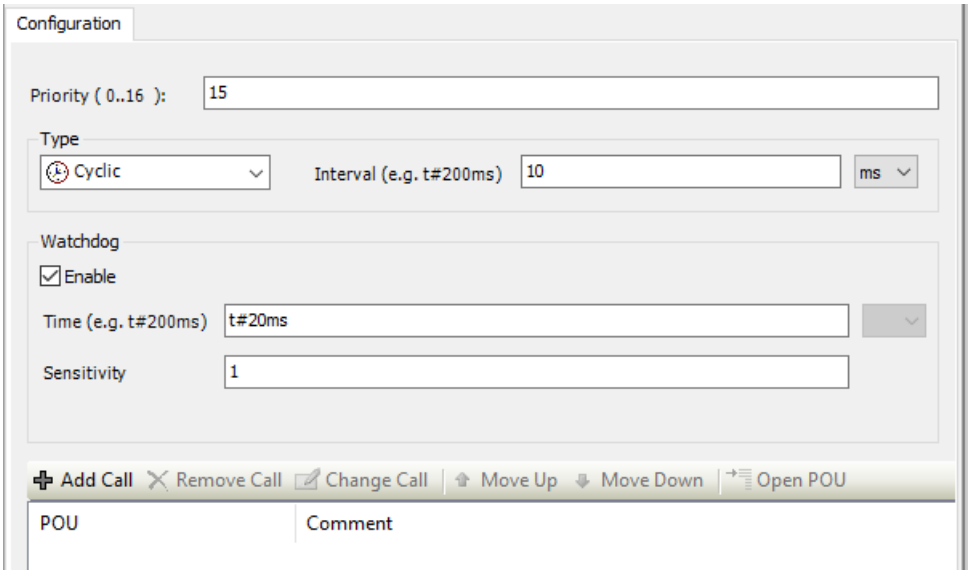
- Non-real-time system tasks: system tasks with no real-time property (e.g. file access, Ethernet communication, OPC UA, …)
- Non-real-time IEC tasks: IEC tasks with no real-time property
- Real-time system tasks: system tasks with real-time property
- Real-time IEC tasks: IEC tasks with real-time property

The possible number of tasks depends on the type of processor module. How to distribute the IEC tasks over multiple CPU cores and on how to use the IEC task configuration for Automation Builder is described in detail in the CODESYS task configuration section.

- Task configuration *Chapter 1.4.1.8.16 “Task Configuration” on page 292*
- Tab 'Configuration' *Chapter 1.4.1.20.2.27.1 “Tab 'Configuration'” on page 942*

Watchdog handling in IEC tasks

If a new project is created or a new task is inserted in the task configuration (*Chapter 1.4.1.8.16 “Task Configuration” on page 292*), the task is created with the “default task settings” priority = 15 and cycle time = 10 ms. The watchdog is activated, set to 20 ms and sensitivity = 1.



The watchdog handling depends also on the setting of the CPU parameter “Missed cycle behavior”:

Parameter	Type	Value	Default Value	Unit	Descripti
Error LED	Enumeration of BYTE	On	On		Error LED:
Check battery	Enumeration of BYTE	On	On		Check bat
Stop on error class	Enumeration of BYTE	Diagnosis of at least error class 2	Diagnosis of at least error class 2		Prevents
Diagnosis - Add PLC name to node name	Enumeration of BYTE	Off	Off		Diagnosis
PLC behaviour after voltage dip	Enumeration of BYTE	Halt	Halt		Behaviour
Diagnosis history	Enumeration of BYTE	On	On		Enables /
Max. diagnosis history entries	WORD(0..50000)	1000	1000		Max. num
Missed cycle behavior	Enumeration of BYTE	Next	Next		Behavior
Communicaton Schema	Enumeration of BYTE	Default	Default		Communic

This parameter configures the behavior of a real-time task if the processing time of the task is longer than the cycle time.

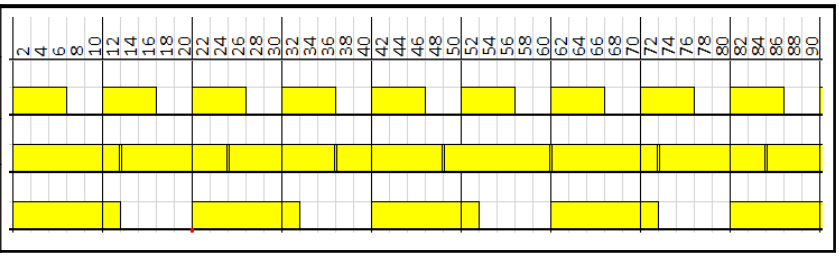
“Next” means – skip the missed cycle and start the task on the next cycle on time. This might result in skipped tasks, but at least the highest priority task is always started on time, if it is not skipped (= default value).

“ASAP” means - start the task immediately when possible.

> *This parameter is valid for all real-time tasks (priorities 0-15) of the PLC application.*

Example 1: default task settings, t_{Task} – processing time of the task in [ms]

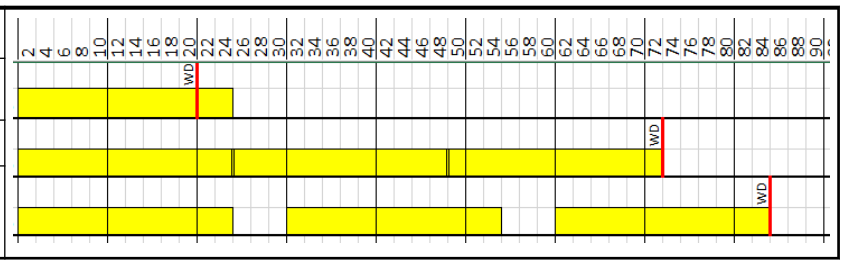
t_{Task}	Par.
6	Next
12	ASAP
12	Next



No watchdog occurs, also if the processing time of the task is longer than the cycle time (cases 2 and 3) since the processing time is shorter than the watchdog time.

Example 2: default task settings, t_{Task} – processing time of the task 24 ms, S_{WD} – sensitivity of the watchdog

S_{WD}	Par.
1	Next
3	ASAP
3	Next



Watchdog occurs in all 3 cases since the processing time of the task is longer than the watchdog time. According to the setting of the sensitivity the watchdog occurs after 1 or 3 cycles.

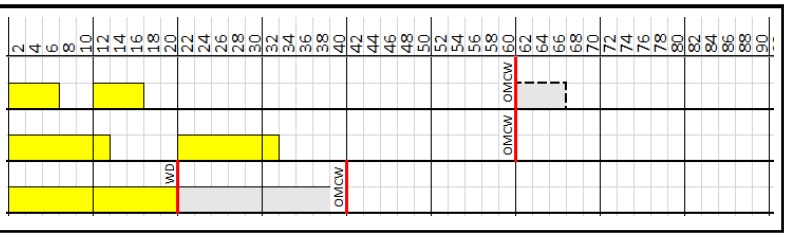
Beside the task watchdog there is the so-called “omitted cycle watchdog” (OMCW). The omitted cycle watchdog is only active if a watchdog has been configured for the task.

The "normal" Watchdog triggers only if the processing time of the task exceeds the set Watchdog value.

The omitted cycle watchdog on the other hand checks completely "failed" cycles. E.g. if the scheduler has a problem and the task never executes its cycle again, then the "normal" watchdog will not be triggered. Therefore, the run time does an additional check, if a task has been executed within the double cycle time or the double watchdog time (the bigger of both is valid). If not, the omitted cycle watchdog is triggered.

Example 3: default task settings, t_{Task} - processing time of the task in ms, C_{lost} - lost cycles

t_{Task}	C_{lost}
6	3-6
12	3-6
∞	2ff

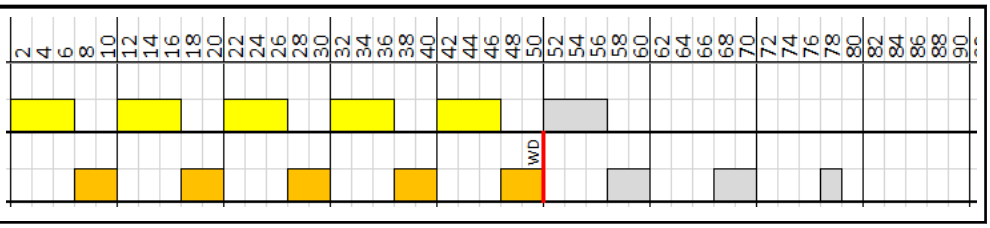


Omitted cycle watchdog occurs after double watchdog time (2 x 20 ms = 40 ms).

Example 4: Two tasks with following settings:

Task	Priority	Cycle time [ms]	Watchdog time [ms]	Sensitivity	Parameter	Task processing time [ms]
1	10	10	20	1	Next	6
2	15	50	50	1		30

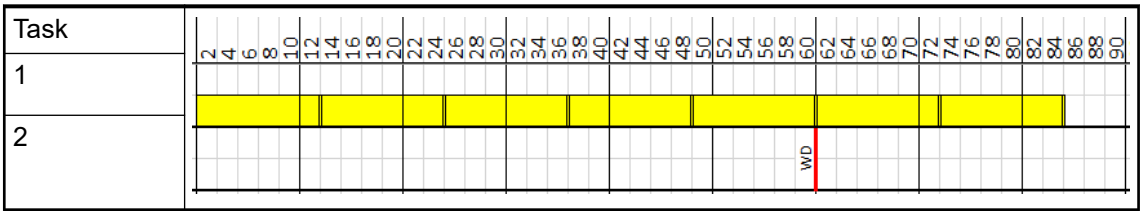
Task
1
2



Watchdog of task 2 is triggered since the task cannot run in the defined task cycle.

Example 5: Two tasks with following settings

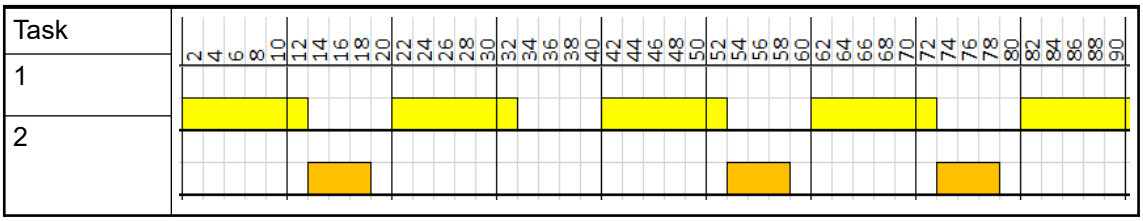
Task	Priority	Cycle time [ms]	Watchdog time [ms]	Sensitivity	Parameter	Task processing time [ms]
1	10	10	20	1	ASAP	12
2	15	30	60	1		6



Watchdog of task 2 is triggered since the task cannot start in the defined task cycle.

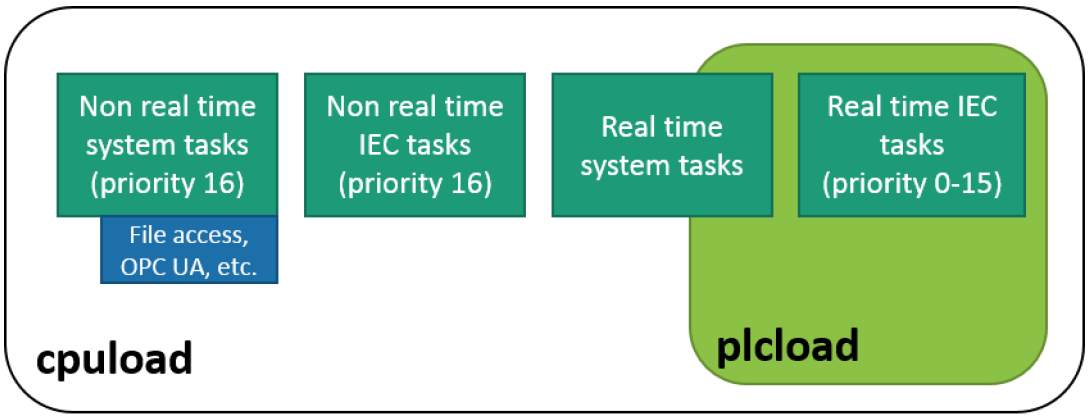
Example 6: Two tasks with following settings

Task	Priority	Cycle time [ms]	Watchdog time [ms]	Sensitivity	Parameter	Task processing time [ms]
1	10	10	20	1	Next	12
2	15	30	60	1		6



No watchdog is triggered, but task 1 is running in 20 ms cycle instead of configured 10 ms cycle. Task 2 is running alternating every 20 ms or 40 ms.

PLC utilization



The parameters `cpuload` and `plcload` represent the actual CPU load or PLC load of the system.

- `cpuload`: This value represents the time the PLC requires to calculate all processes running on the PLC. For a good system performance this value should be less than 80%. In case of a higher value, the degree of utilization should be reduced by using a more powerful PLC or by reducing the amount of processes.
- `plcload`: This value represents the time the PLC requires to calculate all real-time processes. Real-time processes are either high priority system tasks or IEC tasks with a priority between 0 and 15. For a good system performance this value should be less than 60%. In case of a higher value, the degree of utilization should be reduced by using a more powerful PLC.

During commissioning we recommend to monitor the CPU and PLC values online with one of the following methods:

Automation Builder

- Commissioning via *Chapter 1.6.6.4.4 “PLC shell commands” on page 3950* (command 'plcload' and 'cpuload').
- Commissioning via *Chapter 1.4.1.12.3 “Data Recording with Trace” on page 421*. In order to display the load of the CPU or PLC, create a new Device Trace object in your PLC project. Then upload the data into the views *Chapter 1.4.1.20.3.21.19 “ Command 'Upload Trace'” on page 1146*. .

IEC applications/IEC program

To access the parameters `plcload` and `cpuload` please use system functions as follows:

- `plcload`: SchedGetProcessorLoad() included in library ‘CmpSchedule’.
- `cpuload`: SysMCGetLoad() included in library ‘SysCpuMultiCore’.

Managing priorities by selecting the appropriate communication schema

The AC500 V3 PLCs have an integrated preemptive real-time operating system that supports 100 priorities from 0 (lowest priority) to 99 (highest priority). Hereof 0 ... 49 in the non-real-time area and 50...99 in the real-time area.

For real-time tasks in the IEC user program 16 priorities from 0 (highest priority) to 15 (lowest priority) can be used. They correspond to the operating system priorities 67 (for IEC task priority 0) to 52 (for IEC task priority 15).

The file system, the memory card and flash tasks run on lowest real-time priority 50.

The non-real-time IEC task priority 16 runs in the non-real-time area. Likewise, all Ethernet protocols and the diagnosis system run in the non-real-time area.

As of Automation Builder 2.4.1 and *“SystemFW”* 3.4.1 provides a new PLC boot parameter `Communication Schema` (non-real time vs. real-time Ethernet data) for AC500 V3:

- Name: “Default”
  Description: Balanced priority for communication via communication modules (CMs) and onboard Ethernet communication.
  Further information on page 3472
- Name: “Communication modules”
  Description: Priority and high performance for communication module (CM) based communication via sync tasks. Lower priority for onboard Ethernet and local I/O bus.
  Further information on page 3473

- Name: “Onboard Ethernet”
  Description: Priority for onboard Ethernet communication (e. g. via Modbus TCP). Lower priority for communication via communication modules (CMs)
  Further information on page 3473
- Name: “Realtime onboard Ethernet”
  Description: Very high priority for onboard Ethernet communication (e. g. EtherCAT, PROFINET, Ethernet/IP). Low priority for communication via communication modules (CMs)
  Further information on page 3473

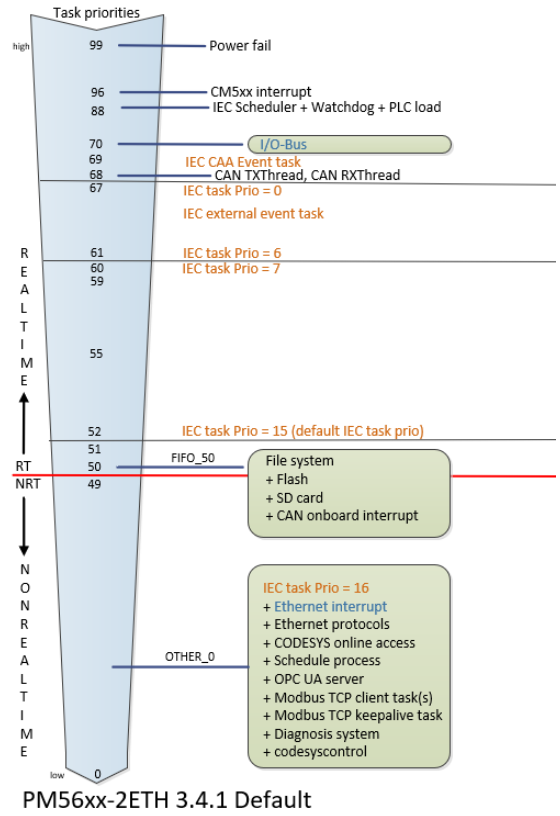
> *The value “Realtime onboard Ethernet” is reserved for later use and has currently the same settings as “Onboard Ethernet” and in addition the I/O bus on the same priority as the Ethernet.*

In version of *“SystemFW”* 3.5.0 the priority of onboard CAN interface has been adapted and is now included in the priority schemas.

In addition the parameter `Communication Schema` is now also available for the eCo-V3 PLCs.

The „Default“ priority schema in *“SystemFW”* 3.4.1

The default value of the boot-parameter `Communication Schema` is the balanced priority for communication via communication modules (CMs) and onboard Ethernet communication. The following figure gives an overview about the main task priorities in the AC500 V3 PLCs.



The highest IEC task priority 0 should be used for high prior functions in the PLC with short execution time.

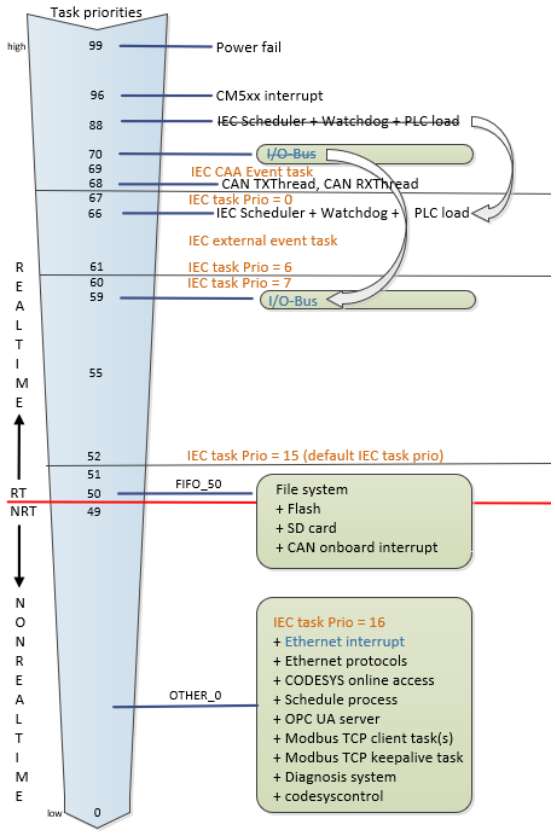
The communication modules interrupts (CM5xx), the local I/O bus, the IEC scheduler observer task and the CAA event task are processed with higher priority than all user IEC tasks.

Ethernet runs on non-real-time priority.

The default setting of the priorities is suitable for most applications and corresponds to the settings in the firmware versions 3.4.0 and before.

The “Communication modules” priority schema in *“SystemFW”* 3.4.1

The communication modules priority schema has been established for priority and high performance for communication module (CM) based communication via sync tasks. Lower priority for onboard Ethernet and local I/O bus.



PM56xx-2ETH 3.4.1 Communication modules

The highest IEC task priority 0 should be used for the sync task of the highest priority communication module, e. g. CM579-ETHCAT, priority 1 for the sync task with second highest priority, and so on.

Only the communication modules interrupts (CM5xx) and the CAA event task is processed with higher priority than the IEC tasks.

The IEC scheduler observer task has been moved to priority 66, means below IEC task priority 0 and on the same level as IEC task priority 1.

The priority of the local I/O bus has been moved to priority 59, means inside the IEC task priority area, but below the external event task priorities.

Ethernet runs on non-real-time priority.

The priority schema communication module should be used in applications with one or more communication modules CM5xx with sync mode. As of Automation Builder 2.4.1 these are the CM579-ETHCAT EtherCAT master and CM598-CN CANopen master communication modules.

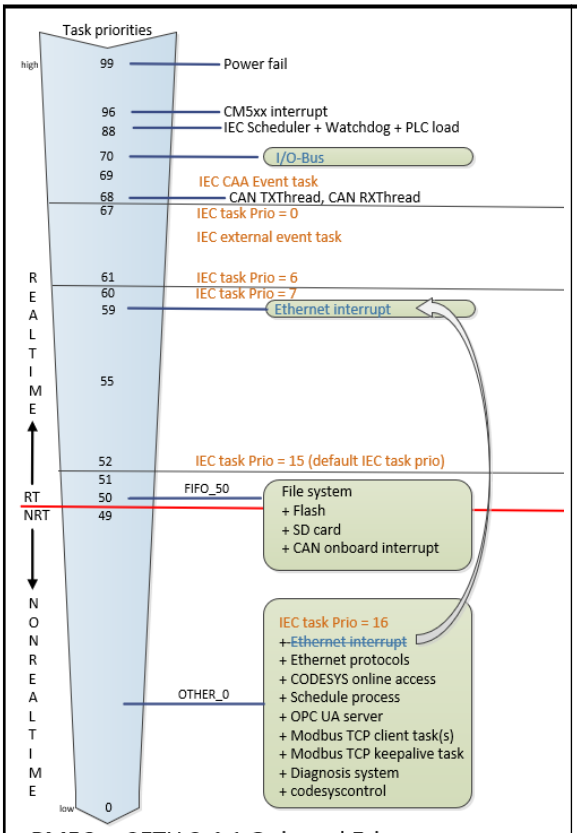
The sync task with priority 0 will be interrupted only by system interrupts. Since the IEC scheduler observer task is located below IEC priority 0, the watchdog for this task is also ineffective. However, this should not interfere with a sync task.

If more than one CM5xx are used in sync mode, the priority order must be defined. The sync task of highest priority CM5xx receives IEC priority 0, the next priority 1 and so on.

In a mixed PLC configuration with communication modules with and without sync mode the interrupts of the communication modules without sync mode will be handled on the priority of the lowest sync task. Currently supported communication modules without sync mode are CM579-PNIO PROFINET IO controller and SM560-S Safety PLC.

The “Onboard Ethernet” priority schema in *“SystemFW”* 3.4.1

The “Onboard Ethernet” priority schema has been established for priority for onboard Ethernet communication (e. g. via Modbus TCP).



PM56xx-2ETH 3.4.1 Onboard Ethernet

The Ethernet interrupt task has been moved from non-real-time area to priority 59, means inside the IEC task priority (real-time) area, but below the external event task priorities.

This priority schema should be used for applications with much Ethernet communication, e. g. Modbus TCP communication with a high number of Modbus TCP clients/servers.

NOTICE!

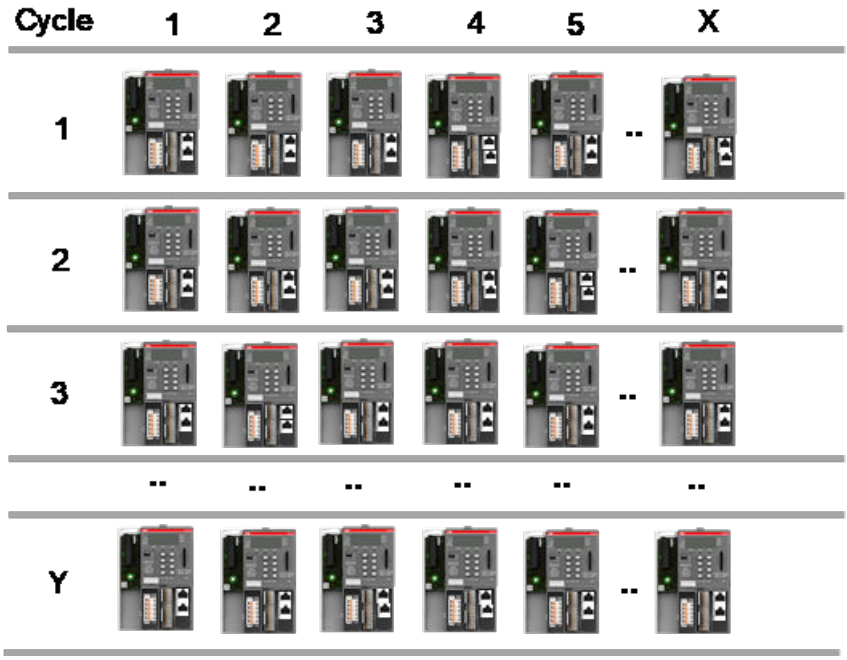
Since the Ethernet interrupt task is running in this mode in the real-time priority area, the Ethernet communication can block IEC tasks with priorities 12-15.

Working with real-time priority at onboard Ethernet and using a high number of Modbus TCP client connections can force high CPU load. To avoid this, it is recommended to call the Modbus function blocks in steps.

Example

100 Modbus TCP client connections shall be used in a 20 ms task.

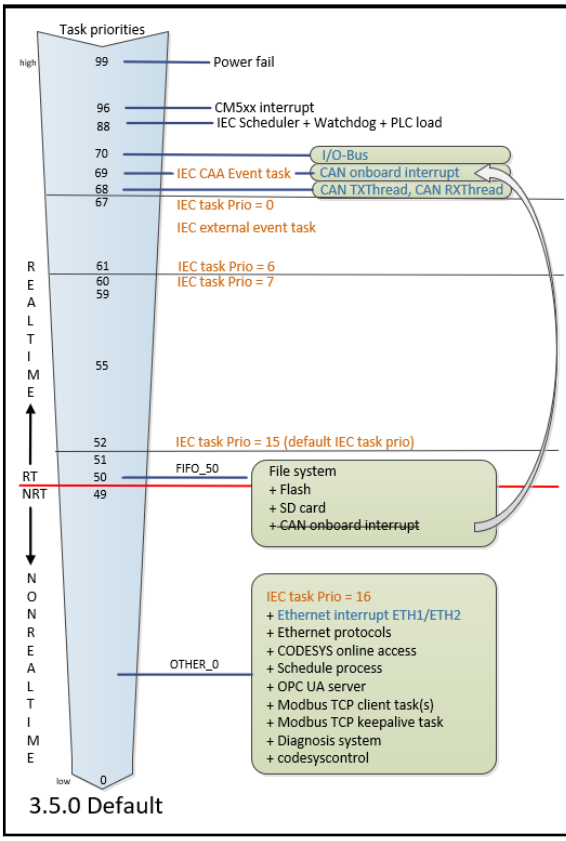
Call 20 function blocks in a first cycle, 20 function blocks in a second cycle and so on.



The „Default“ priority schema in *“SystemFW”* 3.5.0

The default value of the boot-parameter `Communication Schema` is the balanced priority for communication via communication modules (CMs) and onboard Ethernet communication.

The following figure gives an overview about the main task priorities in the AC500 V3 midrange PLCs in version *“SystemFW”* 3.5.0.



The highest IEC task priority 0 should be used for high prior functions in the PLC with short execution time.

The communication modules interrupts (CM5xx), the local I/O bus, the IEC scheduler observer task and the CAA event task are processed with higher priority than all user IEC tasks.

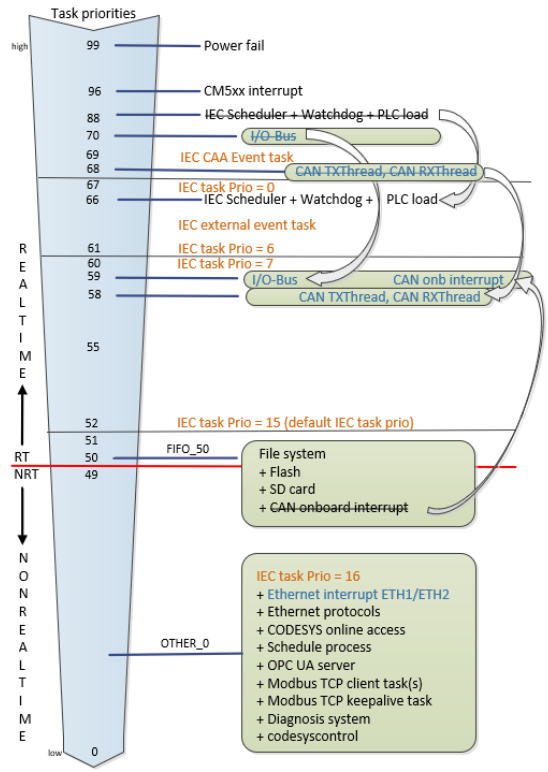
Ethernet runs on non-real-time priority.

The CAN onboard interrupt is moved from non-real-time priority to real-time priority 69, one priority above the CAN transmit and receive threads.

The default setting of the priorities is suitable for most applications and corresponds to the settings in the firmware versions 3.4.0 and before.

The “Communication modules” priority schema in *“SystemFW”* 3.5.0

The communication modules priority schema has been established for priority and high performance for communication module (CM) based communication via sync tasks. Lower priority for onboard Ethernet and local I/O bus.



3.5.0 Communication modules

The highest IEC task priority 0 should be used for the sync task of the highest priority communication module, e. g. CM579-ETHCAT, priority 1 for the sync task with second highest priority, and so on.

Only the communication modules interrupts (CM5xx) and the CAA Event task is processed with higher priority than the IEC tasks.

The IEC scheduler observer task has been moved to priority 66, means below IEC task priority 0 and on the same level as IEC task priority 1.

The priority of the local I/O bus has been moved to priority 59, means inside the IEC task priority area, but below the external event task priorities.

The CAN onboard interrupt has the same priority as the I/O bus and the CAN transmit and receive threads one priority below, means 58.

Ethernet runs on non-real-time priority.

The priority schema communication module should be used in applications with one or more communication modules CM5xx with sync mode. As of Automation Builder 2.4.1 these are the CM579-ETHCAT EtherCAT master and CM598-CN CAN master communication modules.

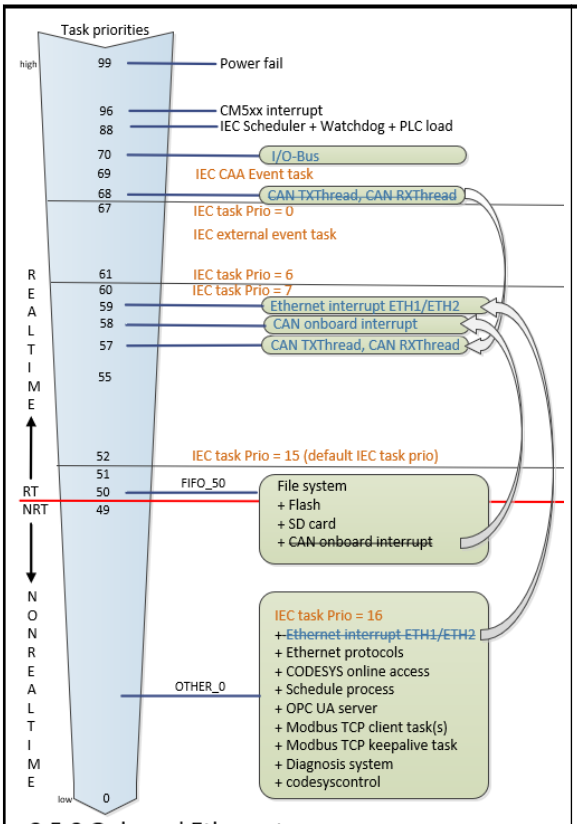
The sync task with Prio 0 will be interrupted only by system interrupts. Since the IEC scheduler observer task is located below IEC Prio 0, the watchdog for this task is also ineffective. However, this should not interfere with a sync task.

If more than one CM5xx are used in sync mode, the priority order must be defined. The sync task of highest priority CM5xx receives IEC Prio 0, the next Prio 1 and so on.

In a mixed PLC configuration with communication modules with and without sync mode the interrupts of the communication modules without sync mode will be handled on the priority of the lowest sync task. Currently supported communication modules without sync mode are CM579-PNIO PROFINET IO controller and SM560-S safety PLC.

The “Onboard Ethernet” priority schema in *“SystemFW”* 3.5.0

The “Onboard Ethernet” priority schema has been established for priority for onboard Ethernet communication (e. g. via Modbus TCP).



The Ethernet interrupt task has been moved from non-real-time area to priority 59, means inside the IEC task priority (real-time) area, but below the external event task priorities.

The CAN onboard interrupt is moved one priority below the Ethernet interrupt to priority 58 and the CAN transmit and receive threads one priority below the CAN onboard interrupt to priority 57.

This priority schema should be used for applications with much Ethernet communication, e. g. Modbus TCP communication with a high number of Modbus TCP clients/servers.

NOTICE!

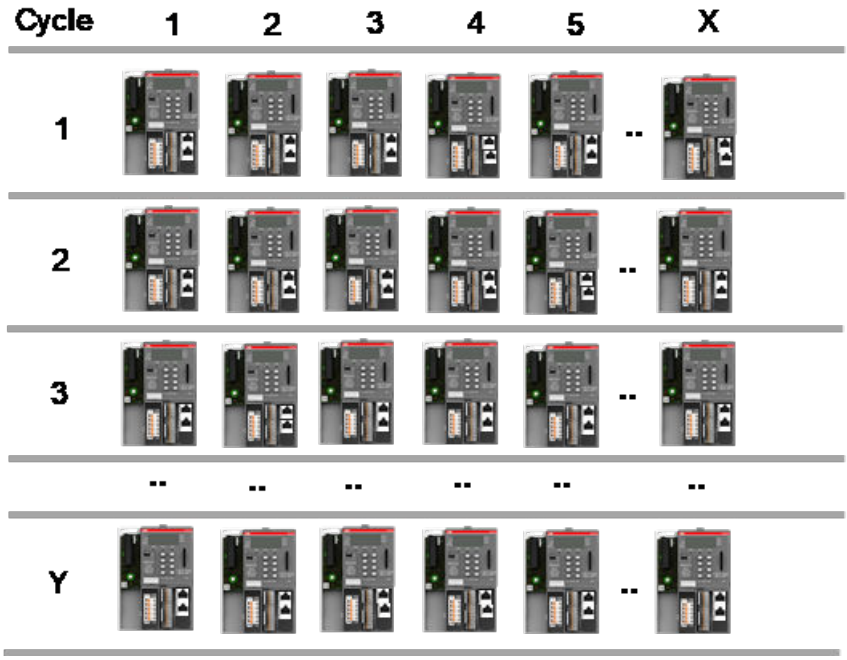
Since the Ethernet interrupt task is running in this mode in the real-time priority area, the Ethernet communication can block IEC tasks with priorities 12-15.

Working with real-time priority at Onboard Ethernet and using a high number of Modbus TCP client connections can force a high CPU Load. To avoid this, we recommend calling the Modbus FB's in steps.

Example

100 Modbus TCP client connections shall be used in a 20 ms task.

Call 20 function blocks in a first cycle, 20 function blocks in a second cycle and so on.



Setting standard configuration

If the target setting configuration is changed, standard configuration can be restored:

1. Open CODESYS.
2. In the *"Resources"* tab, double-click *"PLC Configuration"*.
3. Select *"Menu Extras* ➔ *Standard Configuration"*.

1.6.5.1.3 User Management

With the help of the integrated user management, user groups with different access rights and authorizations can be defined. Configuration and handling of the user managment in Automation Builder and a AC500 V3 is decribed in an *application note*.

1.6.5.1.4 Real-time clock and battery

Notes on real-time clock

The real-time clock is an optional function for AC500-eCo V3 Basic processor modules (e.g. PM5012-x-ETH) and requires a TA5131-RTC. All other AC500-eCo V3 processor modules have an integrated real-time clock.

The real-time clock operates as a PC clock. It saves date and time to a DWORD in DT format (DATE AND TIME FORMAT), i.e., in seconds passed since the start time: 1 January 1970 at 00:00.

For AC500-eCo V3, Basic CPU with TA5131-RTC buffers the real-time clock for 7 days, and Standard/Pro CPU buffers the integrated real-time clock for 20 days. When the CPU is not powered over the buffering time, the real-time clock data will be cleared.

If a battery is connected and full, the real-time clock continues to run even if the control voltage is switched off.

If no battery is inserted or the battery is empty, the real-time clocks starts with the value 0 (=1970-01-01, 00:00:00).

When switching on the control voltage, the system clock of the operating system is set to the value of the real-time clock.

Real-time clock

Real-time clock with PLC browser

The PLC browser/PLC shell commands `date` and `time` are used to set the real-time clock.

The commands `date` <ENTER> or `time` <ENTER> display the current date and time of the real-time clock.

The command: `date yyyy-mm-dd`<ENTER> (year-month-day) sets the date.

The command: `time hh-mm-ss`<ENTER> (hours-minutes-seconds) sets the time.

Example:

The real-time clock should be set to 22 February 2005, 16:50.

1. Enter the date:

   `date 2005-02-22<ENTER>`

   ⇨ Display: `date 2005-02-22 Clock set to 2005-02-22 08:01:07`

   The time remains unchanged.

2. Enter the time:

   `time 16:50<ENTER>`

   ⇨ Display: `time 16:50 Clock set to 2005-02-22 16:50:00`

Real-time clock with user program

The following function blocks located in the folder "Realtime clock" of the system library *ABB_ExtUtils_AC500.lib* can be used to set and display the real-time clock (RTC) with help of the user program:

Function block	Function
CLOCK (V3) *"Library Manager ➔ ABB-AC500 ➔ Use Cases ➔ AC500 Utils ➔ PM<Version> (ABB) ➔ Function Blocks ➔ Realtime clock"*	Sets and displays the real-time clock with values for year, month, day, hours, minutes and seconds. Also the day of week is indicated (Mo=1, Tue=2, Wed=3, Thu=4, Fr=5, Sa=6, Su=0). Note: The week of day cannot be set. It is given by the real-time clock. The input DAY_SET is ignored.
CLOCK_DT (V3) *"Library Manager ➔ ABB-AC500 ➔ Use Cases ➔ AC500 Utils ➔ PM<Version> (ABB) ➔ Function Blocks ➔ Realtime clock"*	Sets and displays the real-time clock in DT format, for example DT#2005-02-17-17:15:00.

Reference for function blocks, functions, structures etc. ↳ *Chapter 1.10 "Reference, function blocks" on page 4292*

AC500 battery

The AC500 battery buffers the following data in case of "control voltage off":

- Retentive variables in SRAM (VAR_RETAIN..END_VAR) ↳ *Chapter 1.6.5.1.1 "Handling of remanent variables for AC500 V3 products" on page 3456*
- Date and time of the real-time clock

Further information:

- *Chapter 1.7.3 “Diagnosis messages” on page 4062*

> *To prevent data loss when using the AC500 battery, the battery status should be periodically monitored by the user program.*

Battery status

The battery status can be monitored either with the help of a user program on the PLC or in Automation Builder.

In the PLC shell of Automation Builder the command "batt" *Chapter 1.6.6.4.4 “PLC shell commands” on page 3950* can be used. *Chapter 1.6.6.4.4 “PLC shell commands” on page 3950*. The following is output:

0	Battery empty
20	Remaining battery charge below 20 %
100	Battery charge OK

In the user program, the battery status can be checked with the function BATT which is available in the folder "Battery" of the system library *ABB_ExtUtils_AC500.lib* (*“Library Manager ➔ ABB-AC500 ➔ Use Cases”*). The following is output:

0	Battery empty
20	Remaining battery charge below 20 %, battery must be replaced
100	Battery charge OK

On the device, the battery status can be checked with the function keys of a processor module. *Chapter 1.6.5.1.6 “LEDs, display and function keys on the front panel” on page 3486*

Chapter 1.6.5.1.6.5.4 “Reading out values” on page 3507

AC500-eCo V3 data buffering

The AC500-eCo V3 buffers the following data in case of "control voltage off":

- Retentive variables in FLASH (VAR_RETAIN..END_VAR) *Chapter 1.6.5.1.1 “Handling of remanent variables for AC500 V3 products” on page 3456*
- Date and time of the real-time clock are using an integrated gold-capacitor with a lower retention time as a battery.

> *The AC500-eCo V3 has no battery but stores the remanent data in flash or the real-time clock using a gold-capacitor, there is no battery or gold-capacitor status or survey.*
>
> *In case of “control voltage off”, the real-time clock is buffered for about 7 days for Basic CPU with TA5131-RTC and about 20 days for Standard or Pro CPUs at 40 °C using temperature.*

1.6.5.1.5 AC500-eCo V3 processor module, LEDs, RUN/STOP switch on front panel

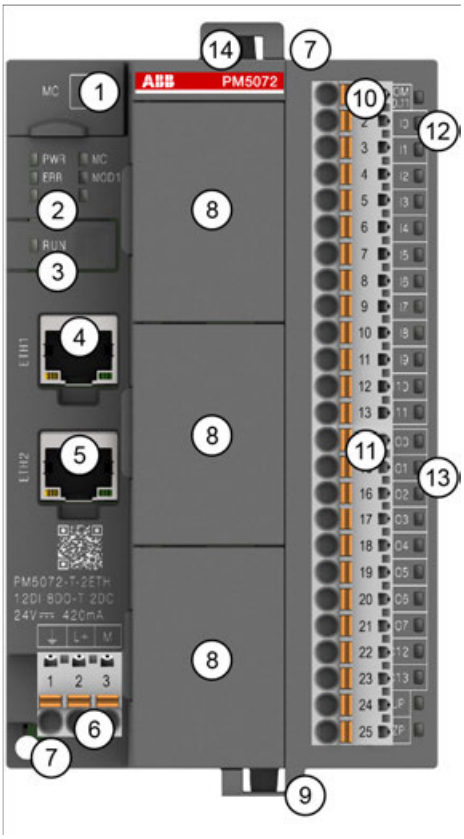


Fig. 304: Example: PM5072-T-2ETH

1 Micro memory card slot
2 5 LEDs to display the states of the processor module (Power, Error, Run, MC, MOD1)
3 RUN button
4 RJ45 female connector for Ethernet1 connection
5 RJ45 female connector for Ethernet2 connection (available for PM5072-T-2ETH(W))
6 3-pin terminal block for power supply 24 V DC
7 2 holes for screw mounting
8 Option board slot cover for option board slot (the number of available slots varies according to the CPU type)
9 Cable fixing
10 13-pin terminal block for onboard I/Os
11 12-pin terminal block for onboard I/Os (not available on PM5012-x-ETH)
12 12 LEDs to display the states of the signals
13 10 LEDs to display the states of the signals
14 Cable fixing accessory TA5301-CFA on the top of the housing (optional)

The processor module is shown with pluggable terminal blocks. These terminal blocks must be ordered separately.

The cable fixing accessory on the top of the housing is optional.

Please use TA5301-CFA cable fixing accessory to provide strain relief.

It can also be used for AC500-eCo I/O modules.

The PM50x2 processor modules are supplied with option board slot covers as standard.

There are various TA51xx option boards for the processor modules that can be ordered separately.

Which and how many option boards can be plugged, depends on the respective processor module.

State LEDs and operating elements

RUN/STOP button

The processor modules, PM50xx series, have a RUN/STOP button. By pressing the RUN/STOP button, the processer modules switch between RUN mode and STOP mode. By long-pressing RUN/STOP button during the processor module power on phase, the processor module will be in MOD1.

State LEDs

The processor modules PM50xx indicate their states of operation via 5 LEDs located on the upper left side of the processor module.

LED	State	Color	LED = ON	LED = OFF	LED flashing
PWR	Power supply	Green	Power supply present	Power supply missing	-
MC	Micro memory card indication	Yellow	Micro memory card is in the socket	Micro memory card is not in the socket	Micro memory card is in read/ write state: any file on card is opened, means activity on card
ERR	Error indication	Red	An error occurred	No errors or only warnings encountered (E4 errors). The LED behavior for the error classes 2 to 4 is configurable.	Fast flashing (4 Hz) displays together with the RUN LED a currently running firmware-upgrade or writing data to the Flash-EPROM. Slow flashing (1 Hz) alone displays shutdown of Request To Send. Medium flashing (2 Hz) alone displays at start of PLC if reboot after watchdog.
MOD1	Mode 1 indication	Yellow	Processor module is in mode 1 state	Processor module is not in mode 1 state	-

LED	State	Color	LED = ON	LED = OFF	LED flashing
RUN	RUN/STOP state	Green	Processor module is in state RUN	Processor module is in state STOP	Fast flashing (4 Hz): The processor module is reading/writing data from/to the memory card. If the ERR-LED is also flashing, data is being written to the Flash-EPROM.
					Slow flashing (1 Hz): The firmware update from the memory card has been completed successfully or Boot project is being updated. Slow flashing (0.5 Hz) together with MOD1 LED ON: Mode1: Boot project is not loaded.
Two LEDs below “ERR” and “MOD1”	Configurable	Yellow	Configurable	Configurable	Additional two LEDs are reserved and can be controlled from IEC user code with FB PmLedSet

User configurable LEDs

The AC500-eCo V3 processor module also provides 2 LEDs below the state LEDs which can be used by user and driven by an application.

The LEDs can be used into a project and controlled using special function blocks which are contained in the PM AC500 library. The POU is PmLedSet located in folder LED control.

I/O LEDs

The processor module provides up to 10 LEDs (PM5012-x-ETH), 20 LEDs (PM5032-R-ETH, PM5052-R-ETH), or 22 LEDs (PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH) to display the states of the inputs and outputs.

Processor module	LED	State	Color	LED = ON	LED = OFF
PM5012-x-ETH	I0..I5	Digital input	Yellow	Input is ON	Input is OFF
	O0..O3	Transistor output	Yellow	Output is ON	Output is OFF

Processor module	LED	State	Color	LED = ON	LED = OFF
	NO0..NO3	Relay output	Yellow	Output is ON	Output is OFF
PM5032-x-ETH PM5052-x-ETH	I0..I11	Digital input	Yellow	Input is ON	Input is OFF
	O0..O7	Transistor output	Yellow	Output is ON	Output is OFF
	NO0..NO5	Relay output	Yellow	Output is ON	Output is OFF
	C12, C13	Digital configurable input/ output	Yellow	Input/Output is ON	Input/Output is OFF
PM5072-T-2ETH PM5072-T-2ETHW	I0..I11	Digital input	Yellow	Input is ON	Input is OFF
	O0..O7	Transistor output	Yellow	Output is ON	Output is OFF
	C12, C13	Digital configurable input/ output	Yellow	Input/Output is ON	Input/Output is OFF

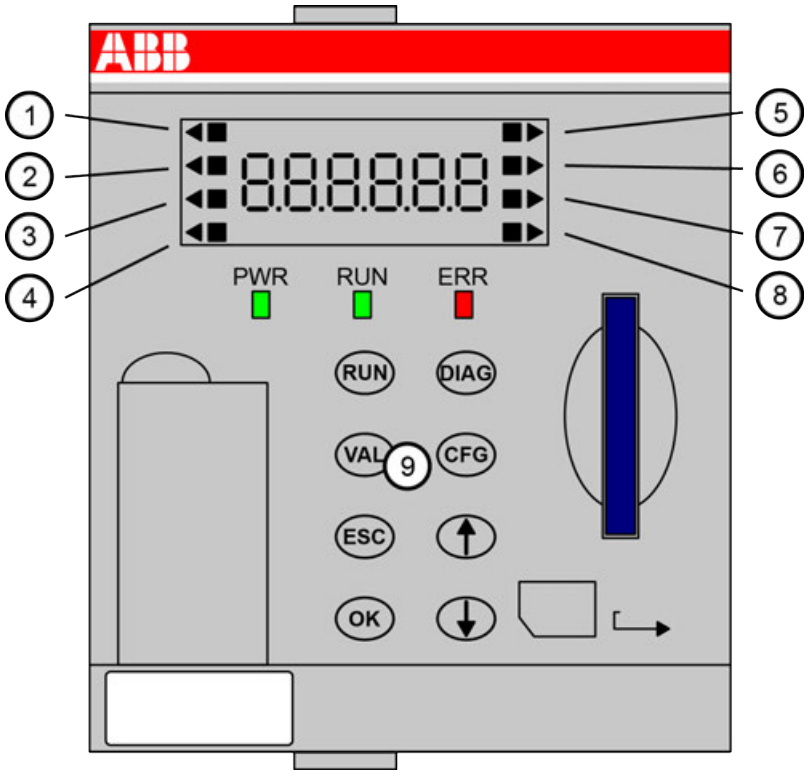
Ethernet state LEDs

Table 617: State LEDs at Ethernet connector

LED	Color	OFF	ON	Flashing
Activity	Yellow	No activity	---	Activity
Link	Green	No link	Link	---

1.6.5.1.6 LEDs, display and function keys on the front panel

Overview



The display of a processor module is equipped with a background-lighted 7-segment display. This display consists of 6 digits for plain text or error codes.

> *Some functionalities may be not yet supported by the product. Please refer to the release notes of the product at time of release.*

Display indicators

- A black square (■) denotes the state/working activity of the corresponding object on the left/right side of the display. The black square flashes according to the device's activity, e.g. during data exchange on ETH1, ETH2, COM1, etc.

> ***MC activity***
> *For the activity of the memory card the black square (■) is shown as long as a file is open on memory card.*

- A black triangle (▶) points to the selected item/interface on the left/right side of the display to be configured or read. Further, it acts as a cursor for the count up/count down function keys.

> *A black triangle (◀) at the BATT item indicates a missing or uncharged battery.*

The indicators point to the following items on the left side of the display:

No.	On the left Side	Description
1	MC (memory card)	Refers to the memory card status.
2	SYS (system)	Refers to the system status.
3	BATT (battery)	Refers to the battery status.
4	I/O bus	Refers to I/O bus connection.

The indicators point to the following items on the right side of the display:

No.	On the right side	Description
5	ETH1	Refers to the first Ethernet interface.
6	ETH2	Refers to the second Ethernet interface.
7	COM1	Refers to COM1 interface.
8	CAN	Refers to CAN interface.

9	Function keys on front panel

Processor module	Display variant	Description
PM56xx-2ETH		Display of a processor module with support for 2 Ethernet interfaces, CAN and COM1.

Text outputs of the display

Display	Description
	Display on system start (power on).
	PLC is in boot mode.
	Is shown on startup after „Boot“, when a wrong DisplayFW is detected, e.g. the old version 3.0. Please update display with DisplayFW 4.1 (or higher).
	PLC is in initialization mode.

Display	Description
	PLC is in STOP mode.
	No system firmware (SystemFW) available. Start update firmware. PLC is waiting for a firmware download via Automation Builder or memory card. See *Chapter 1.6.6.1.4.2 " AC500 V3 firmware installation and update" on page 3653*
	PLC is in RUN mode. **Switch into RUN mode is only possible if a valid boot project is available in the flash memory.**
	Only in RUN mode and as of SystemFW V3.2.0 Reminder: ***demo license*** PLC runs in „Demo mode“, since at least one feature license is missing. Will be displayed for 5 minutes at every license check If „Demo time“ expires, PLC will go to „Stop“.
	Only in RUN mode and as of SystemFW V3.2.0 10 minutes step reminder: ***license was removed*** PLC runs in „Grace mode“, since at least one feature license which has been available disappeard. PLC is waiting for this license. Will be displayed for 5 minutes If „Grace time“ expires, PLC will go to „Stop“.
	New as of SystemFW 3.3.1.103. Text is shown if no communication between CPU and display is possible due do very high CPU load (e.g. endless loop in user program and not activated task watchdog).

Startup procedure of the PLC

Startup procedure of a new PLC from factory

State	Display	Description
0		Display on system start (power on).
1		PLC is in boot mode.

State	Display	Description
2		PLC is in initialization mode.
3		No system firmware (SystemFW) available. Start update firmware. PLC is waiting for a firmware download via Automation Builder or memory card. See *Chapter 1.6.6.1.4.2 " AC500 V3 firmware installation and update" on page 3653*

Startup procedure of a PLC with system firmware

The startup procedure depends on the selected PLC mode.

PLC mode	Display	Startup Behavior
00		The user program will be loaded and run. PLC changes to mode „*RUN*".
01		User program will not be loaded / run. PLC stay in mode „*STOP*".
02		Reserved for further development (currently like Mode 00).

*Mode 01 can be activated via function key **CFG** (see Further information on page 3487), or by pressing function key **RUN** during startup of PLC until Mode 01 is shown on display.*

State	Display	Description
0		Display on system start (power on).
1		PLC is in boot mode (see *Further information on page 3487*).
2		PLC is in initialization mode (see *Further information on page 3487*).
3		PLC is in STOP mode (see *Further information on page 3487*). **Same as status Stop in Automation Builder.**
4		PLC is in RUN mode (see *Further information on page 3487*). **Switch into RUN mode is only possible if a valid boot project is available in the flash memory.**
5		Only in RUN mode and as of SystemFW V3.2.0 Reminder: ***demo license*** PLC runs in „Demo mode“, since at least one feature license is missing. Will be displayed for 5 minutes at every license check If „Demo time“ expires, PLC will go to „Stop“.
6		Only in RUN mode and as of SystemFW V3.2.0 10 minutes step reminder: ***license was removed*** PLC runs in „Grace mode“, since at least one feature license which has been available disappeard. PLC is waiting for this license. Will be displayed for 5 minutes If „Grace time“ expires, PLC will go to „Stop“.

Description of LEDs

The LEDs below the display indicate the status of the processor module:

LED	State	Color	LED = ON	LED = OFF	LED flashes
Power LED (PWR)	Denotes the power supply state of the processor module	Green	Voltage is present (24 V DC)	Voltage is missing	-
Run LED (RUN)	Denotes the activity state of the processor module	Green	Processor module is in RUN mode	Processor module is in STOP mode	If the LED flashes fast (4 Hz) a firmware update is finished with no errors. If the Run LED flashes fast (4 Hz), alternating with a flashing Run LED the firmware is updated. To enforce boot mode 1, keep the RUN function key pressed during the boot procedure. In this case, the Run LED flashes slowly (1 Hz). A subsequent project download (from within Automation Builder) cancels the blinking.
Error LED (ERR)	Denotes an error	Red	An error has occured.	No errors or only warnings have occurred.	If the Error LED flashes slowly (1 Hz) a firmware update from the memory card is finished with errors. If the Error LED flashes fast with AC500 on display a fatal system error has occurred. If the Error LED flashes fast (4 Hz) alternating with a flashing Run LED the firmware is updated.

A running processor module is indicated with the state RUN on the display, a deactivated processor module is indicated with the state STOP. In both cases the display's backlight is off.

Description of the function keys

Overview

The processor module can be operated manually using the function keys on the front panel:

Function Key	Description	Description
RUN	Run	Toggles between RUN and STOP mode. **Switching into RUN mode is only possible if an error free project has been created and downloaded with Automation Builder**.
VAL	Value	Shows different state values of the processor module.
ESC	Escape	Quits the current menu, submenu or function without saving.
OK	OK / Acknowledgement	Acknowledges the current value or selects a menu/ submenu. Changes that have been sent to the processor module successfully are confirmed with *donE* on the display.
DIAG	Diagnostic	Allows evaluation of error messages in detail.
CFG	Configuration	Show/set IP configuration, PLC startup mode and Ethernet address. Enters submenus.
↑	Count up / navigate in submenu	Press the function key repeatedly in order to increase the value each time by 1, or navigate in submenu to previous entry Keep the function key pressed in order to count up fast.
↓	Count down / navigate in submenu	Press the function key repeatedly in order to decrease the value each time by 1, or navigate in submenu to next entry. Keep the function key pressed in order to count down fast.

Backlight is switched on for about 20 seconds by pressing any function key.

Start and stop PLC

Function key *RUN*

State	Description Menu level 0	Result on pressing one of the function keys		
		RUN	OK	ESC
0		Short click: State 1 is displayed. Long click (>5 sec): State 2 is displayed.	No action	No action

State	Description Menu level 0	Result on pressing one of the function keys		
		RUN	OK	ESC
1	 PLC only in state RUN if a correct project is in RAM of PLC	State 0 is displayed. STOP - same as *Online stop* in Automation Builder (halt, no init of variables)		
2		***RUN*** LED=ON	Perform RESET same as *Online reset* in Automation Builder (stop and init variables) State 0 is displayed.	No RESET State 0 is displayed.

Configuration

Configuration CPU firmware SystemFW V3.1.x and DisplayFW V3.0

Function key *CFG* main menu with *ETH1 / ETH2 mode*: *"Two separate interfaces"*

(see *Chapter 1.6.6.2.9.2 "Switch functionality of Ethernet interfaces ETH1/ETH2" on page 3736*)

Navigation starts with the processor module being in RUN/STOP mode (State 0). By pressing one of the three function keys a certain action is triggered. The result of this action is described in the result columns of the tables.

State	Description - Main menu 1	Result on pressing one of the function keys		
		CFG	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed.	Remains in RUN/ STOP mode.	Remains in RUN/ STOP mode.
1		State 2 is displayed.	Return into RUN/ STOP mode.	Refers to sub menu 1
2	 Change the values with the **Count up/Count down** function keys.	State 3 is displayed.	Return into RUN/ STOP mode.	Shows DONE, your settings are saved. Return into RUN/STOP mode.
3		State 4 is displayed.	Return into RUN/ STOP mode.	Refers to sub menu 1

State	Description - Main menu 1	Result on pressing one of the function keys		
4	 Change the values with the **Count up/Count down** function keys.	State 5 is displayed.	Return into RUN/ STOP mode.	Shows DONE, your settings are saved. Return into RUN/STOP mode.
5	 Change the values with the **Count up/Count down** function keys. See also *Further information on page 3489*.	State 1 is displayed.	Return into RUN/ STOP mode.	Shows DONE, your settings are saved. Return into RUN/STOP mode.

Function key *CFG* main menu with *ETH1 / ETH2 mode*: *"Switch functionality ETH1-ETH2"*

(see *Chapter 1.6.6.2.9.2 "Switch functionality of Ethernet interfaces ETH1/ETH2" on page 3736*)

Navigation starts with the processor module being in RUN/STOP mode (State 0). By pressing one of the three function keys a certain action is triggered. The result of this action is described in the result columns of the tables.

State	Description - Main menu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed.	Remains in RUN/ STOP mode.	Remains in RUN/ STOP mode.
1		State 2 is displayed.	Return into RUN/ STOP mode.	Refers to sub menu 1
2	 Change the values with the **Count up/Count down** function keys.	State 3 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 2 is displayed.
3	 Change the values with the **Count up/Count down** function keys. See also *Further information on page 3489*.	State 1 is displayed.	Return into RUN/ STOP mode.	Shows DONE, your settings are saved. Return into RUN/STOP mode.

Function key *CFG* sub menu IPETH1 or IPETH2; *DHCP not active*

State	Description - Submenu 1	Result on pressing one of the function keys		
		CFG	ESC	OK
1.1	 IPETH1 or IPETH2	State 2 is displayed.	Return into RUN/ STOP mode.	State 1.2 is displayed.
1.2	 IP Configuration (address, subnet mask, gateway)	State 1.3 is displayed.	Aborts the menu unchanged. Return to State 1.1	State 3.2 is displayed.
1.3	 Reset to production data (default settings)	State 1.4 is displayed.	Aborts the menu unchanged. Return to State 1.1	Activate RESET to default by pressing ***OK*** twice. Shows DONE, your settings are saved. Return into RUN/STOP mode.
1.4	 Activate DHCP Sets a DHCP address.	State 1.2 is displayed.	Aborts the menu unchanged. Return to State 1.1	Activate DHCP to default by pressing ***OK*** twice Shows DONE, your settings are saved. Return into RUN/STOP mode.

Function key *CFG* sub menu IPETH1 or IPETH2; *DHCP active*

State	Description - Submenu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
2.1	 IPETH1 or IPETH2	State 2 is displayed.	Aborts the menu unchanged. Return to State 0.	State 2.2 is displayed.
2.2	 DHCP active	State 2.3 is displayed.	Aborts the menu unchanged. Return to State 2.1.	--
2.3	 IP Configuration (address, subnet mask, gateway)	State 2.4 is displayed.	Aborts the menu unchanged. Return to State 2.1.	State 3.2 is displayed.

State	Description - Submenu 2	Result on pressing one of the function keys		
2.4	 Reset to production data (default settings)	--	Aborts the menu unchanged. Return to State 2.1.	Activate RESET to default by pressing ***OK*** twice Shows DONE, your settings are saved. Return into RUN/STOP mode.

Function key *CFG* sub menu STATIC

State	Description - Submenu 3	Result on pressing one of the function keys		
		CFG	ESC	OK
3.1	 IP Configuration (address, subnet mask, gateway)	State 2.4 is displayed.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2)	State 3.2 is displayed.
3.2	 IP address A1-A4 ★ Number is blinking if value has changed and is not yet sent to CPU	State 3.3 is displayed.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2)	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.
3.3	 Subnet mask N1-N4 ★ Number is blinking if value has changed and is not yet sent to CPU	State 3.4 is displayed.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2)	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

State	Description - Submenu 3	Result on pressing one of the function keys		
3.4	 Gateway G1-G4 ★ Number is blinking if value has changed and is not yet sent to CPU	State 3.2 is displayed again.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2) Aborts the menu unchanged. Return to State 1.	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

Function key *CFG* sub menu ADR

State	Description - Submenu 4	Result on pressing one of the function keys		
		CFG	ESC	OK
4.1		State 4.2 is displayed.	Aborts the menu unchanged. Return to State 1	*DHCP not active*: State 1.2 is displayed *DHCP active*: State 2.2 is displayed
4.2	 Change the values with the **Count up/Count down** function keys starting with current value. ★ Number is blinking if value has changed and is not yet sent to CPU	State 2.3 is displayed.	Aborts the menu unchanged. Return to State 4.1	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

State	Description - Submenu 4	Result on pressing one of the function keys		
4.3	 Subnet mask N1-N4 ★ Number is blinking if value has changed and is not yet sent to CPU	State 4.1 is displayed.	Aborts the menu unchanged. Return to State 4.1	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

Configuration CPU firmware SystemFW >=V3.2.0 and DisplayFW >=V4.1

Function key *CFG* menu level 1, with *ETH1 / ETH2 mode*: *"Two separate interfaces"*

(see *Chapter 1.6.6.2.9.2 "Switch functionality of Ethernet interfaces ETH1/ETH2" on page 3736*)

Navigation starts with the processor module being in RUN/STOP mode (State 0). By pressing one of the five function keys a certain action is triggered. The result of this action is described in the result columns of the tables.

State	Description - *CFG* menu level 1	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0	The processor module is in RUN/ STOP mode.	State 1 is displayed.			Remains in RUN/STOP mode.	Remains in RUN/STOP mode.
1	 Switch is OFF	Refers to submenu level 2 *"Function key CFG submenu show / set PLC ID " on page 3507*	State 2 is displayed if KNX functionality is active. State 3 is displayed if KNX functionality is inactive.	State 6 is displayed.	Return into RUN/STOP mode.	
2	 KNX program button (appears if functionality is active)		State 3 is displayed.	State 1 is displayed.	Return into RUN/STOP mode.	

State	Description - *CFG* menu level 1	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
3		Refers to submenu level 2 ↳ *"Function key CFG menu level 2 (IPETH1 or IPETH2); " on page 3501*	State 4 is displayed.	State 2 is displayed.	Return into RUN/STOP mode.	Only active if no changes in ***CFG*** menu. Return into RUN/STOP mode.
4		Refers to submenu level 2 ↳ *"Function key CFG menu level 2 (IPETH1 or IPETH2); " on page 3501*	State 5 is displayed.	State 3 is displayed.	Return into RUN/STOP mode.	
5	  Note: COM1 mode RS-232 (default) or RS-485 can only be shown but not changed. This is a PLC boot paramter (see ↳ *Chapter 1.6.6.2.14.3 "Setting up a serial interface" on page 3798*) „“ and must be set in AB ↳ *Chapter 1.6.6.2.14.3.1 "Configuration" on page 3798*. Mode is activated in PLC boot process.		State 6 is displayed.	State 4 is displayed.	Return into RUN/STOP mode.	Return into RUN/STOP mode.
6		Refers to submenu set startup mode of PLC ↳ *"Function key CFG submenu show / set startup mode of PLC " on page 3506*	State 1 is displayed.	State 5 is displayed.	Return into RUN/STOP mode.	

Function key *CFG* menu level 1, with *ETH1 / ETH2 mode*: *"Switch functionality ETH1-ETH2"*

(see ↳ *Chapter 1.6.6.2.9.2 "Switch functionality of Ethernet interfaces ETH1/ETH2" on page 3736*)

Navigation starts with the processor module being in RUN/STOP mode (State 0). By pressing one of the five function keys a certain action is triggered. The result of this action is described in the result columns of the tables.

State	Description - *CFG* menu level 1	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0	The processor module is in RUN/ STOP mode.	State 1 is displayed.			Remains in RUN/STOP mode.	Remains in RUN/STOP mode.
1	 Switch is ON		State 2 is displayed if KNX function-ality is active. State 3 is displayed if KNX function-ality is inactive.	State 5 is displayed.	Return into RUN/STOP mode.	Return into RUN/STOP mode.
2	 KNX program button (appears if function-ality is active)		State 3 is dis-played.	State 1 is displayed.	Return into RUN/STOP mode.	Return into RUN/STOP mode.
3		Refers to submenu level 2 ↳ *"Func-tion key CFG menu level 2 (IPETH1 or IPETH2); " on page 3501*	State 4 is dis-played.	State 2 is displayed.	Return into RUN/STOP mode.	Only active if no changes in ***CFG*** menu. Return into RUN/STOP mode.

State	Description - *CFG* menu level 1	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
4	  Note: COM1 mode RS-232 (default) or RS-485 can only be shown but not changed. This is a PLC boot parameter (see *Chapter 1.6.6.2.14.3 "Setting up a serial interface" on page 3798*) „“ and must be set in AB *Chapter 1.6.6.2.14.3.1 "Configuration" on page 3798*. Mode is activated in PLC boot process.		State 5 is displayed.	State 3 is displayed.	Return into RUN/STOP mode.	Return into RUN/STOP mode.
5		Refers to submenu set startup mode of PLC *"Function key CFG submenu show / set startup mode of PLC " on page 3506*	State 1 is displayed.	State 4 is displayed.	Return into RUN/STOP mode.	Return into RUN/STOP mode.

Function key *CFG* menu level 2 *(IPETH1 or IPETH2)*;

State	Description - *CFG* menu level 2	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0	 IPETH1 or IPETH2	State 1 is displayed.			Return into RUN/STOP mode.	Return into RUN/ STOP mode.

State	Description - *CFG* menu level 2	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
1	 IP Configuration (address, subnet mask, gateway)	Refers to submenu *StAtIC* ↳ *"Function key CFG menu level 3, show / set STATIC" on page 3503*	State 2 is displayed.	State 4 is displayed.	Case 1, no submenu is entered: Return into RUN/STOP mode. Case 2, no changes: State 0 is displayed.	Send all changes to CPU. State 5 is displayed.
2		Refers to submenu set *dHCP* ↳ *"Function key CFG menu level 2 Show/set DHCP" on page 3504*	State 3 is displayed.	State 1 is displayed.		
3	 Activate DHCP Sets a DHCP address.	Refers to submenu set *Id* ↳ *"Function key CFG menu level 2 Show/set Id" on page 3505*	State 4 is displayed.	State 2 is displayed.	State 1 is displayed.	
4	 Reset to production data (default settings)	Refers to submenu set *rESEt* ↳ *"Function key CFG submenu show / set RESET" on page 3506*	State 1 is displayed.	State 3 is displayed.		Send all changes to CPU. State 5 is displayed.
5					Remain all changes State 1 is displayed.	State 6 is displayed.
6		Changes applied *donE* is displayed for 2 sec. then return into RUN/STOP mode.				
		Changes failed *FAILEd* is displayed for 2 sec. then return into RUN/STOP mode.				

Function key *CFG* menu level 3, *show / set STATIC*

State	Description - Sub-menu STATIC	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0	 IP Configuration (address, subnet mask, gateway)	State 1 is dis-played.			Aborts the menu unchanged. Return to IPETH1 or IPETH2	No changes: return to IPETH1 or IPETH2 Changes: (StAtIC blinks) state 5 of previous table is dis-played
1	    IP address A1-A4 **If submenu is entered**: Number is blinking if value has changed and is not yet sent to CPU	Refers to submenu of A1-A4	Count down A1-A4	Count up A4-A1	Aborts the menu unchanged. Return to ***CFG** menu level 2* ***StAtIC***	Take over new values, but don´t send to CPU. Return to ***CFG** menu level 2* ***StAtIC***
2	    Subnet mask N1-N4 **If submenu is entered:** Number is blinking if value has changed and is not yet sent to CPU	Refers to submenu of N1-N4	Count down N1-N4	Count up N4-N1	Aborts the menu unchanged. Return to ***CFG** menu level 2* ***StAtIC***	Take over new values, but don´t send to CPU. Return to ***CFG** menu level 2* ***StAtIC***

State	Description - Sub-menu STATIC	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
3	 Gateway G1-G4 **If submenu is entered:** Number is blinking if value has changed and is not yet sent to CPU	Refers to submenu G1-G4	Count down G1-G4	Count up G4-G1	Aborts the menu unchanged. Return to ***CFG** menu level 2 **StAtIC***	Take over new values, but don´t send to CPU. Return to ***CFG** menu level 2 **StAtIC***

Function key *CFG* menu level 4, Example: *I/P address A4*

State	Description - *CFG* menu level 4	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0	 A4 is blinking if sub-menu is entered	No action	Count down value. Value is blinking if changed	Count up value. Value is blinking if changed	Aborts the menu unchanged. Shows unchanged value.	Take over new value, but don´t send to CPU. Go back to menu level 3 (here subnet mask N1)

Function key *CFG* menu level 2 *Show/set DHCP*

State	Description - Show/set DHCP	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0		State 1 is dis-played.			Aborts the menu unchanged.	
1	 IP address A1-A4 ★ Value is blinking if value has changed and is not yet sent to CPU		State 2 is displayed.	State 2 is displayed.	Aborts the menu unchanged. Go back to ***CFG*** menu level 1.	Take over new values, but don´t send to CPU. Text is blinking if value is changed. State 3 is displayed.
2			State 1 is displayed.	State 1 is displayed.	Go to sub-menu ***StAtIC***.	

State	Description - Show/set DHCP	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
3					Remain all changes State 1 is displayed.	State 4 is displayed.
4		Changes applied ***donE*** is displayed for 2 sec. then return into RUN/STOP mode.				
		Changes failed ***FAILEd*** is displayed for 2 sec. then return into RUN/STOP mode.				

Function key *CFG* menu level 2 *Show/set Id*

State	Description - Show/set Id	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0			Count down value: 255 ... 000, starting with current value	Count up value: 000 ... 255, starting with current value	Case 2, no changes: stop blinking, return to menu ↪ *"Function key CFG menu level 2 (IPETH1 or IPETH2); " on page 3501* .	Go to*CFG* menu level 2 *(IPETH1 or IPETH2)* state 5-6.
1	 IP address A1-A4 ★ Number is blinking if value has changed and is not yet sent to CPU		Count down value: 255 ... 000, starting with current value	Count up value: 000 ... 255, starting with current value	Discard the changes. Stop blinking. Show previous value.	Go to*CFG* menu level 2 *(IPETH1 or IPETH2)* state 5-6.
2			Count down value: 255 ... 000, starting with current value	Count up value: 000 ... 255, starting with current value	Discard the changes. Stop blinking. Show previous value.	Go to*CFG* menu level 2 *(IPETH1 or IPETH2)* state 5-6.

Function key *CFG* submenu show / set RESET

State	Description - Sub-menu show / set RESET	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0		State 1 is displayed.	No action	No action	Aborts the menu unchanged. Return to sub-menu level 1	
1	 ★ Display is blinking if value has changed and is not yet sent to CPU				Aborts the menu unchanged. Return to sub-menu level 1	Discard all made changes. Stop blinking. Send command "reset to factory settings" to CPU *Chapter 1.6.5.1.6.3.1 "Startup procedure of a new PLC from factory" on page 3488* „reset Ask confirmation. Go back to default menue RUN/ STOP. State 2 is displayed.
2					Aborts the menu unchanged. Return to sub-menu STATIC	Sends changed values to CPU, displays state 3.
3		Changes applied ***donE*** is displayed for 2 sec. then return into RUN/STOP mode.				
		Changes failed ***FAILEd*** is displayed for 2 sec. then return into RUN/STOP mode.				

Function key *CFG* submenu show / set startup mode of PLC

State	Description - Sub-menu show / set startup mode of PLC	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
Start			Count down value: 02 ... 0, starting with current value	Count up value: 00 ... 02, starting with current value	Aborts the menu. Go back to main menue RUN/STOP	Sends changed values to CPU.

State	Description - Sub-menu show / set startup mode of PLC	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
1	 ★ Text is blinking if value has changed and is not yet sent to CPU		Count down value: 02 ... 00, starting with current value	Count up value: 00 ... 02, starting with current value	Aborts the menu. Go back to main menue RUN/STOP	Displays 2 sec. ***donE*** or ***FAILEd*** Go back to main menue RUN/STOP
2	 ★ Text is blinking if value has changed and is not yet sent to CPU		Count down value: 02 ... 00, starting with current value	Count up value: 00 ... 02, starting with current value	Aborts the menu. Go back to main menue RUN/STOP	

Function key *CFG* submenu show / set PLC ID

State	Description - Sub-menu show / set startup mode of PLC	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
0	 Switch is OFF	State 1 is displayed			Return into RUN/STOP mode.	
1	 PLC ID		State 2 is displayed	State 2 is displayed	Return into RUN/STOP mode.	
2	  ★ Text PLC is blinking if edit mode with keys up/down is enabled ★ Number is blinking if value has changed and is not yet sent to CPU		Count down value: 255→0, starting with current value	Count up value: 000→255, starting with current value	Aborts the menu. Go back to main menue RUN/STOP	Takes over new value, if changed

Reading out values

Reading out values CPU firmware SystemFW 3.1.x and DisplayFW 3.0

The following settings of the processor module can be read out by pressing the function key ***VAL*** repeatedly:

1. Displays time of the processor module (hh.mm.ss).
2. Displays date of the processor module (yy.mm.dd).
3. Displays state of battery (ub 100 = 100%, ub 020 = 20% or ub 000 = empty).
4. Displays version of display firmware (e.g. d 3.0 r (= display version 3.0 release).

5. Displays version of CPU firmware (e.g. C 3.1.0r (= CPU version 3.1.0 release).
6. Displays CPU type.
7. Displays default text (RUN/STOP).

Reading out values CPU firmware SystemFW >=V3.2.0 and DisplayFW >=V4.1

Function key *VAL* By pressing Function Key ***VAL*** state 1 is displayed

State	Description Menu *VAL*	Result on pressing one of the function keys				
		CFG	↓	↑	ESC	OK
1	 Time of the processor module (hh.mm.ss).	No action	State 2 is displayed	State 6 is displayed	Go back to main menue RUN/STOP	
2	 Date of the processor module (yy.mm.dd).		State 3 is displayed	State 1 is displayed		
3	 State of battery (ub 100 = 100%, ub 020 = 20% or ub 000 = empty).		State 4 is displayed	State 2 is displayed		
4	 Version of display firmware (e.g. d 4.1 r (= display version 4.1 release).		State 5 is displayed	State 3 is displayed		
5	 Version of CPU firmware (e.g. C 3.2.0r (= CPU version 3.2.0 release).		State 6 is displayed	State 4 is displayed		
6	 CPU type.		State 1 is displayed	State 5 is displayed		

Reading out diagnosis messages on the CPU

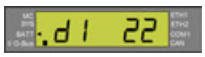
Table 618: Example: no diagnosis message in status list

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed	-	-	-	-
1			No action	No action	Return into RUN/STOP mode.	

Table 619: Example: diagnosis messages in status list

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed	-	-	-	-
1	 Number of diagnosis messages; here 4		Go to first/ next diagnosis message in status list (e.g., state 2)	Go to last/ previous diagnosis message in status list	Return into RUN/STOP mode.	Return into RUN/STOP mode.
2	 Diagnosis message example: *Error battery empty or missing* Toggling between state 2 and 3	Selects displayed diagnosis message and shows details ↻ *Table 620 "Example: error battery empty or missing" on page 3510*	Go to first/ next diagnosis message in status list	Go to last/ previous diagnosis message in status list	Return into RUN/STOP mode.	Acknowledge and return into RUN/STOP mode.
3	 Error ID example Toggling between state 2 and 3					

Table 620: Example: error battery empty or missing

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	 E4 = error severity 4 bAt = subdevice battery Toggling between state 0 and 1	State 2 is displayed	State 2 is displayed	State 6 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
1	 Error ID example Toggling between state 0 and 1					
2	 Error number 8 Battery is missing or empty		State 3 is displayed	State 0 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Displays state 0 Return to diagnosis status list
3	 Detail 1 Subdevice 22: battery		State 4 is displayed	State 2 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
4	 Detail 2 Error type 0: device		State 5 is displayed	State 3 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
5	 Detail 3 Error type number 0: device itself		State 6 is displayed	State 4 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
6	 Detail 4 Additional information 0: none		State 1 is displayed	State 5 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list

Enable flashing of display

Blink functionality As of SystemFW 3.1.0 and AB 2.1.0 the Blink functionality is implemented. *"Blink"* means – activate flashing of backlight of AC500 display in Automation Builder via *"IP-Configuration"* tool
↳ *Chapter 1.6.6.2.2.4.2.2.4 "Blink functionality" on page 3685*.

Wink functionality As of SystemFW 3.1.0 and AB 2.0.0 the Wink functionality is implemented. *"Wink"* means – activate flashing of backlight of AC500 display in Automation Builder via communication settings Wink functionality.

Function blocks

PmErrLedSet This function block switches the ERR-LED ON and OFF.

PmDispSetText With this function block a text can be displayed on the CPU.

↳ *Chapter 1.10 "Reference, function blocks" on page 4292*

1.6.5.1.7 Onboard technologies

Ethernet

Ethernet protocols and ports for AC500-eCo V3 processor modules

Supported as of Automation Builder V 2.4.1

Description	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH	≥ CPU firmware
ABB netConfig	x	x	x	x	V3.4.1
Online access with driver 3S UDP BlkDrvUdp	x	x	x	x	V3.4.1
Online access with driver 3S TCP/IP BlkDrvTcp	x	x	x	x	V3.4.1
Modbus TCP server	x	x	x	x	V3.4.1
Modbus TCP client with POU ETHx_MOD_MAST	x	x	x	x	V3.4.1
UDP out of user program with library netBaseService.lib	x	x	x	x	V3.4.1
UDP data exchange, Network variables	x	x	x	x	V3.4.1
TCP/IP out of user program with library netBaseService.lib	x	x	x	x	V3.4.1
Web server on PLC with web visualization		x	x	x	V3.4.1
IEC60870-5-104 control station incl. 2nd connection and 2nd port					-
IEC60870-5-104 substation incl. 2nd port				x	V3.4.1
FTP server (See ↳ *Chapter 1.6.6.3.5.1 "Configuration of FTP server" on page 3917*)		x	x	x	V3.4.1
CODESYS network variables	x	x	x	x	V3.4.1
OPC DA server	x	x	x	x	V3.4.1

Description	PM5012 -x-ETH	PM5032 -x-ETH	PM5052 -x-ETH	PM507 2-T-2ETH	≥ CPU firm-ware
OPC UA server		x	x	x	V3.4.1
ICMP – ping out of user project with POU *ETHx_ICMP_PING*	x	x	x	x	V3.4.1
DHCP client	x	x	x	x	V3.4.1
SNTP (Simple Network Time Protocol) client system solution (See *Chapter 1.6.6.3.4.2.1 "(S)NTP client configuration" on page 3913*)	x	x	x	x	V3.4.1
SNTP (Simple Network Time Protocol) server system solution (See *Chapter 1.6.6.3.4.2.2 "(S)NTP server configuration" on page 3916*)	x	x	x	x	V3.4.1
Maximum number of Input/output allowed variable on Ethernet for the protocol		1 kB /1 kB	1 kB /1 kB	2 kB /2 kB	V3.4.1
IEC 61850 (MMS server, GOOSE) [2])				x	V3.4.1
EthernetIP Scanner [1, 2])		x	x	x	AB 2.4.1/ FW 3.4.1
EthernetIP Adapter [1, 2])		x	x	x	AB 2.4.1/ FW 3.4.1
KNX - Building communication [2])			x	x	V3.4.1
BACnet-BC - Infrastructure communication [2])				x	V3.3.1
HTTPS – secure web server on PLC with CODESYS web visualization (See *Chapter 1.6.6.3.7.3.2 "Secure web server" on page 3922*)		x	x	x	V3.4.1
WebVisu for data visualisation on web-server HTML5		x	x	x	V3.4.1
FTPS – secure FTP (See *Chapter 1.6.6.3.7.3.3 "Secure FTP" on page 3923*)		x	x	x	V3.4.1
Secure online access with driver 3S UDP BlkDrvUdp	x	x	x	x	V3.4.1
Secure online access with driver 3S TCP/IP BlkDrvTcp	x	x	x	x	V3.4.1
ICMP – ping out of user project with POU ETHx_ICMP_PING or EthIcmpPing (PLCopen style)	x	x	x	x	V3.4.1
Modbus TCP client (master) with POU ETHx_MOD_MAST or ModTcpMast (PLCopen style)	x	x	x	x	V3.4.1

Description	PM5012 -x-ETH	PM5032 -x-ETH	PM5052 -x-ETH	PM507 2-T-2ETH	≥ CPU firm-ware
RTV (Remote Target Visualization)	x	x	x	x	V3.4.1
Remarks: [1]): in preparation [2]): feature is licensed					

Default open Ethernet ports of PM50xx-x-xETH

After startup without a PLC project the AC500-eCo V3 PM50xx-x-xETH contains the following Ethernet ports and sockets:

Protocol	Port
ABB NetConfig [1])	UDP 24576
Online access with driver 3S Tcp/Ip BlkDrvTcp (no scan)	TCP 11740
OPC UA server [2])	TCP 4840
Remarks: [1]): The port 24576 for ABB NetConfig protocol can be disabled via PLC configuration by deleting the protocol node from configuration tree of Ethernet interfaces ETH1 and ETH2. [2]): The port 4840 for OPC UA server is closed by default as of System FW V3.1.0.	

All other ports are closed by default.

Overview of protocols, sockets and ports

Protocol	Port	Sockets
ABB netConfig	24576	1 permanent socket per interface
3S gateway client (e.g. CODESYS) to gateway server	1217	1 permanent socket
Online access with driver 3S UDP BlkDrvUdp (with scan)	1740	1 socket per connection + 4 listen
Online access with driver 3S block driver TCP/IP (no scan)	11740	1 socket per connection + 1 listen
Modbus TCP server	502 or config-urable	1 socket listen + 1 socket per server connection, number of server connections is configu-rable in AB
Modbus TCP client with POU *ETHx_MOD_MAST*	Random	1 socket per connection with POU *ETHx_MOD_MAST*
UDP out of user program with library SysLibSockets.lib	1 ... 65535	1 socket per connection
TCP/IP out of user program with library SysLibSockets.lib	1 ... 65535	1 socket per connection
Web server on PLC with web visualization	80	1 listen and 1 per connection
SNTP client	123	1 permanent socket

Protocol	Port	Sockets
IEC60870-5-104 substation	2404	1 per connection
FTP server (See ↳ *Chapter 1.6.6.3.5.1 “Configuration of FTP server” on page 3917*)	Command port = 21 Data active mode = 20 Data passive mode = random	1 per session, max. 4 allowed
CODESYS network variables	1202	(UDP broadcast)
OPC DA server (default 3S block driver)	UDP = 1740 or TCP/IP =11740	1 socket per connection
OPC UA server	4840	1 permanent socket
ICMP – ping out of user project with POU ETHx_ICMP_PING DHCP	none	No socket
DHCP	67	1 socket during startup
SNTP (Simple Network Time Protocol) client system solution (See ↳ *Chapter 1.6.6.3.4.2.1 “(S)NTP client configuration” on page 3913*)	123	1 permanent socket
SNTP (Simple Network Time Protocol) server system solution (See ↳ *Chapter 1.6.6.3.4.2.2 “(S)NTP server configuration” on page 3916*)	123	1 permanent socket
HTTPS – secure web server on PLC with CODESYS web visualization (See ↳ *Chapter 1.6.6.3.7.3.2 “Secure web server” on page 3922*) Not for PM5012-x-ETH!	443	1 listen and 1 per connection
FTPS – secure FTP (See ↳ *Chapter 1.6.6.3.7.3.3 “Secure FTP” on page 3923*) Not for PM5012-x-ETH!	Command port = 21 Data active mode = 20 Data passive mode = random	1 per session, max. 4 allowed
Secure online access with driver 3S UDP BlkDrvUdp	1740	1 socket per connection + 1 listen
Secure online access with driver 3S TCP/IP BlkDrvTcp	11740	1 socket per connection + 1 listen
ICMP – ping out of user project with POU *ETHx_ICMP_PING* or EthIcmpPing (PLCopen style)	None	No socket
Modbus TCP client (master) with POU *ETHx_MOD_MAST* or *ModTcpMast* (PLCopen style)	Random	1 socket per connection with POU *ETHx_MOD_MAST* or *ModTcpMast*

Limitation of connections per protocol

AC500-eCo V3 processor modules

Protocol	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH	≥ CPU firmware
Modbus TCP server (e.g. for SCADA access)	3	8	10	15	3.4.1
Modbus TCP client with POU *ETHx_MOD_MAST*	8	13	20	30	3.4.1
Modbus TCP client with POU *ETHx_MOD_MAST* or *ModTcpMast* (PLCopen style)	8	13	20	30	3.4.1
IEC60870-5-104 control station incl. 2nd connection and 2nd port					
IEC60870-5-104 substation incl. 2nd port				5	3.4.1
IEC60870-5-104: No. of free tags + additional license for extension [1])				1.000	3.4.1
FTP server		2	2	2	3.4.1
Online access with driver 3S UDP BlkDrvUdp	4	4	4	6	3.4.1
Online access with driver 3S TCP/IP BlkDrvTcp	4	4	4	6	3.4.1
OPC DA server (number of connections)	4	4	4	6	3.4.1
OPC UA server (number of connections)		5	5	10	3.4.1
No. of free tags + additional license for extension [1])		125	250	1.000	3.4.1
min sampling rate (limit)		1000 ms	1000 ms	500 ms	3.4.1
Secure online access with driver 3S UDP BlkDrvUdp	4	4	4	6	3.4.1
Secure online access with driver 3S TCP/IP BlkDrvTcp	4	4	4	6	3.4.1
FTPS - secure FTP server		2	2	2	3.4.1
RTV (Remote Target Visualization)	4	4	4	6	3.4.1
Remarks: [1]): in preparation					

Ethernet protocols and ports for AC500 V3 products

Supported as of Automation Builder V 2.1

Description	PM5630-2ETH	PM5650-2ETH	PM5670-2ETH	PM5675-2ETH	≥ CPU firmware
ABB netConfig	x	x	x	x	V3.0.0
Online access with driver 3S TCP/IP BlkDrvTcp	x	x	x	x	V3.0.0
Modbus TCP server	x	x	x	x	V3.0.3
Modbus TCP client with POU ETHx_MOD_MAST	x	x	x	x	V3.0.1

Description	PM5630 -2ETH	PM5650 -2ETH	PM5670 -2ETH	PM567 5-2ETH	≥ CPU firm- ware
UDP out of user program with library netBaseService.lib	x	x	x	x	V3.0.0
UDP data exchange, Network variables	x	x	x	x	V3.0.0
TCP/IP out of user program with library netBaseService.lib	x	x	x	x	V3.0.0
Web server on PLC with web visualization	x	x	x	x	V3.0.0
NTP/SNTP ((Simple) Network Time Protocol) client with 3S licenced store package SNTPService.package. Library container: SNTPService	x	x	x	x	V3.0.0
IEC60870-5-104 control station incl. 2nd connection and 2nd port	x	x	x	x	V3.0.0
IEC60870-5-104 substation incl. 2nd port	x	x	x	x	V3.0.0
FTP server (See *Chapter 1.6.6.3.5.1 "Configuration of FTP server" on page 3917*)	x	x	x	x	V3.0.0
CODESYS network variables	x	x	x	x	V3.0.0
OPC DA server	x	x	x	x	V3.0.0
OPC UA server	x	x	x	x	V3.0.0
ICMP – ping out of user project with POU *ETHx_ICMP_PING*	x	x	x	x	V3.0.0
DHCP client	x	x	x	x	V3.1.0
NTP/SNTP ((Simple) Network Time Protocol) client system solution (See *Chapter 1.6.6.3.4.2.1 "(S)NTP client configuration" on page 3913*)	x	x	x	x	V3.1.0
NTP/SNTP ((Simple) Network Time Protocol) server system solution (See *Chapter 1.6.6.3.4.2.2 "(S)NTP server configuration" on page 3916*)	x	x	x	x	V3.1.0
Maximum number of Input/output allowed variable on Ethernet for the protocol	2 kB /2 kB	4 kB /4 kB	5 kB /5 kB	5 kB /5 kB	V3.4.0
IEC 61850 (MMS server, GOOSE) [2])	x	x	x	x	V3.1.0
EthernetIP Scanner [1, 2])	x	x	x	x	AB 2.4.1/ FW 3.4.1
EthernetIP Adapter [1, 2])	x	x	x	x	AB 2.4.1/ FW 3.4.1
KNX - Building communication [2])	x	x	x	x	V3.2.x
BACnet-BC - Infrastructure communication [2])	x	x	x	x	V3.3.1

Description	PM5630 -2ETH	PM5650 -2ETH	PM5670 -2ETH	PM567 5-2ETH	≥ CPU firm- ware
HTTPS – secure web server on PLC with CODESYS web visualization (See *Chapter 1.6.6.3.7.3.2 "Secure web server" on page 3922*)	x	x	x	x	V3.1.0
WebVisu for data visualisation on web server HTML5	x	x	x	x	V3.0.0
FTPS – secure FTP (See *Chapter 1.6.6.3.7.3.3 "Secure FTP" on page 3923*)	x	x	x	x	V3.1.0
Secure online access with driver 3S UDP BlkDrvUdp	x	x	x	x	V3.1.0
Secure online access with driver 3S TCP/IP BlkDrvTcp	x	x	x	x	V3.1.0
ICMP – ping out of user project with POU ETHx_ICMP_PING or EthIcmpPing (PLCopen style)	x	x	x	x	V3.1.0
Modbus TCP client (master) with POU ETHx_MOD_MAST or ModTcpMast (PLCopen style)	x	x	x	x	V3.1.0
RTV (Remote Target Visualization)	x	x	x	x	V3.1.0
Remarks: [1]): in preparation [2]): feature is licensed					

Default open Ethernet ports of PM56xx-2ETH

After startup without a PLC project the PM56xx-2ETH contains the following Ethernet ports and sockets:

Protocol	Port
ABB NetConfig [1])	UDP 24576
Online access with driver 3S UDP BlkDrvUdp (with scan)	UDP 1740
Online access with driver 3S Tcp/Ip BlkDrvTcp (no scan)	TCP 11740
OPC UA server [2])	TCP 4840
Remarks: [1]): The port 24576 for ABB NetConfig protocol can be disabled via PLC configuration by deleting the protocol node from configuration tree of Ethernet interfaces ETH1 and ETH2. [2]): The port 4840 for OPC UA server is closed by default as of SystemFW V3.1.0.	

All other ports are closed by default.

Overview of protocols, sockets and ports

Protocol	Port	Sockets
ABB netConfig	24576	1 permanent socket per interface
3S gateway client (e.g. CODESYS) to gateway server	1217	1 permanent socket
Online access with driver 3S UDP BlkDrvUdp (with scan)	1740	1 socket per connection + 4 listen
Online access with driver 3S block driver TCP/IP (no scan)	11740	1 socket per connection + 1 listen
Modbus TCP server	502 or configurable	1 socket listen + 1 socket per server connection, number of server connections is configurable in AB
Modbus TCP client with POU *ETHx_MOD_MAST*	Random	1 socket per connection with POU *ETHx_MOD_MAST*
UDP out of user program with library SysLibSockets.lib	1 ... 65535	1 socket per connection
TCP/IP out of user program with library SysLibSockets.lib	1 ... 65535	1 socket per connection
Web server on PLC with web visualization	80	1 listen and 1 per connection
NTP/SNTP client	123	1 permanent socket
IEC60870-5-104 control station	Random	1 per connection
IEC60870-5-104 substation	2404	1 per connection
FTP server (See *Chapter 1.6.6.3.5.1 “Configuration of FTP server” on page 3917*)	Command port = 21 Data active mode = 20 Data passive mode = random	1 per session, max. 4 allowed
CODESYS network variables	1202	(UDP broadcast)
OPC DA server (default 3S block driver)	UDP = 1740 or TCP/IP =11740	1 socket per connection
OPC UA server	4840	1 permanent socket
ICMP – ping out of user project with POU ETHx_ICMP_PING DHCP	none	No socket
DHCP	67	1 socket during startup
NTP/SNTP ((Simple) Network Time Protocol) client system solution (See *Chapter 1.6.6.3.4.2.1 “(S)NTP client configuration” on page 3913*)	123	1 permanent socket
NTP/SNTP ((Simple) Network Time Protocol) server system solution (See *Chapter 1.6.6.3.4.2.2 “(S)NTP server configuration” on page 3916*)	123	1 permanent socket

Protocol	Port	Sockets
HTTPS – secure web server on PLC with CODESYS web visualization (See *Chapter 1.6.6.3.7.3.2 "Secure web server" on page 3922*)	443	1 listen and 1 per connection
FTPS – secure FTP (See *Chapter 1.6.6.3.7.3.3 "Secure FTP" on page 3923*)	Command port = 21 Data active mode = 20 Data passive mode = random	1 per session, max. 4 allowed
Secure online access with driver 3S UDP BlkDrvUdp	1740	1 socket per connection + 1 listen
Secure online access with driver 3S TCP/IP BlkDrvTcp	11740	1 socket per connection + 1 listen
ICMP – ping out of user project with POU *ETHx_ICMP_PING* or EthIcmpPing (PLCopen style)	None	No socket
Modbus TCP client (master) with POU *ETHx_MOD_MAST* or *ModTcpMast* (PLCopen style)	Random	1 socket per connection with POU *ETHx_MOD_MAST* or *ModTcpMast*

Limitation of connections per protocol

Protocol	PM5630-2ETH	PM5650-2ETH	PM5670-2ETH	PM5675-2ETH	≥ CPU firmware
Modbus TCP server (e.g. for SCADA access)	30 40 15	100 40 25	100 40 50	100 40 50	3.0.3 3.1.0 3.1.3
Modbus TCP client with POU *ETHx_MOD_MAST*	n/a 40 30	100 40 50	n/a 40 120	n/a 40 120	3.0.1 3.1.0 3.1.3
Modbus TCP client with POU *ETHx_MOD_MAST* or *ModTcpMast* (PLCopen style)	30 30	100 50	100 120	100 120	3.1.0 3.1.3
IEC60870-5-104 control station incl. 2nd connection and 2nd port	10 5	10 10	10 20	10 20	3.1.0 3.4.0
IEC60870-5-104 substation incl. 2nd port	10 5	10 10	10 20	10 20	3.1.0 3.4.0
IEC60870-5-104: No. of free tags + additional license for extension [1])	1.000	5.000	10.000	10.000	3.4.0
FTP server	4	4	4	4	3.1.0
Online access with driver 3S UDP BlkDrvUdp	n/a 8	4 8	n/a 8	n/a 8	3.0.0 3.1.0

Protocol		PM5630-2ETH	PM5650-2ETH	PM5670-2ETH	PM5675-2ETH	≥ CPU firmware
Online access with driver 3S TCP/IP BlkDrvTcp		n/a 8	4 8	n/a 8	n/a 8	3.0.0 3.1.0
OPC DA server (number of connections)		n/a 8	4 8	n/a 8	n/a 8	3.0.0 3.1.0
OPC UA server (number of connections)		50 10	50 20	50 50	50 50	3.1.0 3.4.0
	No. of free tags + additional license for extension [1])	1.000	5.000	30.000	30.000	3.4.0
	min sampling rate (limit)	500 ms	100 ms	50 ms	50 ms	3.4.0
Secure online access with driver 3S UDP BlkDrvUdp		8	8	8	8	3.1.0
Secure online access with driver 3S TCP/IP BlkDrvTcp		8	8	8	8	3.1.0
FTPS - secure FTP server		4	4	4	4	3.1.0
RTV (Remote Target Visualization)		5	5	5	5	3.1.0
Remarks: [1]): in preparation						

The PLC types PM5630-2ETH, PM5670-2ETH and PM5675-2ETH are available as of SystemFW 3.1.0.

Ethernet configuration

Default Ethernet configuration

Module	IP Address	Netmask	Comment
PM5xx2-x-ETH	ETH: 192.168.0.10	255.255.255.0	
PM5072-T-2ETH	ETH1: 192.168.0.10 ETH2: 192.168.1.10	255.255.255.0	The Ethernet ports must be configured in different sub networks.
PM56xx-2ETH	ETH1: 192.168.0.10 ETH2: 192.168.1.10	255.255.255.0	The Ethernet ports must be configured in different sub networks.

For changing the default addresses or the description of the function keys CFG see:

Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675

Chapter 1.6.5.1.6.5 “Description of the function keys” on page 3491.

Online access

Preferred driver for online access: 3S UDP block driver BlkDrvUdp. This driver allows to scan and select the connected PLC's.

Alternative: 3S TCP/IP block driver. This driver requires at least 2 sockets:

- 1x driver "BlkDrvTcp" on port 11740
- 1x listen on port 11740 if PLC has established online connection

> *Online access can be established from:*
> – *Automation Builder command 'Login' ↳ Chapter 1.4.1.20.3.6.2 "Command 'Login'" on page 1028*
> – *CODESYS OPC DA server*
> – *Panel CP600 series*

Each established connection needs one socket. In addition one socket on port 11740 is listening.

1. Startup the PLC.
   ⇨ One socket on port 11740 (listen).
2. Login from Automation Builder via driver "BlKDrvTcp".
   ⇨ 2 sockets on port 11740 (1x online, 1x listen)
3. Additional login out of OPC server with the same driver.
   ⇨ 3 sockets on port 11740 (2x online, 1x listen)
4. Additional connect CP600 via driver "BlkDrvTcp".
   ⇨ 4 sockets on port 11740 (3x online, 1x listen)

SNTP client and server

As of version 3.1.0 the SystemFW provides a SNTP Protocol implementation which can be used for time synchronization of PLC clock. It can be used as SNTP Client or / and SNTP Server. But only one instance of each can be executed at the same time on one PLC. See ↳ *Chapter 1.6.6.3.4.2 "Configuration of the (S)NTP protocol" on page 3913*.

> *The SNTP server is listening only on the Ethernet interface, which the protocol is configured on. It is not possible to have an SNTP server on several Ethernet interfaces.*

To read diagnosis information from the SNTP protocol within an IEC application the function block *PmSntpInfo* can be used. This Function block is part of the library *ABB_Pm_AC500.lib*. It can also be used to determine the synchronization state of the PLC clock.

Using network variables in AC500 V3

When using network variables via UDP broadcast, the default broadcast address is set to *255.255.255.255*.

This will not work on PLCs with multiple Ethernet interfaces, because of undecidable routing.

Set the broadcast address to a matching subnet broadcast address, depending on which interface should be used to send the variables into the network.

Example

- ETH1 with *IP 192.168.0.10* netmask *255.255.255.0*
- ETH2 with *IP 192.168.1.10* netmask *255.255.255.0*

If you want the network variables to be broadcast on ETH1, use broadcast address *192.168.0.255*.

Onboard CAN configuration

AC500 V3 PLCs provide the following methods for CAN integration:

- Onboard CAN interface
- CANopen master-slave arrangement (with CM598-CN as a master device)

Table 621: Differences in supported protocols

	Onboard CAN	**CM598-CN**
CANopen Manager	X	
CAN 2A/2B	X	X
J1939	X	

Onboard CAN interface is not available on AC500-eCo V3!

Supported protocols

Onboard CAN interface supports the following protocols

- CANopen Manager: Connection of CI581 and CI582 without additional I/O modules
- CAN 2A/2B
- J1939

Configuration in Automation Builder is described in chapter ↳ *Chapter 1.6.6.2.11.1.1 “CM598-CAN - CANopen master communication module” on page 3737*.

Further information can be found in chapter ↳ *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

1.6.5.1.8 Hot swap

Preconditions for using hot swap

Hot swap

WARNING!

Risk of explosion or fire in hazardous environments during hot swapping!

Hot swap must not be performed in flammable environments to avoid life-threatening injury and property damage resulting from fire or explosion.

WARNING!

Electric shock due to negligent behavior during hot swapping!

To avoid electric shock

- make sure the following conditions apply:
  - Digital outputs are not under load.
  - Input/output voltages above safety extra low voltage/ protective extra low voltage (SELV/PELV) are switched off.
  - Modules are fully interlocked with the terminal unit with both snap-fits engaged before switching on loads or input/output voltage.
- Never touch exposed contacts (dangerous voltages).
- Stay away from electrical contacts to avoid arc discharge.
- Do not operate a mechanical installation improperly.

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

H = **H**ot swap

Hot swap

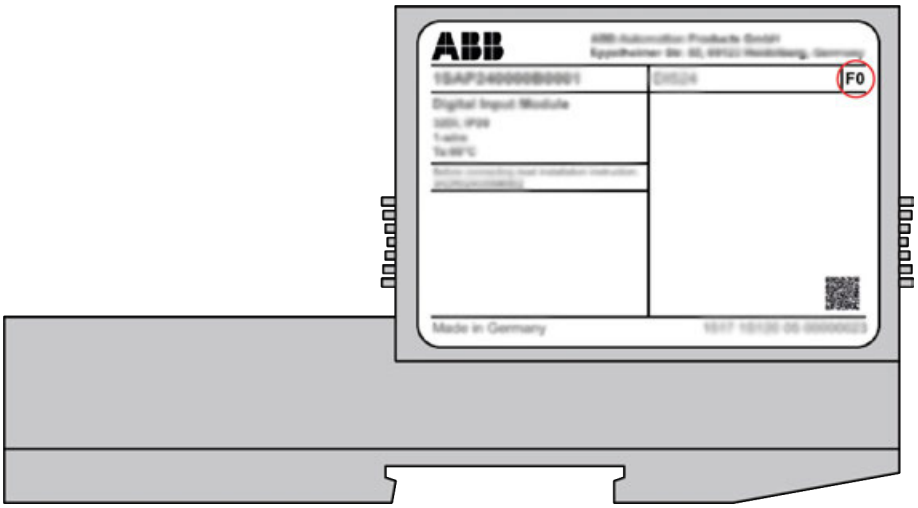
System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor modules PM56xx-2ETH with firmware version as of V3.2.0.*

Hot swap is not supported by AC500-eCo V3 CPU!



The index of the module is in the right corner of the label.

NOTICE!

Risk of damage to I/O modules!

Modules with index below F0 can be damaged when inserted or removed from the terminal unit in a powered system.

NOTICE!

Risk of damage to I/O modules!

Do not perform hot swapping if any I/O module with firmware version lower than 3.0.14 is part of the I/O configuration.

For min. required device index see table below.

Device	Min. required device index for I/O module as of FW Version 3.0.14
AC522(-XC)	F0
AI523 (-XC)	D2
AI531	D4
AI531-XC	D2
AI561	B2
AI562	B2
AI563	B3
AO523 (-XC)	D2
AO561	B2
AX521 (-XC)	D2
AX522 (-XC)	D2
AX561	B2
CD522 (-XC)	D1

Device	Min. required device index for I/O module as of FW Version 3.0.14
DA501 (-XC)	D2
DA502 (-XC)	F0
DC522 (-XC)	D2
DC523 (-XC)	D2
DC532 (-XC)	D2
DC561	B2
DC562	A2
DI524 (-XC)	D2
DI561	B2
DI562	B2
DI571	B2
DI572	A1
DO524 (-XC)	A3
DO526	A2
DO526-XC	A0
DO561	B2
DO562	A2
DO571	B3
DO572	B2
DO573	A1
DX522 (-XC)	D2
DX531	D2
DX561	B2
DX571	B3
FM562	A1

Compatibility of hot swap

Hot swap is not supported by AC500-eCo V3 CPU!

	Central I/O on V3 CPU
I/O module on TU5xx-H connected to I/O bus master	AC500 V3 CPU types: PM56xx-2ETH
Required version of I/O bus master	Firmware as of V3.2.0
Fieldbus master when used as remote I/O with AC500 V3	-
When used as remote I/O on third party controller (PLC or DCS)	-

Hot swap behavior

The following table describes the behavior in case of I/O attached to the AC500 CPU with firmware supporting hot swap on the I/O bus.

Hot Swap Behavior	Central I/O on V3 CPU
Start-up behavior with unplugged or damaged I/O module on hot swap terminal unit TU5xx-H	System and I/O modules attached to the CPU are starting (except unplugged or damaged module when plugged on hot swap terminal unit). As soon as the correct and operational I/O module is plugged on the terminal unit, the module is configured and ready to start. No specific setting needed.
Start-up behavior with wrong I/O module type on any terminal unit	System and I/O modules are not starting
Diagnosis of presence of hot swap terminal unit	Diagnosis using PLC browser command "io-bus desc" in Automation Builder V3. The PLC browser then provides an overview of the modules on the I/O bus including the position of hot swap terminal units in the I/O bus. In the application program this can be detected with a function block "IoModuleHotSwapInfo" (Library: AC500_Io / Function Blocks / I/O-Bus) . One instance of function block is needed per terminal unit on the I/O bus. The function block provides five outputs delivering information about slut number, hot swap capability and plugged/unplugged state of the I/O module
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	If module is pulled then diagnosis Err 9480 “Module removed from Hot Swap Terminal Unit” is generated
Diagnosis after plugging the I/O module on the hot swap terminal unit	Diagnosis Err 9480 is automatically acknowledged

1.6.5.1.9 KNX IP integration

This document describes the system aspects of AC500 V3 PLCs interface to KNX and its integration into the engineering tools.

It assumes - beneath basic AC500 and Automation Builder know-how at least basic experience and expertise in use of KNX and ETS (engineering software for KNX).

Additional information can be found:

- In the example projects and their documentation (C:\Users\Public\Documents\Automation-Builder\Examples\PS5604-KNX).
- In *ABB products and services*.

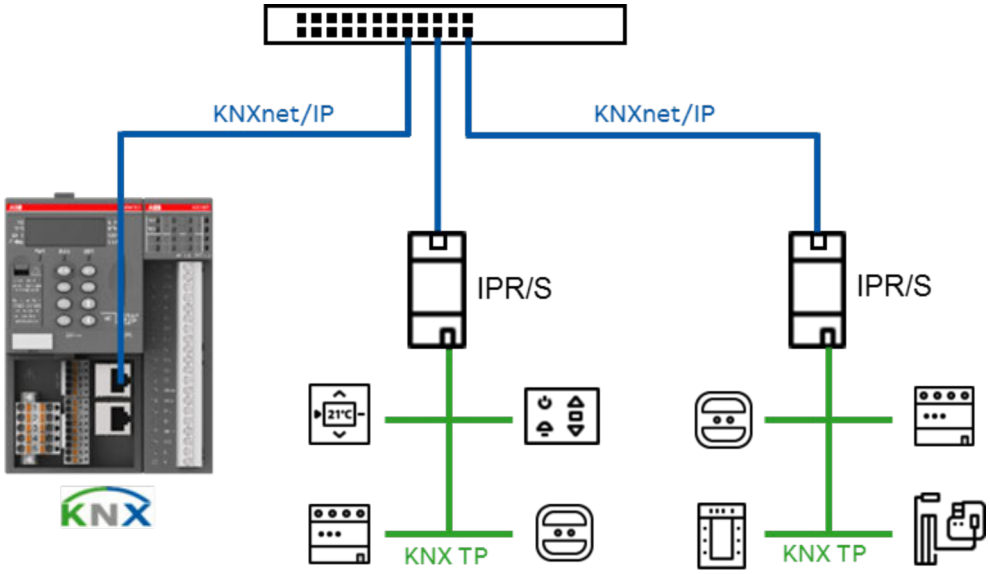
Introduction

KNX is a bus system used more on the room and floor level in buildings (e.g. for lighting, shading and local HVAC devices).

The KNX as such doesn´t necessarily need a dedicated controller for simple connection of sensor/switch to receiving/actuator devices.

The signals exchanged via the protocol are so called “group addresses” (“objects”), which are downloaded via ETS to all the thereby linked (=grouped) devices.

On the room level it typically has a serial wiring called KNX TP (twisted pair), which then is linked to floor or central building or management level via IP routers. On Ethernet it is called KNXnet/IP abbreviated also as KNX IP.



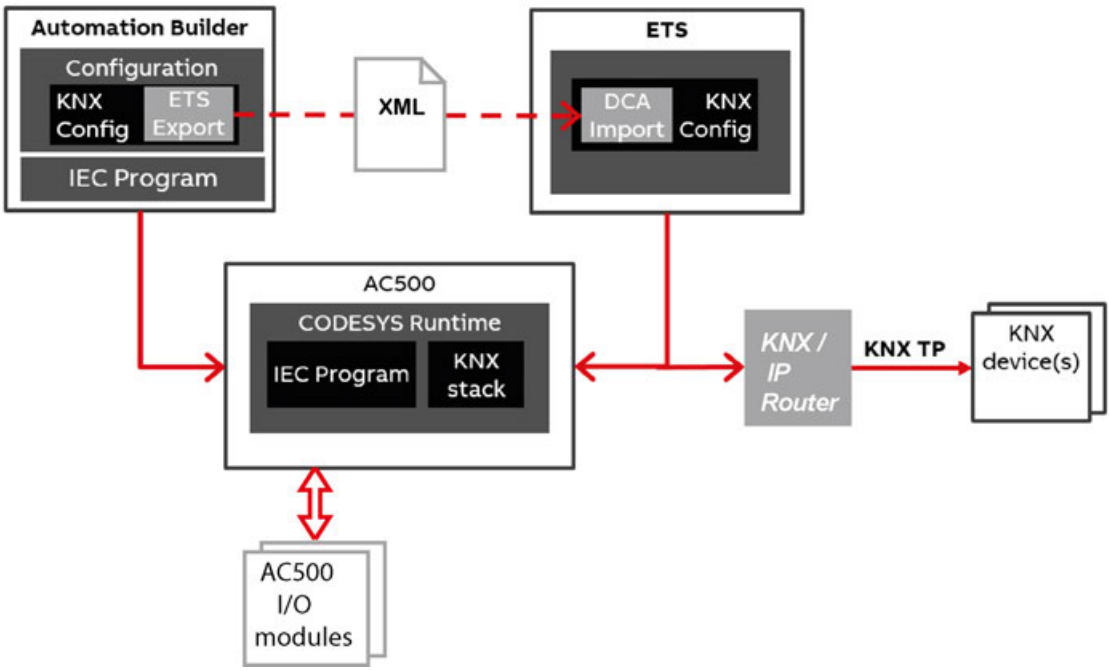
The AC500 V3 PLC is after the Automation Builder engineering step and download a standard KNX device, in which KNX communication is done via the IP network interface. It is arranged topologically on the area / main line of the KNX IP routers and communicates with them via the KNXnet/IP protocol.

Engineering workflow

Both engineering software systems for AC500 V3 PLC (Automation Builder) and KNX (ETS) are directly linked.

A data exchange for the group objects (hereinafter also called communication objects) from the Automation Builder to ETS (via an XML file) and received by a DCA (Device configuration APP) for ETS is available.

The AC500 V3 PLC is integrated into the ETS via a certified KNX “device” with the transferred group objects as configured in Automation Builder and a physical KNX address (transferred via ETS and KNX IP to the AC500 V3 PLC).



Programming and commissioning of the AC500 V3 PLC starts with Automation Builder:

Configuration of the AC500 V3 PLC, its communications, here KNX, and I/O modules and all necessary parameters.

1. Configure programmable KNX controller in CODESYS by adding group objects to the device.
2. Use group objects as inputs and outputs in the IEC application.
3. Download of the above into the AC500 V3 PLC (via the engineering interface)
4. Export of the group objects for ETS via XML file.

   The subsequent linking of the AC500 V3 PLC and the other KNX devices takes place with the vendor independent KNX commissioning software ETS:
5. Install DCA Plugin and the AC500 V3 PLC device description in ETS
6. Connect group objects in ETS and assign group addresses.
7. KNX IP download to AC500 V3 PLC.
8. The physical KNX address of the AC500 V3 PLC must be set before or during download of the KNX configuration.

The programming of the AC500 V3 PLC and the KNX commissioning can be done also by different people at different times and with same or separate engineering PCs. Both projects carry out their own download parts of their respective configurations to the AC500 V3 PLC.

The only data exchange between the two Engineering programs are the details about the KNX group objects defined in the ABB Automation Builder. This is done flexibly via the XML configuration file.

Prerequisites

- PC(s) with Windows 7 or higher with Administrator right(s)
- At least temporary network access to the internet for downloading and installing of:
  - Automation Builder as of version AB 2.1.2, (and e.g. example .project)
  - ETS5 and the necessary additional files (DCA .etsapp, device description .knxprod) plus possibly an matching example .knxproj) to above Automation Builder .project

- Network access to the local network, were the AC500 V3 PLC and KNX devices are connected.
- PS5604-KNX AC500 runtime license for each dedicated AC500 V3 PLC used in KNX networks (see *Chapter 1.6.5.1.9.6.1 “KNX runtime license” on page 3541*).
- The current IP address of the engineering PC(s) where Automation Builder and ETS are located in same Network / masked IP range, as the AC500 V3 PLC to be used.

General settings and system behavior

The KNX interface at the AC500 V3 PLC is only active during the PLC is in *RUN*.

1. Download Automation Builder program.
2. Run PLC.
3. Set physical address or download KNX application Programm via ETS.
   ⇨ The bus status can be viewed in Online View of Automation Builder.

KNX communication is only working after download the matching ETS application to the AC500 V3 PLC. Until then, the AC500 V3 PLC KNX communication is deactivated and marked with a warning symbol.

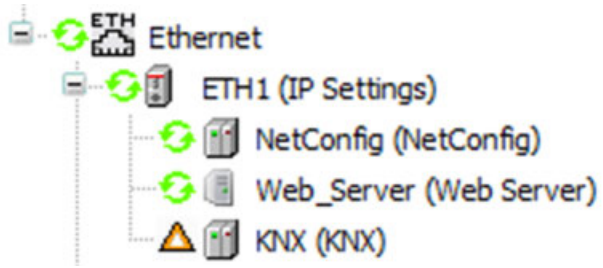


Fig. 305: KNX Interface not ready

However in this state the AC500 V3 PLC can still be switched to the KNX *programming mode* and the physical KNX address can be programmed. Also the device info can be read by ETS.

If the KNX interface is ready, this can be recognized by the green symbol on the KNX interface in the Automation Builder.

Fig. 306: KNX Interface ready

Start-up behavior

Start/Stop PLC

KNX bus works only in RUN mode.

If the PLC is in "STOP" mode the KNX bus and the outputs are reset.

To avoid this behavior in "*STOP*" mode set the following preferences at the PLC_AC500_CPU:

1. Double-click PLC_AC500_V3 <...> and click PLC Settings.
2. Enable checkbox *Update IO while in stop* and select in dropdown-menu *Behavior for Outputs in Stop "Keep current values"*.

Warm start / Cold start

If the PLC is reset also the connected objects will be reset on the KNX bus.

Power ON/Off

After Power ON the KNX Interface need approximately 1 s to start after the PLC program had started. During this period no inputs will be recognized by the PLC and no outputs will be send to the bus.

Engineering of KNX in Automation Builder

Creation of KNX group objects

Attention
This information refers to Automation Builder as of version 2.2.0.

The data exchange with the KNX bus is done via KNX group objects.

1. Double-click node *"KNX"* in the device tree *"click General ➔ click Add"*.
   ⇨ The window *Communication object* appears.
2. Enter your properties:
   - Group Object Number:
     The number of the KNX Group Object must match within the controller. It is displayed in the ETS and influences the display order in the ETS and the Automation Builder.
   - Type:
     Selection of the communication direction.
     – Input means that the controller receives values from the KNX bus.
     – Output means that the controller sends values to the KNX bus.
   - Data Point Type:
     Specification of the KNX data point type (DPT) of the Group Object. This determines the memory size, scaling and unit. For further information see the KNX Standard.
   - Group Object Name:
     The name of the KNX Group Object. It is freely selectable and is displayed in the ETS under the field name.
   - Group Object Function:
     The name of the function of the Group Object. It is freely selectable and is displayed in the ETS under the field Function.

Based on this selection, the flags of the KNX Group Object are set accordingly in the ETS.

You can use the *[Export CSV...]* button in the *"General"* tab menu bar to display the list of KNX group objects in a spreadsheet program such as Excel and edit and extend it flexibly. Then you can import them again via *[Import CSV]*.

After you have created all the required KNX group objects, export them using the *[Export to ETS]* button. This exported file contains the configuration of the KNX group objects of the AC500 V3 PLC and is imported by ETS for linking to other KNX devices. If you have not yet created project information under main menu *"Project ➔ Project Information"*, the default values will be used during the export.

To use these KNX group objects in your application program, you must assign them with IEC61131-3 variables. This additional abstraction layer of an additional variable allows you to create modular automation programs that are independent of the used bus system or input / output modules.

The assignment is possible either via the parameter page *“KNX I/O Mapping”* or *“I/O mapping list”*. Both editors offer the same function in different representations.

On the KNX I/O Mapping page, the KNX variables are shown hierarchically. Each KNX Group Object consists of several channels with additional information. These differ depending on whether it is an input or an output.

The view is structured as follows:

- Variable:
  Enter the name of the IEC 61131-3 variable that you want to assign to this channel (KNX Group Object).
- Mapping:
  Shows if the channel is already linked
- Channel:
  Name of the Channel (Channel name)
- Address:
  The memory address under which the information is stored in the memory of the AC500 V3 PLC. Inputs start with %I and outputs start with %Q.
- Type:
  Specification of the IEC 61131-3 variable type
- Default Value:
  The value used after starting the controller.
  – At a KNX Group Object input, this value is used by the automation program until a value has been received from the KNX bus.
  – At a KNX Group Object output, this value is sent to the bus when the controller is started.
- Unit:
  Specification of the KNX data point type (DPT)
- Description:
  Note text

A KNX Group Object “input” consists of a status and a control part:

The Channel name of the status part consists of: Object Number + Object Name + Object Function and include the following informations:

- UpdateFlag:
  This status flag is set to the value *“true”* for one cycle as soon as a new KNX telegram has been received. Even if the value of the telegram does not differ from the previous one.
- ValueChanged:
  This status flag is set to the value *“true”* as soon as a new KNX telegram has been received and the value differs from the previous one.
- ValueValid:
  This status flag is set to the value *“true”* as soon as a KNX telegram has been received for the first time after the controller has been started.
- WatchdogTimeout:
  As of Automation Builder 2.2.1 it will be possible to define a Watchdog Timeout for each input object. If a timeout occur this flag will be set to the value *“true”* for one cycle.
- Value:
  The current value of the KNX Group Object received from the KNX bus.

The Channel name of the control part consists of: “Control” + Object Number + Object Name + Object Function and include the following control possibilities:

- Reset status flags:
  When this flag is set from *“false”* to *“true”* by the automation program then the above-mentioned status flags of the KNX Group Object are reset to the value *“false”*.
- Send value read:
  When this flag is set from *“false”* to *“true”* by the automation program, a ValueRead telegram is sent to the KNX bus. This causes the KNX remote device to send back its current value.

A KNX Group Object “output” is represented as follows:

The Channel name of the Group Object consists of: Object Number + Object Name + Object Function

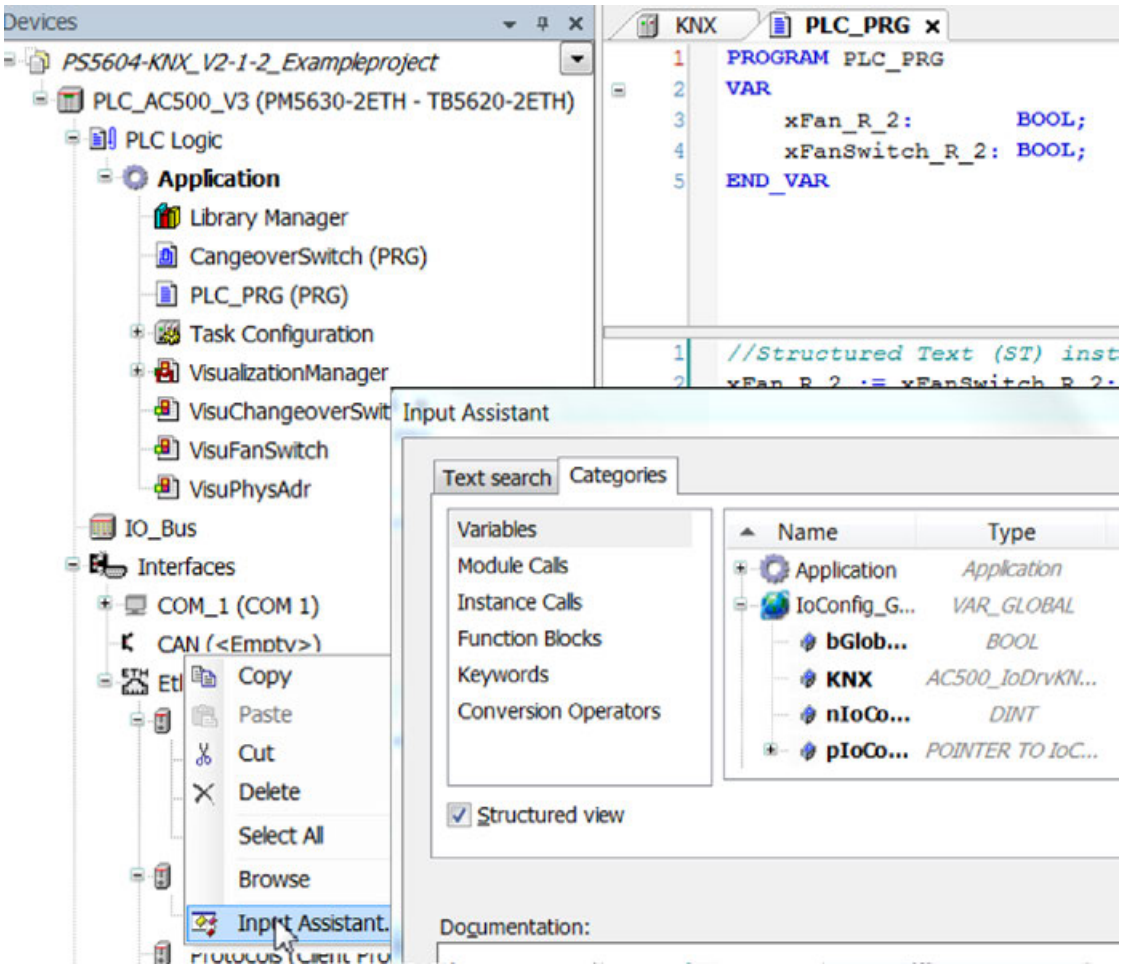
- Trigger Output:
  When this flag is set from *“false”* to *“true”* by the automation program, the current value is sent immediately to the KNX bus. The sending conditions that are may be activated in the ETS (send on change and cyclic sending) will be restarted
- Disable Output:
  As long this flag is set to *“true”* by the automation program, the sending conditions send on change and cyclic sending in the ETS are deactivated.
- Value:
  The current value of the KNX Group Object that is sent to the KNX bus.

The permanently defined Program LED Status represent the function as known in other KNX devices, showing the status of the programming LED.

Create an application program

The KNX variables defined on the KNX I/O Mapping page are available programwide under IoConfig_Globals_Mapping.

These you can see if you click in to the programming window and either via right-click select *“Input Assistant”* or press *F2*.



Export XML file

To exchange the configured KNX group objects the configuration has to be exported via XML file.

If later both projects (from Automation Builder and ETS) are loaded on the PLC, the PLC checks if the two projects have the same source and fit together. This will be done by an automatically calculated Checksum. For calculating the Checksum the following information's from the Project information will be used:

- Company
- Title
- Version
- Timestamp

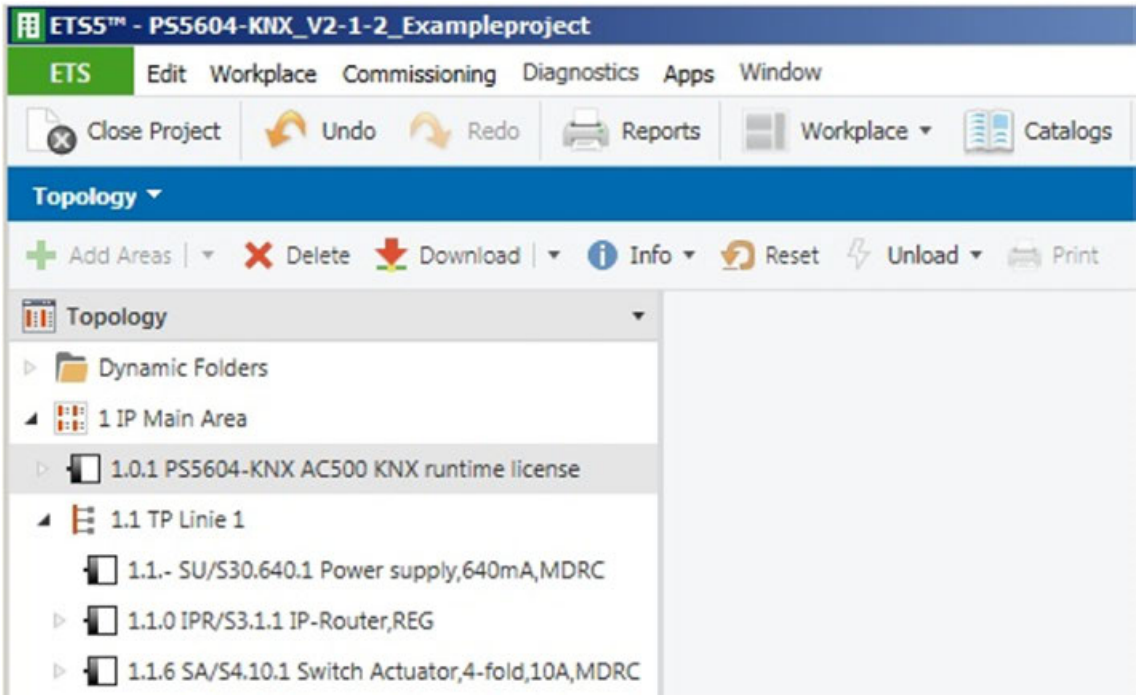
This Information will also be shown in the ETS after loading the XML file. If the user has not entered any project information some default values will be set.

Integration of the PLC in KNX

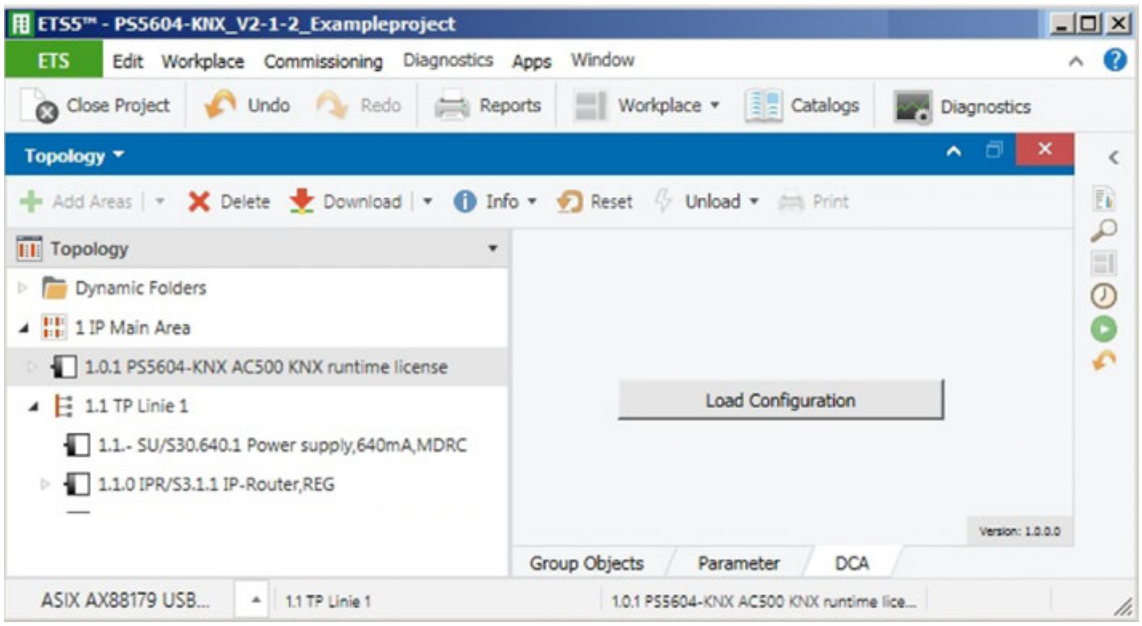
Insert controller

1. Start the ETS and insert the PS5604-KNX AC500 as controller from the ETS device catalog into your ETS project.
2. Assign a physical KNX address to the controller.

   ⇨ The controller is placed topologically on the IP *Main Area*.



Import configuration



1. Select the *"PS5604-KNX AC500"* in the ETS explorer tree and click on *"DCA"* tab in the editor window.
2. Click on Load Configuration and select the configuration XML file.
   ⇨ The KNX group objects defined in AC500 V3 PLC in Automation Builder are displayed in the ETS.

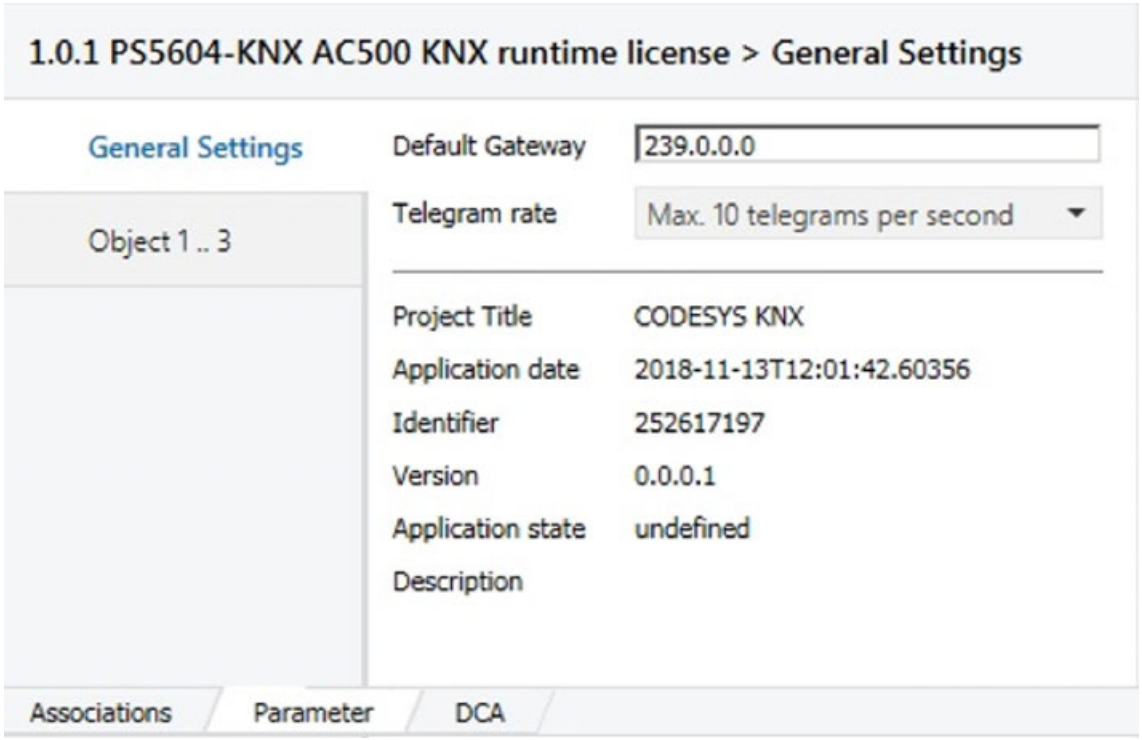
Connect controller with KNX Devices

1. Right-click on a *"PS5604-KNX AC500"* group object and assign a KNX group address or drag and drop from group address window.
2. Interlink group objects by assigning the same KNX Group Address.

Parameters of the device

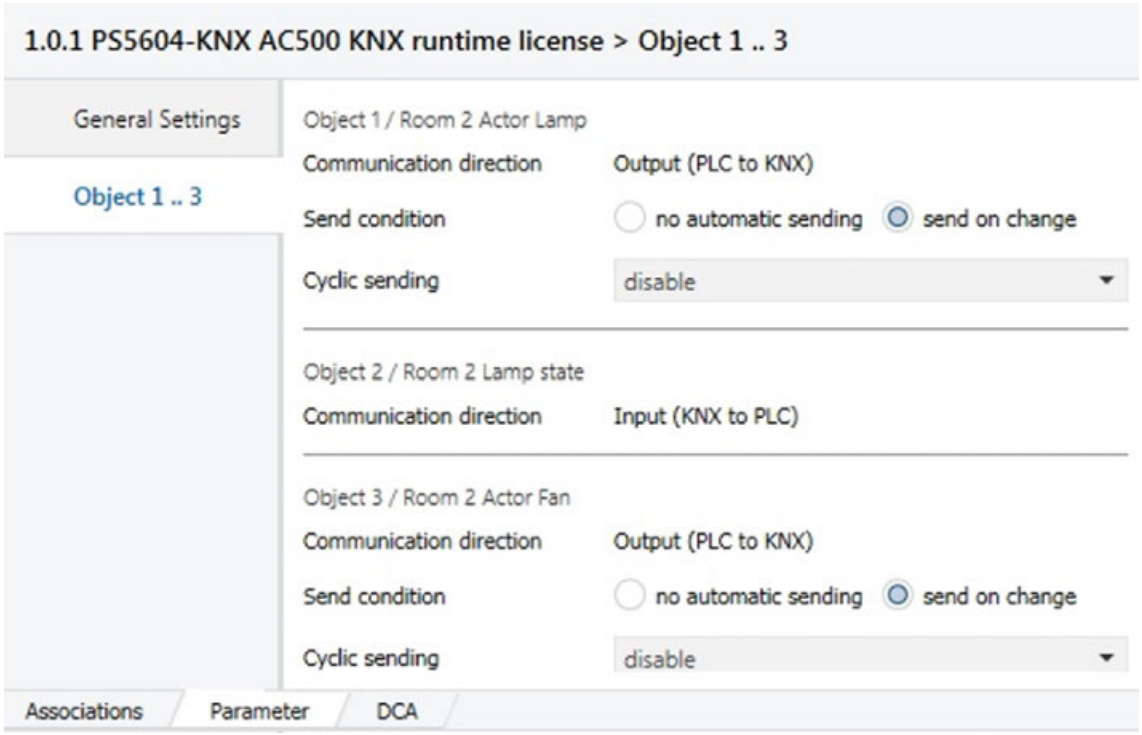
The following settings are possible in the ETS parameters of the *PS5604-KNX AC500*.

Tab *General settings*



- Default Gateway:
  The used KNXnet/IP broadcast address. This must match the KNX system (KNX IP router). It is the default setting that is usually not changed.
  - The entry field *Default Gateway* can contain another IP address for the Multicast communication. The normal Multicast IP address for KNX ist 224.0.23.12.
  - If another Multicast IP address is to be used, it can be chosen in the area from 239.0.0.0 to 239.255.255.255. This alternative Multicast IP address can be defined in the input field *Default Gateway*.
- Telegram rate:
  The maximum transmission rate of the AC500 V3 PLC can be limited in order to prevent an excessive bus load and thus to avoid malfunction of the KNX system.
  The KNX telegrams are buffered until they have been sent. New values which have been calculated by the automation program in the meantime are updating the cached values. The old cached value is discarded and not sent.
- Project Information:
  At this point, the project information of the Automation Builder project is displayed.

Tab *Object 1 .. 3*



For each KNX Group Object of the AC500 V3 PLC an Object entry is displayed in the device parameters. This is named after the number of the KNX Group Object.

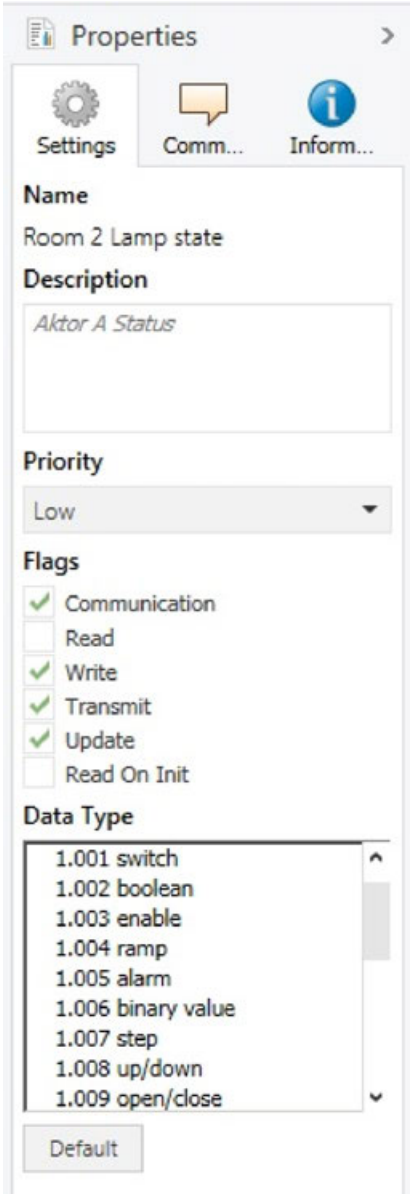
For outputs (controller sends to the KNX bus) the KNX transmission conditions can be set:

- Communication direction:
  Setting of the transmission direction of the object.
  – Input (KNX to PLC): The Controller receives values from the KNX bus.
  – Output (PLC to KNX): The Controller sends values to the KNX bus.
- Send condition (only for outputs):
  Setting whether the Controller sends a telegram to the KNX bus automatically when the object value is changed. The following options are available
  – No automatic sending:
    No automatic sending to the KNX bus. This must be done via the program code by the Trigger Output flag.
  – Send on change:
    Every time the object value changes, a telegram is sent to the KNX bus. No matter how minor this change is.
  – Send on difference (only for group objects which are not DPT 1.* Boolean): Every time the object value changes, this value is only sent to the KNX bus if it differs from the last sent value at least by the settable difference.
- Sending difference (only if Send on difference is active):
  Input of the difference by which the object value must change to be send. You can enter numbers with decimal places.
- Cyclic sending (only for outputs):
  Setting whether in addition the object value is sent cyclically repeatedly to the bus. This also happens if this object value has not changed. Two different range of values for the cycle time can be specified.
- Cycle time (only when Cyclic sending is active):
  Specification of the cycle time for the cyclic transmission.
  Input format:
  hour:minute:second
  Note: The cycle time of the KNX stack depends on the cycle time of the task that executes the stack. A long task time causes long download times from ETS. Consider the CPU load and cycle times of other processes running on the CPU when selecting a cycle time for the KNX stack.

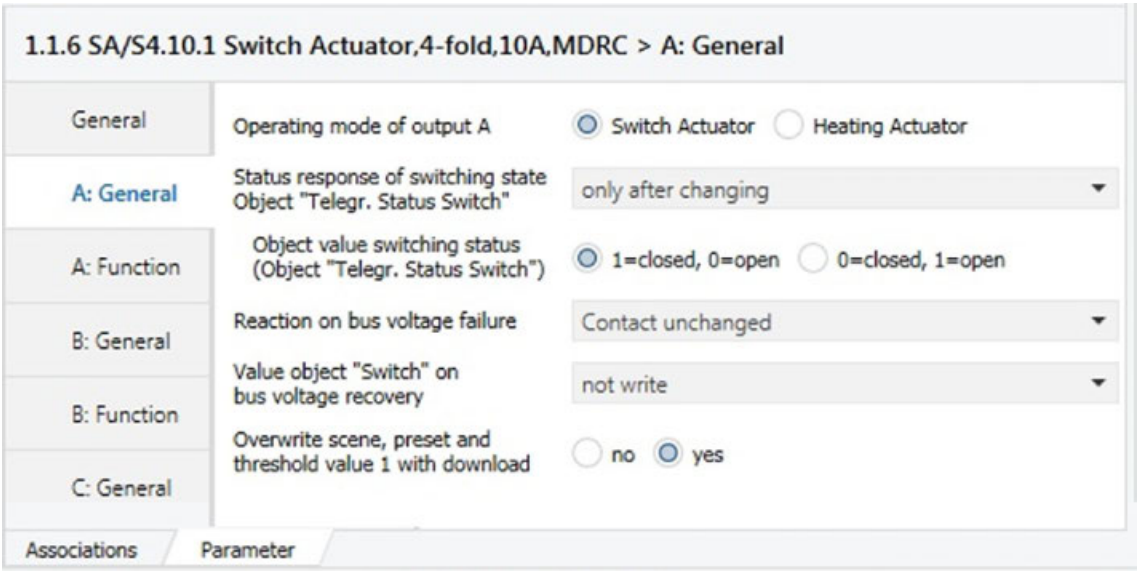
Regardless of the set transmission conditions, the program code can trigger by the flag **Trigger Output** a sending of the value to the KNX bus at any time.

By activating the flag **Read on Init** of the KNX group objects in the right ETS properties panel, the Controller sends a value read query to the connected KNX device at startup. This then responds with its current object value.

In this properties panel you can also select the appropriate subdata point type of the KNX Group Object. This defines the unit of the value in the KNX system. For example DPT 9.001 represents temperature in ° C.



If for example the response of an actuator state is needed for an input "Aktor A Status", this feature can be enabled in the parameter of the Switch Actuator (e.g. 1.1.6 SA/S4.10.1).



The current IP address as well as further information of the AC500 V3 PLC can be read via the ETS Device Info function. For this the physical KNX address is necessary. You can determine the address by the ETS function **Programming Mode**.

With the ETS function **Group Monitor** you can analyze the telegrams on the KNX bus. You can also use it to write/read KNX telegrams.

Download ETS configuration to controller

The download of the ETS configuration to the AC500 V3 PLC is done via the ETS function *"Download"* in the menu bar. This download happens via the KNX interface directly to the AC500 V3 PLC.

Best you select in ETS the network interface of the computer as the bus interface. Thus, a fast data exchange is possible and the data is not routed via the KNX TP bus.

At the first download, the physical KNX address of the controller is programmed. To do this, set the AC500 V3 PLC to KNX programming mode.

This can be done either via the display or functions inside the application program of the controller (e.g. connected to a Webvisu like done in the example program).

Via AC500 display

↳ *Chapter 1.6.5.1.6.5.3.2 "Configuration CPU firmware SystemFW >=V3.2.0 and DisplayFW >=V4.1" on page 3498*

> ***Attention!***
> *The activation of the KNX programming mode via the display only works with Automation Builder as of version 2.2.0.*

> *The KNX configuration of AC500-eCo V3 uses TA5130-KNXPB.*

The AC500 V3 PLC must be in RUN mode.

1. Press the ***CFG*** function key.
   ⇨ Switch is OFF (*S OFF*) is displayed.

2. Press the *Arrow Down* function key
   ⇨ *Pbut 0* is displayed.

   (*Pbut* is standing for progamming button, the 0 (or the 1) showing the status of the programming LED (0=Off; 1 =ON))
3. Press the ***CFG*** function key.
   ⇨ The display shows *Pbut 1* flashing.
4. Confirm this with the ***OK*** function key.
   ⇨ The display permanently shows *Pbut 1*. The AC500 V3 PLC is in KNX programming mode.

AC500-eCo V3 via TA5130-KNXPB

1. Ensure the TA5130-KNXPB is plugged in during power up, PLC is in RUN mode.
2. Ensure that TA5130-KNXPB is configured inside Automation Builder configuration.
3. Push the button on the TA5130-KNXPB.
   ⇨ The LED on the option board should turn ON. The AC500-eCo V3 is in KNX programming mode.

The AC500 V3 PLC automatically terminates the KNX programming mode after the programming of the physical KNX address.

Alternatively you can terminate the programming mode with *Pbut 0* by pressing the ***CFG*** function key.

For AC500-eCo V3: push the button on the TA5130-KNXPB again to terminate the programming mode, the LED should turn OFF.

You can exit the menu at any time with the ***ESC*** function key.

Via AC500 application

Please use the following variable for setting the KNX Program Button:

`AC500_IoDrvKNX.GVL.IoDrvKNXCopyChannels.ProgramButton`

The controller automatically terminates the programming mode after programming the physical address with the ETS.

The AC500 V3 PLC has then besides the Automation Builder configuration also the appropriate ETS configuration and starts its KNX communication.

Download all other linked KNX devices as well as the KNX IP routers. The ETS automatically creates the filter tables of the KNX IP routers so that the KNX telegrams are routed from the KNX TP lines to the IP line of the AC500 Controller.

Make changes

Changes can be made in the Automation Builder as well as in the ETS without the need for a change in the other software or the need for a new data exchange.

Only if changes are made to the KNX group objects in the Automation Builder, a data exchange with the ETS is again necessary. Afterwards, a download is required both in the Automation Builder and in the ETS. Only when these two configurations have been downloaded again to the AC500 V3 PLC, the KNX communication is in operation again.

The DCA detects changes to names and numbers of the KNX group objects when importing the configuration file in the ETS and keeps the already made settings and linked Group Addresses of these changed group objects.

Remarks

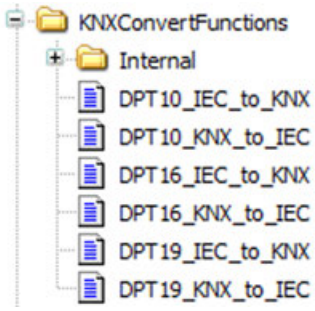
KNX runtime license

The standard V3 AC500 CPUs are shipped from the factory without firmware and need an installed PS5604-KNX runtime license for KNX operation in each CPU. The PS5604-KNX is a license document with activation code and needs to be purchased separately. The license can after a first download to one CPU also be transferred to another CPU via Automation Builder.

Data conversion

The KNX standard defines a big-endian byte order while the IEC 61131-3 is based on the little-endian byte order. Therefore, the controller automatically converts the data point types.

However, if you access the bits of the structured KNX data point types (DPT) for time, date (DPT 10.* , DPT 16.* , DPT 19.*) in your program code, you have to note the reverse byte order. Therefore, as of Automation Builder version 2.2.0, corresponding function libraries are available that provide conversion functions for these data point types.



1.6.5.1.10 Communication with Modbus RTU

Protocol description

The Modbus RTU protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

The Modbus operating mode of a serial interface is set in the PLC configuration See *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*

> *To use Modbus RTU protocol on an AC500-eCo V3 PLC, the CPU must be equipped with an option board for COMx serial communication TA5141-RS232I, TA5142-RS485 or TA5142-RS485I option board. The type of the option board adapter must be selected according to the type of physical serial interface needed.*
>
> *According to the CPU type, up to 3 option boards for COMx serial communication can be used. Following serial interface option boards can be used, see Chapter 1.6.6.2.8.3 “Attach an option board for COMx serial communication” on page 3721*

Modbus client

In this operating mode, the telegram traffic with the server(s) is handled via the function block *ModRtuMast*.

This function block sends Modbus request telegrams to the server(s) via the set interface and receives Modbus response telegrams from the server(s) via this interface.

The Modbus blocks transferred by the server contain the following information:

- Modbus address of the interrogated server (1 byte)
- Function code that defines the request of the client (1 byte)
- Data to be exchanged (n bytes)
- CRC16 control code (2 bytes)

Modbus server

In this operating mode, no function block is required for Modbus communication. Sending and receiving Modbus telegrams is performed automatically.

The AC500 CPUs process the following Modbus operation codes:

Function code		Description
DEC	**HEX**	
01 or 02	01 or 02	Read n bits
03 or 04	03 or 04	Read n words
05	05	Write one bit (encoded in one word)
06	06	Write one word
15	0F	Write n bits (encoded in one byte)
16	10	Write n words
22	16	Mask write
23	17	Read/write multiple words in one telegram

The following restrictions apply to the length of the data to be sent:

Function code		Max. length
DEC	**HEX**	
01 or 02	01 or 02	2000 bits
03 or 04	03 or 04	125 words / 62 double words
05	05	1 bit
06	06	1 word
15	0F	2000 bits
16	10	123 words / 61 double words
22	16	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 121 words / 60 double words

Technical data

The Modbus operating mode and the interface parameters are set in the ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*.

Table 622: Description of the Modbus protocol

Parameter	Value
Supported standard	See ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*
Number of connection points	1 client Max. 1 server with RS-232 interface Max. 31 servers with RS-485
Protocol	Modbus
Operating mode	Client/server
Address	Server only
Data transmission control	CRC16
Data transmission speed	From 9,600 bits/s to 115,200 bits/s ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*
Encoding	1 start bit 8 data bits 1 or 2 stop bits 1 parity bit ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*)
Max. cable length for RS-485 on COM1 for AC500 CPU	1.200 m at 19.200 baud

Modbus addresses for AC500-eCo V3 processor modules PM50x2

Modbus address table

A range of maximum 64 kB is allowed for the access via Modbus to the addressable flag area (%M area). Thus, the complete address range 0000hex up to 7FFFhex is available for Modbus.

The availability of the segments depends on the CPU. The size of the %M area can be found in the technical data of the CPUs and in the target system settings.

Inputs and outputs cannot be directly accessed using Modbus.

Following values apply:

	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
Size of the %M area	4 kB	16 kB	16 kB	64 kB
Modbus address range (Word accesses)				
HEX	0000 … 07FF	0000 … 1FFF	0000 … 1FFF	0000 … 7FFF
DEC	0000 … 2047	0000 … 8191	0000 … 8191	0000 … 32767
Byte	%MB0 … %MB4097	%MB0 … %MB16382	%MB0 … %MB16382	%MB0 … %MB65534
Word	%MW0 … %MW2047	%MW0 … %MW8191	%MW0 … %MW8191	%MW0 … %MW32767

Modbus addresses for AC500 V3 processor modules PM56xx

Modbus address table

Table 623: Modbus addresses (word accesses)

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0000	0	%MB0	%MX0.0 ... %MX0.7	%MW0	%MD0
		%MB1	%MX1.0 ... %MX1.7		
0001	1	%MB2	%MX2.0 ... %MX2.7	%MW1	
		%MB3	%MX3.0 ... %MX3.7		
0002	2	%MB4	%MX4.0 ... %MX4.7	%MW2	%MD1
		%MB5	%MX5.0 ... %MX5.7		
0003	3	%MB6	%MX6.0 ... %MX6.7	%MW3	
		%MB7	%MX7.0 ... %MX7.7		
...					
7FFE	32766	%MB65532	%MX65532.0 ... %MX65532.7	%MW32766	%MD16383

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
		%MB65533	%MX65533.0 ... %MX65533.7		
7FFF	32767	%MB65534	%MX65534.0 ... %MX65534.7	%MW32767	
		%MB65535	%MX65535.0 ... %MX65535.7		
8000	32768	%MB65536	%MX65536.0 ... %MX65536.7	%MW32768	%MD16384
		%MB65537	%MX65537.0 ... %MX65537.7		
8001	32769	%MB65538	%MX65538.0 ... %MX65538.7	%MW32769	
		%MB65539	%MX65539.0 ... %MX65539.7		
8002	32770	%MB65540	%MX65540.0 ... %MX65540.7	%MW32770	%MD16385
		%MB65541	%MX65541.0 ... %MX65541.7		
8003	32771	%MB65542	%MX65542.0 ... %MX65542.7	%MW32771	
		%MB65543	%MX65543.0 ... %MX65543.7		
...					
FFFE	65534	%MB131068	%MX131068.0 ... %MX131068.7	%MW65534	%MD32767
		%MB131069	%MX131069.0 ... %MX131069.7		
FFFF	65535	%MB131070	%MX131070.0 ... %MX131070.7	%MW65535	
		%MB131071	%MX131071.0 ... %MX131071.7		

Table 624: Address assignment (bit accesses)

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0000	0	%MB0	%MX0.0	%MW0	%MD0
0001	1		%MX0.1		
0002	2		%MX0.2		
0003	3		%MX0.3		
0004	4		%MX0.4		
0005	5		%MX0.5		
0006	6		%MX0.6		
0007	7		%MX0.7		
0008	8	%MB1	%MX1.0		
0009	9		%MX1.1		
000A	10		%MX1.2		
000B	11		%MX1.3		
000C	12		%MX1.4		
000D	13		%MX1.5		
000E	14		%MX1.6		
000F	15		%MX1.7		
0010	16	%MB2	%MX2.0	%MW1	
0011	17		%MX2.1		
0012	18		%MX2.2		
0013	19		%MX2.3		
0014	20		%MX2.4		
0015	21		%MX2.5		
0016	22		%MX2.6		
0017	23		%MX2.7		
0018	24	%MB3	%MX3.0		
0019	25		%MX3.1		
001A	26		%MX3.2		
001B	27		%MX3.3		
001C	28		%MX3.4		
001D	29		%MX3.5		
001E	30		%MX3.6		
001F	31		%MX3.7		
0020	32	%MB4	%MX4.0	%MW2	%MD1
0021	33		%MX4.1		
0022	34		%MX4.2		
...	...	...	...	...	...
0FFF	4095	%MB511	%MX511.7	%MW255	%MD127
1000	4096	%MB512	%MX512.0	%MW256	%MD128

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
...	...	...	...	...	...
7FFF	32767	%MB4095	%MX4095.7	%MW2047	%MD1023
8000	32768	%MB4096	%MX4096.0	%MW2048	%MD1024
...	...	...	...	...	...
FFFF	65535	%MB8191	%MX8191.7	%MW4095	%MD2047

Calculation of the bit variable from the hexadecimal address:

Formula:			
	Bit variable (BOOL) := %MXBYTE.BIT		
where:	DEC	Decimal address	
	BYTE	DEC / 8	
	BIT	DEC mod 8	(Modulo division)

Examples:

- Address hexadecimal = 16#2002
  DEC := 8194
  BYTE := 8194 / 8 := 1024
  BIT := 8194 mod 8 := 2
  Bit variable: %MX1024.2
- Address hexadecimal = 16#3016
  DEC := 12310
  BYTE := 12310 / 8 := 1538,75 -> 1538
  BIT := 12310 mod 8 := 6
  Bit variable: %MX1538.6
- Address hexadecimal = 16#55AA
  DEC := 21930
  BYTE := 21930 / 8 := 2741,25 -> 2741
  BIT := 21930 mod 8 := 2
  Bit variable: %MX2741.2

Calculation of the hexadecimal address from the bit variable:

Examples:

- Bit variable := %MX515.4
  DEC := 515 * 8 + 4 := 4124
  Address hex := 16#101C
- Bit variable := %MX3.3
  DEC := 3 * 8 + 3 := 27
  Address hex := 16#001B
- Bit variable := %MX6666.2
  DEC := 6666 * 8 + 2 := 53330
  Address hex := 16#D052

Peculiarities for accessing Modbus addresses

Peculiarities for bit access:

- A WORD in the %M area is assigned to each Modbus address 0000hex .. FFFFhex.
- Bit addresses 0000hex .. FFFFhex are contained in the word range %MW0 .. %MW4095

Areas protect from read/write access by Modbus client

As described in *Chapter 1.6.6.3.3.1.1 “Configuration of Modbus TCP/IP server” on page 3910*, one write-protected and one read-protected area can be defined. If you try to write to a write-protected area or to read from a read-protected area, an exception response is generated.

Local data of the Modbus client

The address of the area from which data are to be read or to which data are to be written is specified in the function block *ModRtuMast* at input "Data", via the ADR operator.

For the AC500, the following areas can be accessed using the ADR operator:

- Inputs area (%I area)
- Outputs area (%Q area)
- Area of non-buffered variables (VAR .. END_VAR or VAR_GLOBAL END_VAR)
- Addressable flag area (also protected areas for %M area)
- Area of buffered variables (VAR RETAIN .. END_VAR or VAR_GLOBAL RETAIN .. END_VAR)

Modbus telegrams

The send and receive of telegrams shown in this section are not visible in the PLC. However, the complete telegrams can be made visible using a serial data analyzer connected to the connection line between server and client, if required.

The amount of user data depends on the capabilities of the server and the client.

For the following examples, it is assumed that one AC500 Modbus module is used as client and another one is used as server. There may be different properties if modules of other manufacturers are used.

FCT 1 or 2: Read n bits

Table 625: Client request

Server address	Function code	Server operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low

Table 626: Server response

Server address	Function code	Number of Bytes	...Data...	CRC	
				High	Low

FCT 3 or 4: Read n words

Table 627: Client request

Server address	Function code	Server operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Table 628: Server response

Server address	Function code	Number of Bytes	Data		CRC	
			High	Low	High	Low

FCT 3 or 4: Read n double words

The function code "read double word" is not defined in the Modbus RTU standard. This is why the double word is composed of a low word and a high word (depending on the manufacturer) Same tables as *Chapter 1.6.5.1.10.6.2 “FCT 3 or 4: Read n words” on page 3549*.

FCT 5: Write 1 bit

For the function code "write 1 bit", the value of the bit to be written is encoded in one word.

BIT = TRUE -> Data word = FF 00 HEX

BIT = FALSE -> Data word = 00 00 HEX

Table 629: Client request

Function code	Server operand address		Number of words		CRC	
	High	Low	High	Low	High	Low

Table 630: Server response

Function code	Server operand address		Data		CRC	
	High	Low	High	Low	High	Low

FCT 6: Write 1 word

Table 631: Server request

Server address	Function code	Server operand address		Data		CRC	
		High	Low	High	Low	High	Low

Table 632: Server response

Server address	Function code	Server operand address		Data		CRC	
		High	Low	High	Low	High	Low

FCT 15: Write n bits

Table 633: Client request

Server operand address		Number of bits		Number of bytes	...Data...	CRC	
High	Low	High	Low			High	Low

Table 634: Server response

Server address	Function code	Server operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low

FCT 16: Write n words

Table 635: Client request

Server operand address		Number of words		Number of bytes	...Data...	CRC	
High	Low	High	Low			High	Low

Table 636: Server response

Function code	Server operand address		Number of words		CRC	
	High	Low	High	Low	High	Low

FCT 16: Write n double words

The function code "write double word" is not defined in the Modbus RTU standard. This is why the double word is composed of a low word and a high word (depending on the manufacturer).

Table 637: Client request

Server operand address		Number of words		Number of bytes	...Data...	CRC	
High	Low	High	Low			High	Low

Table 638: Server response

Server address	Function code	Server operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

FCT 22: Mask write register

Table 639: Client request

Server address	Function code	Server operand address		AND Mask		OR Mask		CRC	
		High	Low	High	Low	High	Low	High	Low

Table 640: Server response

Server address	Function code	Server operand address		AND Mask		OR Mask		CRC	
		High	Low	High	Low	High	Low	High	Low

FCT 23: Read/Write n words

Table 641: Client request

Server address	Function code	Operand addr. read		Number of words read		Operand addr. write		Number of words write		Number of bytes write	...Data...	CRC	
		High	Low	High	Low	High	Low	High	Low			High	Low

Table 642: Server response

Server address	Function code	Number of bytes read	...Data...	CRC	
				High	Low

Exception response by server

In operating mode Modbus client, the AC500 does only send requests, if the parameters at the *ModRtuMast* inputs are logically correct.

Nevertheless, it can happen that a server cannot process the request of the client or that the server cannot interpret the request due to transmission errors or in case it's capabilities are exceeded in any way. In those cases, the server returns an exception response to the client. In order to identify this response as an exception response, the function code returned by the server is a logical OR interconnection of the function code received from the client and the value 80HEX.

Table 643: Server response

Server address	OR 80HEX	Error code	CRC	
			High	Low

Possible error codes of the client

Code	Description
01DEC	ILLEGAL FUNCTION The server does not support the function requested by the client
02DEC	ILLEGAL DATA ADDRESS Invalid operand address in the server or operand area exceeded
03DEC	ILLEGAL DATA VALUE At least one value is outside the permitted range of values
04DEC	SERVER DEVICE FAILURE An unrecoverable error occurred while the server was attempting to perform the requested action

Code	Description
05DEC	ACKNOWLEDGE Specialized use in conjunction with programming commands. The server has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed
06DEC	SERVER DEVICE BUSY Specialized use in conjunction with programming commands. The server is engaged in processing a long–duration program command. The client should retransmit the message later when the server is free.
07DEC	NEGATIVE ACKNOWLEDGE Specialized use in conjunction with programming commands. The server cannot perform the programming functions. Client should request diagnostic or error information from server.
08DEC	MEMORY PARITY ERROR Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server attempted to read record file, but detected a parity error in the memory. The client can retry the request, but service may be required on the server device.
10DEC	GATEWAY PATH UNAVAILABLE Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
11DEC	GATEWAY TARGET DEVICE FAILED TO RESPOND Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

Example

Table 644: Example:

Modbus request of the client:			
	Function code:	01	Read n bits
	Server operand address:	4000HEX = 16384DEC	Area for read access disabled in server

Modbus response of the server:			
	Function code:	81HEX	
	Error code:	03	

Processing bits

Some of the Modbus function codes are used to read or write bits (coils, discrete inputs). While a variable of data type WORD can be accessed easily, accessing a stream of bits is complex.

Data type *Chapter 1.4.1.19.5.10 “Data Type 'BIT'” on page 656* must not be mixed up with data type *Chapter 1.4.1.19.5.1 “Data type 'BOOL'” on page 647*. Variables of both types may have values ‘TRUE’ or ‘FALSE’. But while BIT means one single bit only, BOOL requires a byte (8 bit) of memory.

Modbus client

When accessing bits in a *Server*, the local data referred to at *Client* function blocks input data is always expected to be of format BOOL.

```
VAR
abBoolArray : ARRAY [0..15] OF BOOL;
ModMast : ModTcpMast;
END_VAR
```

```
abBoolArray[0]  := TRUE;
abBoolArray[15] := FALSE;

ModMast(Execute:= ModMastExecute,
        Eth:= ETH1,
        IPAdr:= IP_ADR_STRING_TO_DWORD('192.168.0.100'),
        UnitID:= 0,
        Fct:= 15,
        Addr:= 16#0000,
        Nb:= SIZEOF(abBoolArray),
        Data:= ADR(abBoolArray));
```

Modbus server

Using the bit offset

The simplest way to access a certain bit within a larger variable is to directly use the bit offset (0 based; see *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*).

```
VAR
awWordVariable AT %MW0 : ARRAY [0..7] OF WORD;
END_VAR
```

```
awWordVariable[0].0  := TRUE;
awWordVariable[7].15 := FALSE;
```

Defining symbolic names for the bit offsets

A more convenient way to access bits e.g. within a word is to define a symbolic name for each single offset ↳ *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*.

```
VAR_GLOBAL CONSTANT
(*bit offsets in awWordVariable[0]*)
NameBitOffset_0_00 : INT := 0;
NameBitOffset_0_01 : INT := 1;
(*...*)
NameBitOffset_0_15 : INT := 15;
(*bit offsets in awWordVariable[7]*)
NameBitOffset_7_00 : INT := 0;
NameBitOffset_7_01 : INT := 1;
(*...*)
NameBitOffset_7_15 : INT := 15;
END_VAR
```

```
VAR
awWordVariable AT %MW0 : ARRAY [0..7] OF WORD;
END_VAR
```

```
awWordVariable[0].NameBitOffset_0_00  := TRUE;
awWordVariable[7].NameBitOffset_7_15 := FALSE;
```

Defining a data type

A further alternative is to define your own data types (see *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835*) according to the requirements of your particular application (see *“Symbolic bit access in structure variables” on page 642*).

```
TYPE ControlWordType :
STRUCT
NameBitOffset_00 : BIT;
NameBitOffset_01 : BIT;
NameBitOffset_02 : BIT;
NameBitOffset_03 : BIT;
NameBitOffset_04 : BIT;
NameBitOffset_05 : BIT;
NameBitOffset_06 : BIT;
NameBitOffset_07 : BIT;
NameBitOffset_08 : BIT;
NameBitOffset_09 : BIT;
NameBitOffset_10 : BIT;
NameBitOffset_11 : BIT;
NameBitOffset_12 : BIT;
NameBitOffset_13 : BIT;
NameBitOffset_14 : BIT;
NameBitOffset_15 : BIT;
END_STRUCT
END_TYPE
```

```
VAR
atTypeVariable AT %MW0 : ARRAY [0..7] OF ControlWordType;
END_VAR
```

```
atTypeVariable[0].NameBitOffset_00 := TRUE;
atTypeVariable[7].NameBitOffset_15 := FALSE;
```

Defining a complex data type

In case your application requires some more complex data types you can combine data types (DUT; see *“Symbolic bit access in structure variables” on page 642*).

```
TYPE ControlWord0Type :
STRUCT
NameBitOffset_0_00 : BIT;
NameBitOffset_0_01 : BIT;
NameBitOffset_0_02 : BIT;
NameBitOffset_0_03 : BIT;
NameBitOffset_0_04 : BIT;
NameBitOffset_0_05 : BIT;
NameBitOffset_0_06 : BIT;
NameBitOffset_0_07 : BIT;
NameBitOffset_0_08 : BIT;
NameBitOffset_0_09 : BIT;
NameBitOffset_0_10 : BIT;
NameBitOffset_0_11 : BIT;
NameBitOffset_0_12 : BIT;
NameBitOffset_0_13 : BIT;
NameBitOffset_0_14 : BIT;
NameBitOffset_0_15 : BIT;
END_STRUCT
END_TYPE
```

ControlWord1Type

```
1  TYPE ControlWord1Type :
2  STRUCT
3  NameBitOffset_1_00 : BIT;
4  NameBitOffset_1_01 : BIT;
5  NameBitOffset_1_02 : BIT;
6  NameBitOffset_1_03 : BIT;
7  NameBitOffset_1_04 : BIT;
8  NameBitOffset_1_05 : BIT;
9  NameBitOffset_1_06 : BIT;
10 NameBitOffset_1_07 : BIT;
11 NameBitOffset_1_08 : BIT;
12 NameBitOffset_1_09 : BIT;
13 NameBitOffset_1_10 : BIT;
14 NameBitOffset_1_11 : BIT;
15 NameBitOffset_1_12 : BIT;
16 NameBitOffset_1_13 : BIT;
17 NameBitOffset_1_14 : BIT;
18 NameBitOffset_1_15 : BIT;
19 END_STRUCT
20 END_TYPE
```

```
TYPE ControlWordListType :
STRUCT
ControlWord0 : ControlWord0Type;
ControlWord1 : ControlWord1Type;
END_STRUCT
END TYPE
```

```
VAR
atWordListType AT %MW0 : ControlWordListType;
END_VAR
```

```
atWordListType.ControlWord0.NameBitOffset_0_00 := TRUE;
atWordListType.ControlWord1.NameBitOffset_1_15 := FALSE;
```

Pack/unpack BOOL variables

In case you prefer variables of type BOOL you can use the functions for packing MEM_Pack_BitsToByte and unpacking MEM_UnpackWord of the CAA_Memory.library, which can be found with the Library Manager ↳ *Chapter 1.5.3 “Library Manager functionality” on page 2146*.

```
VAR
abBoolVariable : ARRAY [0..15] OF BOOL;
wWordVariable0 : AT %MW0 WORD;
wWordVariable1 : AT %MW1 WORD;
END_VAR

VAR
Unpack  : Mem.UnpackWord;
END_VAR
```

```
abBoolVariable[0]  := TRUE;
abBoolVariable[15] := FALSE;

wWordVariable0 := Mem.PackBitsToWord(abBoolVariable[0], abBoolVariable[1], abBoolVariable[2], abBoolVariable[3],
                                     abBoolVariable[4], abBoolVariable[5], abBoolVariable[6], abBoolVariable[7],
                                     abBoolVariable[8], abBoolVariable[9], abBoolVariable[10], abBoolVariable[11],
                                     abBoolVariable[12], abBoolVariable[13], abBoolVariable[14], abBoolVariable[15]);
wWordVariable1 := 1;
Unpack(wValue:= wWordVariable1,
       xBit0=> abBoolVariable[0], xBit1=> abBoolVariable[1], xBit2=> abBoolVariable[2], xBit3=> abBoolVariable[3],
       xBit4=> abBoolVariable[4], xBit5=> abBoolVariable[5], xBit6=> abBoolVariable[6], xBit7=> abBoolVariable[7],
       xBit8=> abBoolVariable[8], xBit9=> abBoolVariable[9], xBit10=> abBoolVariable[10], xBit11=> abBoolVariable[11],
       xBit12=> abBoolVariable[12], xBit13=> abBoolVariable[13], xBit14=> abBoolVariable[14], xBit15=> abBoolVariable[15]);
```

Function block ModRtuMast

This function block is only required in the operating mode Modbus client. It handles the communication (transmission of telegrams to the servers and receipt of telegrams from the servers). The function block can be used for the local serial interfaces of the controller. A separate instance of the function block has to be used for each interface.

ModRtuMast is contained in the library *AC500_ModRtuMast*.

1.6.5.1.11 Communication with Modbus TCP/IP

Protocol description

The Modbus TCP protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Each Ethernet interface can work as Modbus client and server interface in parallel if required.

The Modbus operating mode of an Ethernet interface is set in *Modbus on TCP/IP* ↳ *Chapter 1.6.6.3.3 “Modbus protocol” on page 3910*external.

Modbus client

In this operating mode, the telegram traffic with the server(s) is handled via the function block ETHx_MOD_MAST, which can be found through the Library Manager ↳ *Chapter 1.5.3 “Library Manager functionality” on page 2146*. This function block sends Modbus request telegrams to the server(s) via the set interface and receives Modbus response telegrams from the server(s) via this interface.

The Modbus function blocks transferred by the client contain the following information:

- Transaction identifier for synchronization between messages of server and client (2 byte)
- Protocol identifier (0 for Modbus/TCP) (2 byte)
- Length field (Number of bytes in frame) (2 byte)
- Unit identifier (1 byte)
- Function code that defines the request of the client (1 byte)
- Data to be exchanged (n bytes)

Modbus server

In this operating mode, no function block is required for Modbus communication. Sending and receiving Modbus telegrams is performed automatically.

The AC500 CPUs process the following Modbus operation codes:

Function code		Description
DEC	**HEX**	
01 or 02	01 or 02	Read n bits
03 or 04	03 or 04	Read n words
05	05	Write one bit (encoded in one word)
06	06	Write one word
15	0F	Write n bits (encoded in one byte)
16	10	Write n words
22	16	Mask write
23	17	Read/write multiple words in one telegram

The following restrictions apply to the length of the data to be sent:

Function code		Max. length
DEC	**HEX**	
01 or 02	01 or 02	2000 bits
03 or 04	03 or 04	125 words / 62 double words

Function code		Max. length
DEC	**HEX**	
05	05	1 bit
06	06	1 word
15	0F	2000 bits
16	10	123 words / 61 double words
22	16	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 121 words / 60 double words

Technical data

Configuration of Modbus on TCP/IP is described in the chapter ↳ *Chapter 1.6.6.3.3 “Modbus protocol” on page 3910*.

Modbus addresses for AC500-eCo V3 processor modules PM50xx

Modbus address table

A range of maximum 64 kB is allowed for the access via Modbus to the addressable flag area (%M area). Thus, the complete address range 0000hex up to 7FFFhex is available for Modbus.

The availability of the segments depends on the CPU. The size of the %M area can be found in the technical data of the CPUs and in the target system settings.

Inputs and outputs cannot be directly accessed using Modbus.

Following values apply:

	PM5012-x-ETH	**PM5032-x-ETH**	**PM5052-x-ETH**	**PM5072-T-2ETH**
Size of the %M area	4 kB	16 kb	16 kB	64 kB
Modbus address range (Word accesses)				
HEX	0000 … 07FF	0000 … 1FFF	0000 … 1FFF	0000 … 7FFF
DEC	0000 … 2047	0000 … 8191	0000 … 8191	0000 … 32767
Byte	%MB0 … %MB4097	%MB0 … %MB16382	%MB0 … %MB16382	%MB0 … %MB65534
Word	%MW0 … %MW2047	%MW0 … %MW8191	%MW0 … %MW8191	%MW0 … %MW32767

Modbus addresses for AC500 V3 processor modules PM56xx

Modbus address table

Table 645: Modbus addresses (word accesses)

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0000	0	%MB0	%MX0.0 ... %MX0.7	%MW0	%MD0
		%MB1	%MX1.0 ... %MX1.7		
0001	1	%MB2	%MX2.0 ... %MX2.7	%MW1	
		%MB3	%MX3.0 ... %MX3.7		
0002	2	%MB4	%MX4.0 ... %MX4.7	%MW2	%MD1
		%MB5	%MX5.0 ... %MX5.7		
0003	3	%MB6	%MX6.0 ... %MX6.7	%MW3	
		%MB7	%MX7.0 ... %MX7.7		
...					
7FFE	32766	%MB65532	%MX65532.0 ... %MX65532.7	%MW32766	%MD16383
		%MB65533	%MX65533.0 ... %MX65533.7		
7FFF	32767	%MB65534	%MX65534.0 ... %MX65534.7	%MW32767	
		%MB65535	%MX65535.0 ... %MX65535.7		
8000	32768	%MB65536	%MX65536.0 ... %MX65536.7	%MW32768	%MD16384
		%MB65537	%MX65537.0 ... %MX65537.7		
8001	32769	%MB65538	%MX65538.0 ... %MX65538.7	%MW32769	
		%MB65539	%MX65539.0 ... %MX65539.7		
8002	32770	%MB65540	%MX65540.0 ... %MX65540.7	%MW32770	%MD16385

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
		%MB65541	%MX65541.0 ... %MX65541.7		
8003	32771	%MB65542	%MX65542.0 ... %MX65542.7	%MW32771	
		%MB65543	%MX65543.0 ... %MX65543.7		
...					
FFFE	65534	%MB131068	%MX131068.0 ... %MX131068.7	%MW65534	%MD32767
		%MB131069	%MX131069.0 ... %MX131069.7		
FFFF	65535	%MB131070	%MX131070.0 ... %MX131070.7	%MW65535	
		%MB131071	%MX131071.0 ... %MX131071.7		

Table 646: Address assignment (bit accesses)

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0000	0	%MB0	%MX0.0	%MW0	%MD0
0001	1		%MX0.1		
0002	2		%MX0.2		
0003	3		%MX0.3		
0004	4		%MX0.4		
0005	5		%MX0.5		
0006	6		%MX0.6		
0007	7		%MX0.7		
0008	8	%MB1	%MX1.0		
0009	9		%MX1.1		
000A	10		%MX1.2		
000B	11		%MX1.3		
000C	12		%MX1.4		
000D	13		%MX1.5		

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
000E	14		%MX1.6		
000F	15		%MX1.7		
0010	16	%MB2	%MX2.0	%MW1	
0011	17		%MX2.1		
0012	18		%MX2.2		
0013	19		%MX2.3		
0014	20		%MX2.4		
0015	21		%MX2.5		
0016	22		%MX2.6		
0017	23		%MX2.7		
0018	24	%MB3	%MX3.0		
0019	25		%MX3.1		
001A	26		%MX3.2		
001B	27		%MX3.3		
001C	28		%MX3.4		
001D	29		%MX3.5		
001E	30		%MX3.6		
001F	31		%MX3.7		
0020	32	%MB4	%MX4.0	%MW2	%MD1
0021	33		%MX4.1		
0022	34		%MX4.2		
...	...	...	...	...	...
0FFF	4095	%MB511	%MX511.7	%MW255	%MD127
1000	4096	%MB512	%MX512.0	%MW256	%MD128
...	...	...	...	...	...
7FFF	32767	%MB4095	%MX4095.7	%MW2047	%MD1023
8000	32768	%MB4096	%MX4096.0	%MW2048	%MD1024
...	...	...	...	...	...
FFFF	65535	%MB8191	%MX8191.7	%MW4095	%MD2047

Calculation of the bit variable from the hexadecimal address:

Formula:			
	Bit variable (BOOL) := %MXBYTE.BIT		
where:	DEC	Decimal address	
	BYTE	DEC / 8	
	BIT	DEC mod 8	(Modulo division)

Examples:

- Address hexadecimal = 16#2002
  DEC := 8194
  BYTE := 8194 / 8 := 1024
  BIT := 8194 mod 8 := 2
  Bit variable: %MX1024.2
- Address hexadecimal = 16#3016
  DEC := 12310
  BYTE := 12310 / 8 := 1538,75 -> 1538
  BIT := 12310 mod 8 := 6
  Bit variable: %MX1538.6
- Address hexadecimal = 16#55AA
  DEC := 21930
  BYTE := 21930 / 8 := 2741,25 -> 2741
  BIT := 21930 mod 8 := 2
  Bit variable: %MX2741.2

Calculation of the hexadecimal address from the bit variable:

Examples:

- Bit variable := %MX515.4
  DEC := 515 * 8 + 4 := 4124
  Address hex := 16#101C
- Bit variable := %MX3.3
  DEC := 3 * 8 + 3 := 27
  Address hex := 16#001B
- Bit variable := %MX6666.2
  DEC := 6666 * 8 + 2 := 53330
  Address hex := 16#D052

Peculiarities for accessing Modbus addresses

Peculiarities for bit access:

- A WORD in the %M area is assigned to each Modbus address 0000hex .. FFFFhex.
- Bit addresses 0000hex .. FFFFhex are contained in the word range %MW0 .. %MW4095

Areas protect from read/write access by Modbus client

As described in ↳ *Chapter 1.6.6.3.3.1.1 “Configuration of Modbus TCP/IP server” on page 3910*, one write-protected and one read-protected area can be defined. If you try to write to a write-protected area or to read from a read-protected area, an exception response is generated.

Local data of the Modbus client

The address of the area from which data are to be read or to which data are to be written is specified in the function block ETHx_MOD_MAST or *ModTcpMast* at input "Data", via the ADR operator.

For more information about the function blocks use the Library Manager ↳ *Chapter 1.5.3 “Library Manager functionality” on page 2146*.

For the AC500, the following areas can be accessed using the ADR operator:

- Inputs area (%I area)
- Outputs area (%Q area)
- Area of non-buffered variables (VAR .. END_VAR or VAR_GLOBAL END_VAR)
- Addressable flag area (also protected areas for %M area)
- Area of buffered variables (VAR RETAIN .. END_VAR or VAR_GLOBAL RETAIN .. END_VAR)

Modbus telegrams

For a detailed description of the Modbus TCP telegrams and their elements please see the corresponding specifications on public websites.

Exception response by server

In operating mode Modbus client, the AC500 does only send requests, if the parameters at the MODMAST inputs are logically correct. Nevertheless, it can happen that a server cannot process the request of the client or that the server cannot interpret the request due to transmission errors or in case it's capabilities are exceeded in any way. In those cases, the server returns an exception response to the client. In order to identify this response as an exception response, the function code returned by the server is a logical OR interconnection of the function code received from the client and the value 80HEX.

General telegram description

Table 647: Server response

Error code	CRC	
	High	Low

Possible error codes of the client

Code	Description
01DEC	ILLEGAL FUNCTION The server does not support the function requested by the client
02DEC	ILLEGAL DATA ADDRESS Invalid operand address in the server or operand area exceeded
03DEC	ILLEGAL DATA VALUE At least one value is outside the permitted range of values
04DEC	SERVER DEVICE FAILURE An unrecoverable error occurred while the server was attempting to perform the requested action
05DEC	ACKNOWLEDGE Specialized use in conjunction with programming commands. The server has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed

Code	Description
06DEC	SERVER DEVICE BUSY Specialized use in conjunction with programming commands. The server is engaged in processing a long–duration program command. The client should retransmit the message later when the server is free.
07DEC	NEGATIVE ACKNOWLEDGE Specialized use in conjunction with programming commands. The server cannot perform the programming functions. Client should request diagnostic or error information from server.
08DEC	MEMORY PARITY ERROR Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server attempted to read record file, but detected a parity error in the memory. The client can retry the request, but service may be required on the server device.
09DEC	UNDEFINED Actually not defined by Modbus specification but might be used by particular servers.
10DEC	GATEWAY PATH UNAVAILABLE Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
11DEC	GATEWAY TARGET DEVICE FAILED TO RESPOND Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

Example

Table 648: Example:

Modbus request of the client:			
	Function code:	01	Read n bits
	Server operand address:	4000HEX = 16384DEC	Area for read access disabled in server

Modbus response of the server:			
	Function code:	81HEX	
	Error code:	03	

Processing bits

Some of the Modbus function codes are used to read or write bits (coils, discrete inputs). While a variable of data type WORD can be accessed easily, accessing a stream of bits is complex.

Data type ↬ *Chapter 1.4.1.19.5.10 “Data Type 'BIT'” on page 656* must not be mixed up with data type ↬ *Chapter 1.4.1.19.5.1 “Data type 'BOOL'” on page 647*. Variables of both types may have values ‘TRUE’ or ‘FALSE’. But while BIT means one single bit only, BOOL requires a byte (8 bit) of memory.

Modbus client

When accessing bits in a *Server*, the local data referred to at *Client* function blocks input data is always expected to be of format BOOL.

```
VAR
abBoolArray : ARRAY [0..15] OF BOOL;
ModMast : ModTcpMast;
END_VAR
```

```
abBoolArray[0]  := TRUE;
abBoolArray[15] := FALSE;

ModMast(Execute:= ModMastExecute,
        Eth:= ETH1,
        IPAdr:= IP_ADR_STRING_TO_DWORD('192.168.0.100'),
        UnitID:= 0,
        Fct:= 15,
        Addr:= 16#0000,
        Nb:= SIZEOF(abBoolArray),
        Data:= ADR(abBoolArray));
```

Modbus server

Using the bit offset

The simplest way to access a certain bit within a larger variable is to directly use the bit offset (0 based; see ↬ *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*).

```
VAR
awWordVariable AT %MW0 : ARRAY [0..7] OF WORD;
END_VAR
```

```
awWordVariable[0].0  := TRUE;
awWordVariable[7].15 := FALSE;
```

Defining symbolic names for the bit offsets

A more convenient way to access bits e.g. within a word is to define a symbolic name for each single offset ↳ *Chapter 1.4.1.19.4.9 “Bit Access in Variables” on page 641*.

```
VAR_GLOBAL CONSTANT
(*bit offsets in awWordVariable[0]*)
NameBitOffset_0_00 : INT := 0;
NameBitOffset_0_01 : INT := 1;
(*...*)
NameBitOffset_0_15 : INT := 15;
(*bit offsets in awWordVariable[7]*)
NameBitOffset_7_00 : INT := 0;
NameBitOffset_7_01 : INT := 1;
(*...*)
NameBitOffset_7_15 : INT := 15;
END_VAR
```

```
VAR
awWordVariable AT %MW0 : ARRAY [0..7] OF WORD;
END_VAR
```

```
awWordVariable[0].NameBitOffset_0_00  := TRUE;
awWordVariable[7].NameBitOffset_7_15 := FALSE;
```

Defining a data type

A further alternative is to define your own data types (see *Chapter 1.4.1.20.2.6 “Object 'DUT'” on page 835*) according to the requirements of your particular application (see *“Symbolic bit access in structure variables” on page 642*).

```
TYPE ControlWordType :
STRUCT
NameBitOffset_00 : BIT;
NameBitOffset_01 : BIT;
NameBitOffset_02 : BIT;
NameBitOffset_03 : BIT;
NameBitOffset_04 : BIT;
NameBitOffset_05 : BIT;
NameBitOffset_06 : BIT;
NameBitOffset_07 : BIT;
NameBitOffset_08 : BIT;
NameBitOffset_09 : BIT;
NameBitOffset_10 : BIT;
NameBitOffset_11 : BIT;
NameBitOffset_12 : BIT;
NameBitOffset_13 : BIT;
NameBitOffset_14 : BIT;
NameBitOffset_15 : BIT;
END_STRUCT
END_TYPE
```

```
VAR
atTypeVariable AT %MW0 : ARRAY [0..7] OF ControlWordType;
END_VAR
```

```
atTypeVariable[0].NameBitOffset_00 := TRUE;
atTypeVariable[7].NameBitOffset_15 := FALSE;
```

Defining a complex data type

In case your application requires some more complex data types you can combine data types (DUT; see *"Symbolic bit access in structure variables" on page 642*).

```
TYPE ControlWord0Type :
STRUCT
NameBitOffset_0_00 : BIT;
NameBitOffset_0_01 : BIT;
NameBitOffset_0_02 : BIT;
NameBitOffset_0_03 : BIT;
NameBitOffset_0_04 : BIT;
NameBitOffset_0_05 : BIT;
NameBitOffset_0_06 : BIT;
NameBitOffset_0_07 : BIT;
NameBitOffset_0_08 : BIT;
NameBitOffset_0_09 : BIT;
NameBitOffset_0_10 : BIT;
NameBitOffset_0_11 : BIT;
NameBitOffset_0_12 : BIT;
NameBitOffset_0_13 : BIT;
NameBitOffset_0_14 : BIT;
NameBitOffset_0_15 : BIT;
END_STRUCT
END_TYPE
```

ControlWord1Type

```
1  TYPE ControlWord1Type :
2  STRUCT
3  NameBitOffset_1_00 : BIT;
4  NameBitOffset_1_01 : BIT;
5  NameBitOffset_1_02 : BIT;
6  NameBitOffset_1_03 : BIT;
7  NameBitOffset_1_04 : BIT;
8  NameBitOffset_1_05 : BIT;
9  NameBitOffset_1_06 : BIT;
10 NameBitOffset_1_07 : BIT;
11 NameBitOffset_1_08 : BIT;
12 NameBitOffset_1_09 : BIT;
13 NameBitOffset_1_10 : BIT;
14 NameBitOffset_1_11 : BIT;
15 NameBitOffset_1_12 : BIT;
16 NameBitOffset_1_13 : BIT;
17 NameBitOffset_1_14 : BIT;
18 NameBitOffset_1_15 : BIT;
19 END_STRUCT
20 END_TYPE
```

```
TYPE ControlWordListType :
STRUCT
ControlWord0 : ControlWord0Type;
ControlWord1 : ControlWord1Type;
END_STRUCT
END TYPE
```

```
VAR
atWordListType AT %MW0 : ControlWordListType;
END_VAR
```

```
atWordListType.ControlWord0.NameBitOffset_0_00 := TRUE;
atWordListType.ControlWord1.NameBitOffset_1_15 := FALSE;
```

Pack/unpack BOOL variables

In case you prefer variables of type BOOL you can use the functions for packing MEM_Pack_BitsToByte and unpacking MEM_UnpackWord of the CAA_Memory.library, which can be found with the Library Manager *Chapter 1.5.3 “Library Manager functionality” on page 2146*.

```
VAR
abBoolVariable : ARRAY [0..15] OF BOOL;
wWordVariable0 : AT %MW0 WORD;
wWordVariable1 : AT %MW1 WORD;
END_VAR

VAR
Unpack  : Mem.UnpackWord;
END_VAR
```

```
abBoolVariable[0]  := TRUE;
abBoolVariable[15] := FALSE;

wWordVariable0 := Mem.PackBitsToWord(abBoolVariable[0], abBoolVariable[1], abBoolVariable[2], abBoolVariable[3],
                                     abBoolVariable[4], abBoolVariable[5], abBoolVariable[6], abBoolVariable[7],
                                     abBoolVariable[8], abBoolVariable[9], abBoolVariable[10], abBoolVariable[11],
                                     abBoolVariable[12], abBoolVariable[13], abBoolVariable[14], abBoolVariable[15]);
wWordVariable1 := 1;
Unpack(wValue:= wWordVariable1,
       xBit0=> abBoolVariable[0], xBit1=> abBoolVariable[1], xBit2=> abBoolVariable[2], xBit3=> abBoolVariable[3],
       xBit4=> abBoolVariable[4], xBit5=> abBoolVariable[5], xBit6=> abBoolVariable[6], xBit7=> abBoolVariable[7],
       xBit8=> abBoolVariable[8], xBit9=> abBoolVariable[9], xBit10=> abBoolVariable[10], xBit11=> abBoolVariable[11],
       xBit12=> abBoolVariable[12], xBit13=> abBoolVariable[13], xBit14=> abBoolVariable[14], xBit15=> abBoolVariable[15]);
```

Function block ETHx_MOD_MAST and ModTcpMast

These function blocks are only required for the operating mode Modbus client. It handles the communication (transmission of telegrams to the servers and receipt of telegrams from the servers). The function block can be used for the Ethernet interfaces of the controller.

ETHx_MOD_MAST is contained in the library Ethernet_AC500_V10.lib.

ModTcpMast is contained in the library ABB_ModbusTcp_AC500.

1.6.5.1.12 Fast counters

Fast counters in AC500 devices

For AC500 devices the function "fast counter" is available in S500 I/O modules as of firmware version V1.3.

For AC500-eCo V3 devices the function "fast counter" is available in onboard I/Os of PM50x2 modules, according to the CPU type, the fast inputs have different functionality or frequency.

Integrated fast counters are only available for digital I/O modules.

The digital I/O modules on the I/O bus contain two fast counters each.

If the counter is used, it needs up to 2 digital inputs and one digital output.

If the fast counter is deactivated, the inputs and outputs reserved for the counter can be used for other tasks.

See *Chapter 1.6.6.2.13.9 “Fast counter” on page 3778*.

A fast counter is available in the following constellations:

- In digital I/O modules, connected to an AC500 processor module.
- In AC500-eCo V3 processor modules PM50x2 with onboard I/Os
- In CANopen communication interface modules.

- In Modbus, PROFIBUS and PROFINET communication interface modules and in the connected digital I/O modules.
- In digital I/O modules, connected to an EtherCAT communication interface module.

Fast counter integrated in S500 modules

The following table shows the S500 modules which contain a fast counter and which of the digital inputs and outputs are reserved for the counter.

Module	Assigned inputs [1])		Assigned output	Remarks
	Channel A	Channel B	Channel C [2]) or (CF)	
DA501	DC16	DC17	DC18	The counter function is not available if the modules are mounted on the communication interface modules CI581-CN or CI582-CN
DA502	DC16	DC17	DC18 - in mode 1 and mode 2 DO0 - in mode 101 and mode 102 [3])	
DC522	C8	C9	C10	
DC523	C16	C17	C18	
DC532	C24	C25	C26	
DI524	I24	I25	No hardware output available	
DX522	I0	I1	The counter does not activate any relay output	
CI501-PNIO, CI541-DP, CI581-CN, CI521-MODTCP	DI0	DI1	DO0	
CI502-PNIO, CI542-DP, CI582-CN; CI522-MODTCP	DI8	DI9	DO8	

[1]) The two hardware inputs (channels A and B) are also and always available within the normal process image, irrespective of the operating mode of the counter.

[2]) The hardware output channel C is activated by the fast counter only in the operating modes 1 and 2.

[3]) Especially for module DA502: The counter operating mode 101 is the same as mode 1, but the assigned output is DO0 instead of DC18. Also the counter operating mode 102 is the same as mode 2, but the assigned output is DO0 instead of DC18.

The counter function is performed within the communication interface module and, accordingly, in the digital I/O module(s). It works independently of the user program and is therefore able to respond quickly to external signals. A simultaneous counter operation of several digital I/O modules is possible.

Each module counter can be configured for one out of 10 possible modes. The desired operating mode is selected in the PLC configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. While integrating a module containing a fast counter in the PLC configuration, the necessary operands are created and reserved immediately. Thus, a counter implementation carried out later on does not cause an address shift.

Features independent of the fast counter operating mode

- The pulses at the fast counters' inputs or the evaluated signals of the traces A and B in case of incremental position sensors are counted.
- The counting frequencies of the communication interface modules of PROFINET, PROFIBUS and CANopen are max. 200 kHz (in modes 1 to 6), max. 50 kHz (in mode 7), max. 35 kHz (in mode 9), and max. 20 kHz (in mode 10).
- If the modules DA501, DC522, DC523, DC532 are used, each counting input must be circuited externally in series with a resistor of 470 Ω / 1 W, in order to safely avoid influences from the deactivated module outputs to the connected sensors.
- The positive signal edges are counted, if not noted differently.
- By setting the operating mode 0, the counting function is switched off. In this case, the reserved inputs and outputs can be used for other tasks. Simultaneous use of these terminals for the fast counter and other signals must be avoided.
- The fast counter's actual value is provided as a double word (32 bits).
- The fast counter can count upwards in all operating modes. It counts beginning at the start value (set value) up to the end value (max. from 0 to 4,294,967,295 or hexadecimal from 00 00 00 00 to FF FF FF FF. After reaching 4,294,967,295, the counter jumps with the next pulse to 0. When the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only when the fast counter is set again (set value), CF is reset to FALSE.

Further information

- Operating modes of the fast counter: *Chapter 1.6.6.2.13.9.1.2 “Operating modes” on page 3781*
- Configuration of the fast counter: *Chapter 1.6.6.2.13.9 “Fast counter” on page 3778*

1.6.5.1.13 Onboard I/O on AC500-eCo V3 processor modules

Onboard I/Os

The AC500-eCo V3 processor modules have onboard I/Os which provide several functionalities. According to the CPU type, the number or the functionality of the onboard I/Os can be different.

Intended purpose

Table 649: Numbers and types of the onboard I/Os

Processor module	No. and type of digital inputs	No. and type of digital outputs	No. and type of configurable inputs/outputs
PM5012-T-ETH	6 24 V DC (one isolation group)	4 0.5 A max., transistor (one isolation group)	None
PM5012-R-ETH	6 24 V DC (one isolation group)	4 2 A max., relay (two isolation groups)	None

Processor module	No. and type of digital inputs	No. and type of digital outputs	No. and type of configurable inputs/outputs
PM5032-T-ETH	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5032-R-ETH	12 24 V DC (one isolation group)	6 2 A max., relay (two isolation groups)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5052-T-ETH	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5052-R-ETH	12 24 V DC (one isolation group)	6 2 A max., relay (two isolation groups)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5072-T-2ETH	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)
PM5072-T-2ETHW	12 24 V DC (one isolation group)	8 0.5 A max., transistor (one isolation group)	2 24 V DC input or 0.5 A max., transistor output (one isolation group)

Functionality

<table>
<tr><th rowspan="2">Parameter</th><th colspan="4">Value</th></tr>
<tr><th>PM5012-T-ETH</th><th>PM5012-R-ETH</th><th>PM5032-T-ETH
PM5052-T-ETH
PM5072-T-2ETH(W)</th><th>PM5032-R-ETH
PM5052-R-ETH</th></tr>
<tr><td>Digital inputs</td><td colspan="2">6</td><td colspan="2">12</td></tr>
<tr><td>Functionality of digital inputs
(encoder, fast counter, counter, interrupt)</td><td colspan="2" rowspan="3">6 DI fast input 24 V DC (max. 5 kHz)
usable as
● 6 DI 24 V DC standard
● 2 channel 5 kHz encoder with frequency measurement or
● 2 channel 5 kHz encoder with frequency measurement and with touch/reset using standard DI or
● 2 fast counter (5 kHz)
● 4 DI as interrupt input with 1 dedicated interrupt task and input information</td><td colspan="2">4 DI fast input 24 V DC (max. 200 kHz)
usable as
● 4 DI 24 V DC standard or
● 4 fast counter (100 kHz) or
● 2 A/B encoder (200 kHz) with frequency measurement or
● 2 full A/B encoders 0 and 1 (200 kHz) with frequency measurement and with touch/reset using standard highspeed (5 kHz) DI
● 1 full A/B encoder 0 (200 kHz) with frequency measurement and optional with touch/reset using 2 touch/sync inputs with A/B encoder 0</td></tr>
<tr><td colspan="2">4 DI fast input 24 V DC (5 kHz)
usable as
● 4 DI 24 V DC standard or
● 4 DI as interrupt input with 1 dedicated interrupt task and input information
● 4 touch/sync inputs with A/B encoder 0 or 1</td></tr>
<tr><td colspan="2">4 standard DI 24 V DC</td></tr>
<tr><td>Digital outputs</td><td colspan="2">4</td><td>8</td><td>6</td></tr>
</table>

Parameter	Value			
	PM5012-T-ETH	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH(W)**	**PM5032-R-ETH** **PM5052-R-ETH**
Functionality of digital outputs	**4 fast output DO-T** 24 V DC/0.5 A (max. 5 kHz) usable as ● 4 DO-T 24 V DC/0.5 A or ● 4 PWM Note: The speed must be limited below 100 Hz. The low speed PWM can be used for heating control. ● 4 limit switch	**4 DO-R** 24 V DC / 240 V AC 2A in 2 groups	**4 fast output DO-T** 24 V DC (100 kHz) usable as ● 4 DO-T 24 V DC/0.5 A ● 4 limit/ switch outputs for encoder/ counter or ● 4 PWM (30 kHz, 2 µs accuracy and maximum duty 95 %) or ● 2 PTO (200 kHz) CW/CCW or Pulse/Direc-tion ● 4 PTO (PWM) 100 kHz Pulse/ Direction using standard output **4 fast output DO-T** 24 V DC/0.5 A (5 kHz) (max. 5 kHz) usable as ● 4 DO-T 24 V DC/0.5 A ● 4 limit/ switch outputs for encoder/ counter or ● 4 PWM Note: The speed must be limited below 100 Hz. The low speed PWM can be used for heating control.	**6 DO-R** 24 V DC / 240 V AC 2A in 2 groups

Parameter	Value			
	PM5012-T-ETH	**PM5012-R-ETH**	**PM5032-T-ETH** **PM5052-T-ETH** **PM5072-T-2ETH(W)**	**PM5032-R-ETH** **PM5052-R-ETH**
Digital inputs/ outputs, configurable	-	-	2	2
Functionality of digital inputs/ outputs, configurable	-	-	**2 DC 24 V DC** ● 2 standard I/Os configurable	**2 DC 24 V DC** usable as ● 2 DC standard (DI 24 V DC or DO-T) or ● 2 PWM (30 kHz) or ● 1 PTO (200 kHz) as Pulse/Direction or CW/CCW
LED displays	For signal states			
Internal power supply	Via processor module			
External power supply	Via UP and ZP terminal			

Fast counter in AC500-eCo V3 (Onboard I/O in PM50xx)

For AC500 devices the function "fast counter" is available in S500 I/O modules as of firmware version V1.3.

For AC500-eCo V3 devices the function "fast counter" is available in onboard I/Os of PM50xx.

The AC500-eCo V3 processor modules with onboard I/Os provide some special functionality on the digital inputs or digital outputs. Fast counter, encoder inputs, interrupt inputs or PWM/PTO outputs are available depending on the device used.

The fast counter functionality can be activated within the onboard I/O configuration.

The fast counter can work in pulse/direction mode or A/B track counter mode.

The pulse/direction counter detects the rising edge of the counter input. It will increase or decrease the count value (depending on the direction input) at every rising edge.

The A/B track counter is used to count the signal from an encoder.

The counter can count with quad phases. In the following the behavior of the A/B track counter is described.

Further information:

Operating modes of the fast counter: Chapter 1.6.6.2.13.9.1.2 “Operating modes” on page 3781

Configurarion of the fast counter: Chapter 1.6.6.2.7.2 “Fast counters in the onboard I/Os” on page 3710

1.6.5.1.14 Simple motion

Introduction

The AC500-eCo V3 PLC provide several HW and SW features allowing to realize some motion application.

Specific fast onboard I/O and dedicated SW library function blocks (simple motion) are available and can manage up to 2x Axis on the CPU.

The simple motion capability is based on a library for the onboard I/O and some motion control blocks allowing point-to-point or velocity control.

All the AC500-eCo V3 PLC from Basic, Standard or Pro type offer dedicated feature according to their performance classes.

	Basic		**Standard**		**Pro**
	PM5012-x-ETH		**PM5032-x-ETH / PM5052-x-ETH**		**PM5072-T-2ETH**
	Relay outputs	**Transistor outputs**	**Relay outputs**	**Transistor outputs**	**Transistor outputs**
HSC - High-speed counter	Up to 2 (5 kHz)		Up to 4 (100 kHz)		
Frequency measurement	Up to 2 (5 kHz)		Up to 2 (200 kHz)		
A / B Encoder	1 A/B simple encoder (5 kHz) with sync/reset		Up to 2 A/B encoder 200 kHz with sync/reset inputs		
Interrupt inputs	Up to 4		Up to 4		
PTO - pulse-train output	-		1 Pulse/ Direction or CW/CCW both mode with 200 kHz	Up to 2 Pulse/Direction or CW/CCW both mode with 200 kHz	
				Up to 4 Pulse/Direction with 100 kHz using fast Output channels for Pulse and standard outputs for direction on SW motion function bloc	
PWM - pulse-width modulation	-	Up to 4 (100 Hz)	Up to 2 (30 kHz)	Up to 4 (30 kHz)	
Limit switches	-		Up to 2	Up to 8	

Hardware components for motion control

Basic CPU – PM5012-R-ETH and PM5012-T-ETH

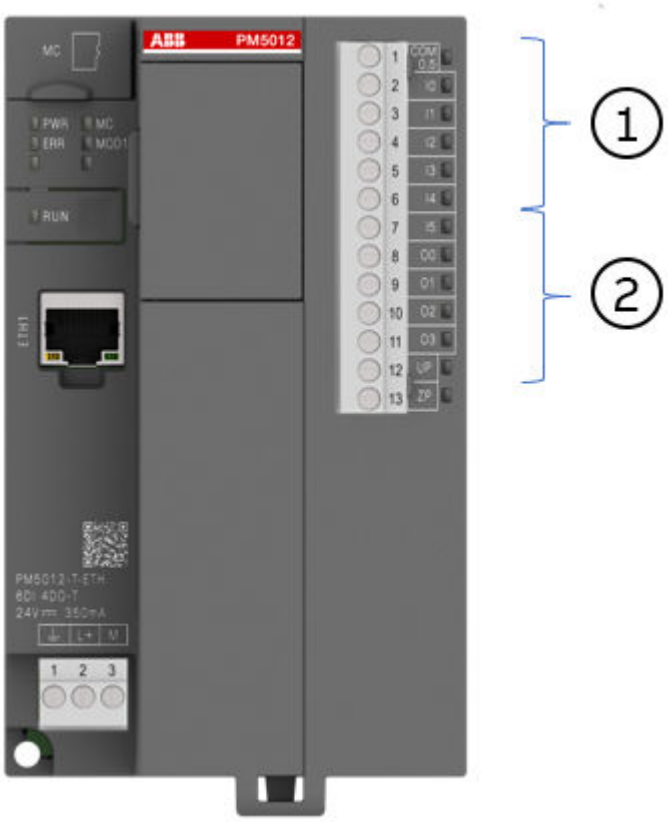


Fig. 307: Example: PM5012-T-ETH

1 HSC 5kHz frequency measurement interrupt I/O
2 PWM output

Standard and Pro CPU - PM5032-x-ETH / PM5052-x-ETH / PM5072-T-2ETH

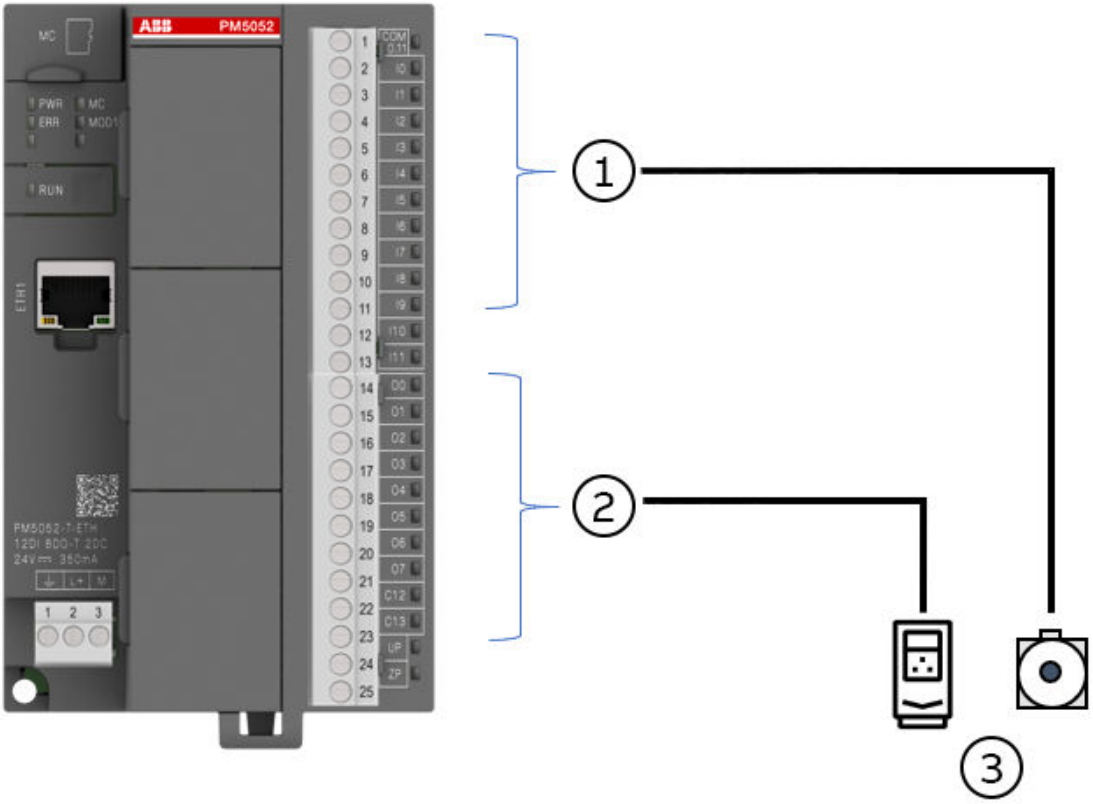


Fig. 308: Example: PM5052-T-ETH

1 HSC 100 kHz and 5 kHz A/B Encoder 200 kHz interrupt I/O standard inputs
2 PTO 100 kHz/200kHz PWM 30 kHz limit switch standard outputs
3 Drives, Encoder, Stepper Motor

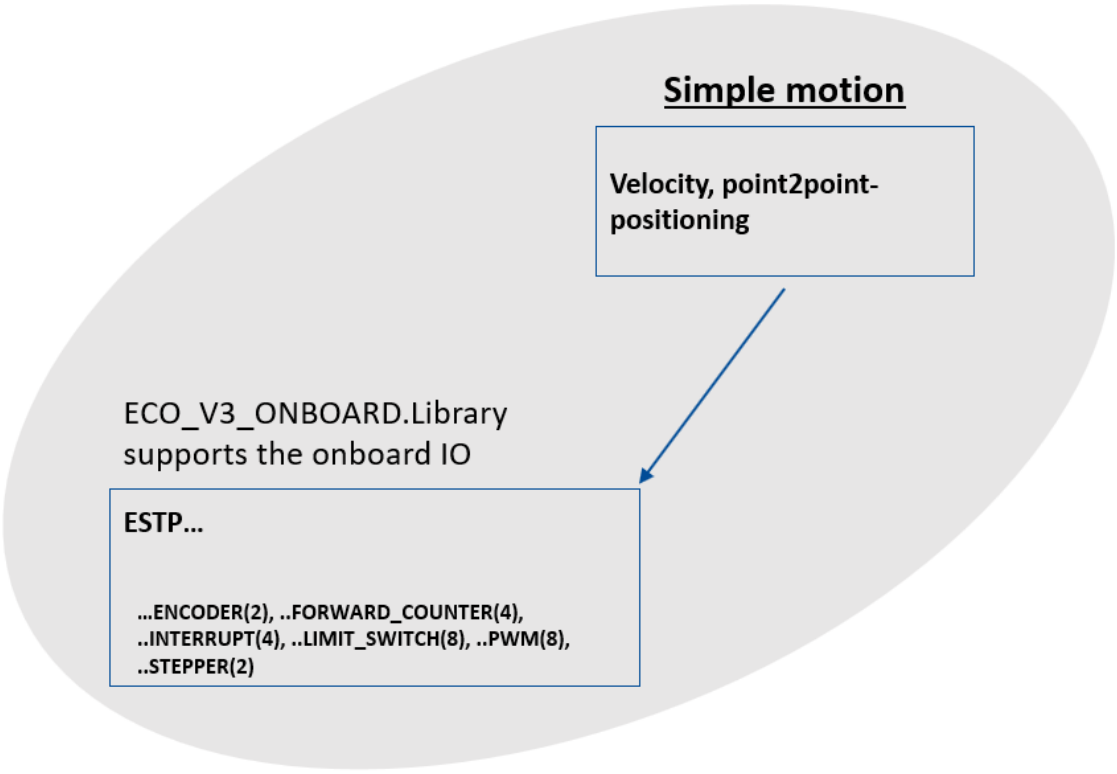
For PLC with relay outputs, the input features are identical.

The digital configurable inputs/outputs can be used for PTO/PWM functions.

System technology

The following chapters describe the system technology of the AC500-eCo V3 using motion examples.

The simple motion set of function blocks is standard part of the system libraries for AC500-eCo V3.



Use the onboard I/Os as encoder with A and B signals

Parameter configuration

The onboard I/O accept encoder signal A and B. When configure the encoder track A, the encoder track B will be automatically inserted.

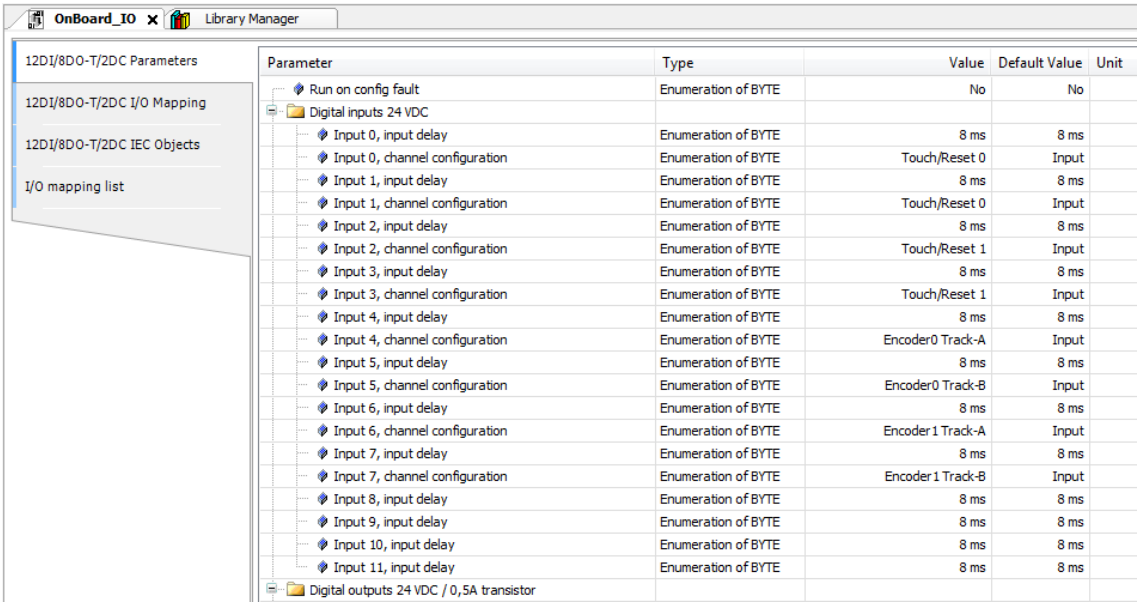
The user can configure the following input channel as encoder input.

- *"Encoder 0 Track – A"*: Input channel 4
- *"Encoder 0 Track – B"*: Input channel 5
- *"Encoder 1 Track – A"*: Input channel 6
- *"Encoder 1 Track – B"*: Input channel 7

After configuring the encoder input channel, the user can configure the touch/reset for the respective encoder channel.

See also the following chapter: ↪ Chapter 1.6.6.2.5 "Configure the onboard I/O channel" on page 3700.

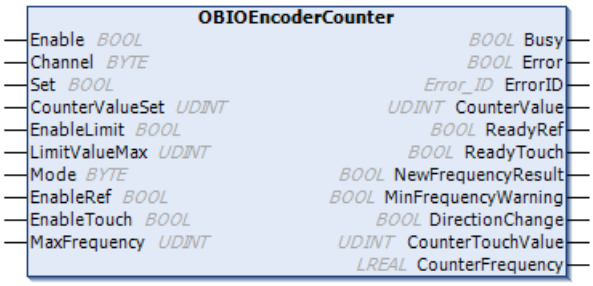
E.g. PM50x2-T-xETH with 2x A/B encoders with Touch/Reset on I0..I3



Parameter	Type	Value	Default Value	Unit
Run on config fault	Enumeration of BYTE	No	No	
Digital inputs 24 VDC				
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 0, channel configuration	Enumeration of BYTE	Touch/Reset 0	Input	
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 1, channel configuration	Enumeration of BYTE	Touch/Reset 0	Input	
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 2, channel configuration	Enumeration of BYTE	Touch/Reset 1	Input	
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 3, channel configuration	Enumeration of BYTE	Touch/Reset 1	Input	
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 4, channel configuration	Enumeration of BYTE	Encoder0 Track-A	Input	
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 5, channel configuration	Enumeration of BYTE	Encoder0 Track-B	Input	
Input 6, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 6, channel configuration	Enumeration of BYTE	Encoder1 Track-A	Input	
Input 7, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 7, channel configuration	Enumeration of BYTE	Encoder1 Track-B	Input	
Input 8, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 9, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 10, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 11, input delay	Enumeration of BYTE	8 ms	8 ms	
Digital outputs 24 VDC / 0,5A transistor				

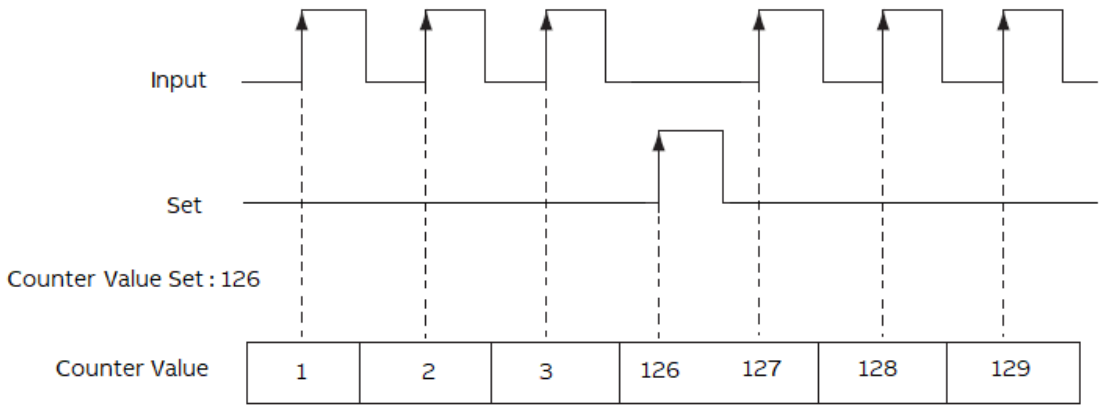
Function block

OBIOEncoder Counter



If *"Enable"* is TRUE, the *"OBIOEncoderCounter"* instruction increments the counter by one base on the input.

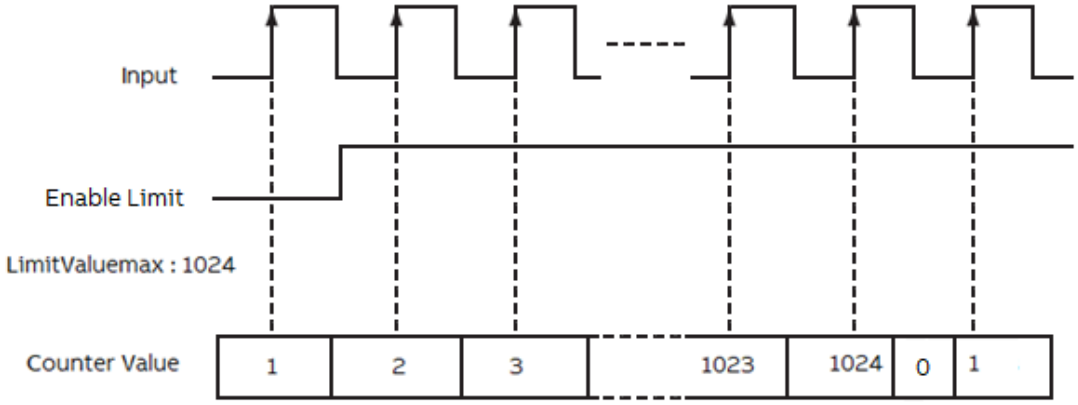
If *"Set"* bit is TRUE, the *"OBIOEncoderCounter"* instruction moves the *"CounterValueSet"* to the *"CounterValue"*.



If *"Enable"* is TRUE, the *"OBIOEncoderCounter"* instruction increments the counter by one based on the input.

If *"EnableLimit"* bit is TRUE, the accumulated value continues incrementing.

After *"CounterValue"* reaches the *"LimitValueMax"*, the *"OBIOEncoderCounter"* instruction writes 0 to the *"CounterValue"*.



"Encoder Counter Mode": 0 = *"90° Mode"*.

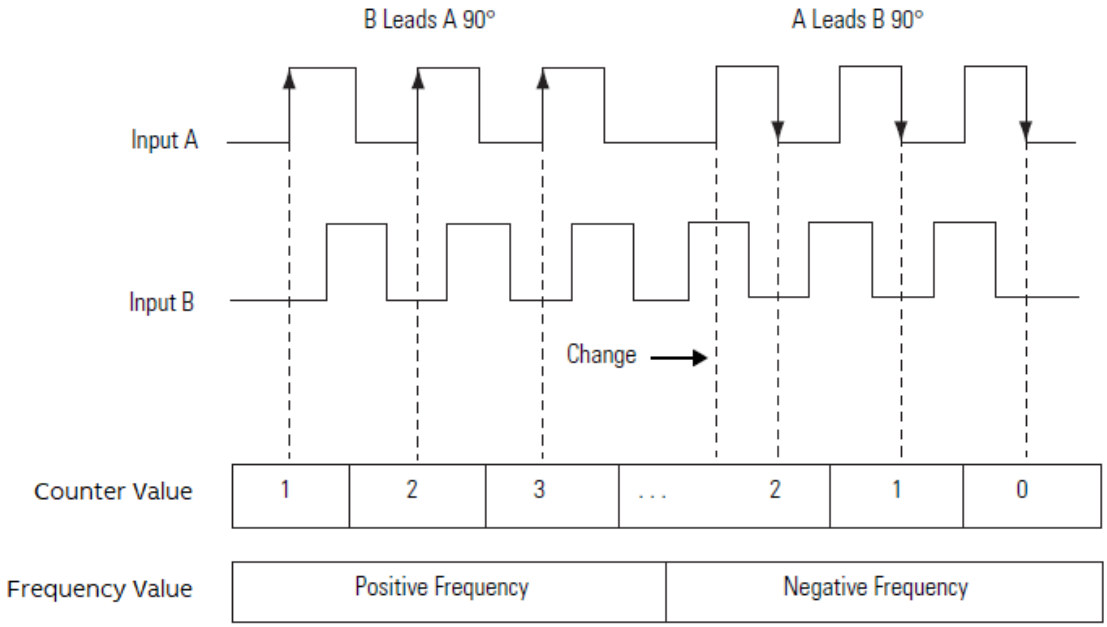
In this encoder counter mode, an increasing count results when input B is 90° ahead of input A.

The count is initiated on the rising edge of input A, and the direction of the encoder is clockwise (positive).

The module produces a decreasing count when input A is 90° ahead of Input B.

The count is initiated on the falling edge of input A, and the direction is counterclockwise (negative).

By monitoring both the number of pulses and the phase relationships of input A and B, you can accurately determine the position and direction of the rotation.



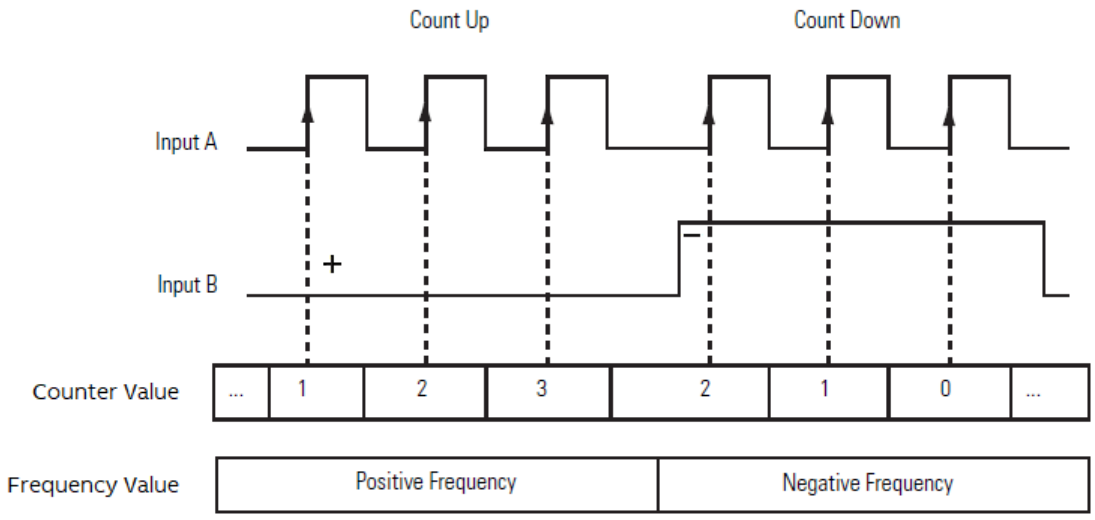
"Encoder Counter Mode": 1 = *"Pulse/Direction"*.

In this encoder counter mode, the count increases or decreases based on the state of input B, which can be a random signal.

If input B is high, the counter will count down.

If input B is low the counter counts up.

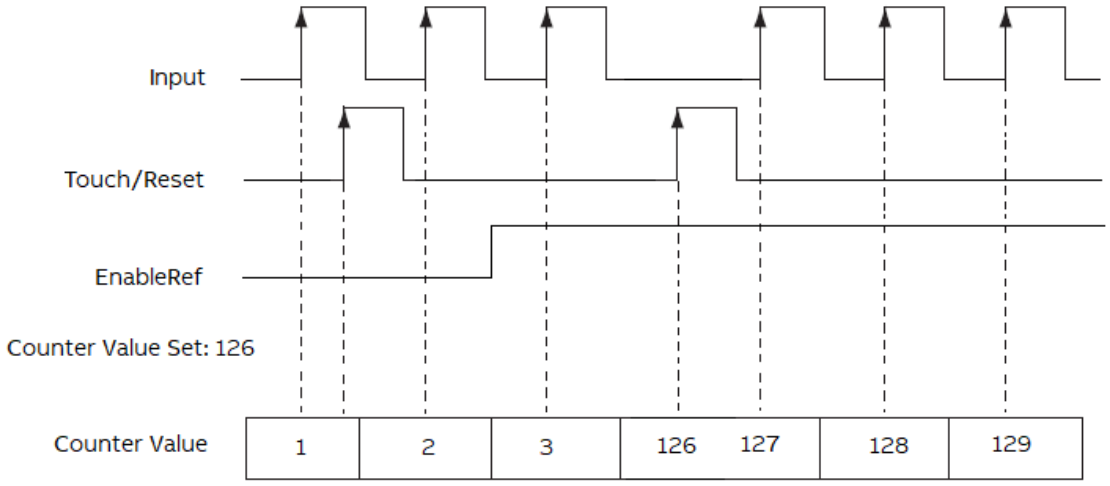
Counting is done on the leading edge of input A.



If *"Enable"* is TRUE, the *"OBIOEncoderCounter"* instruction increments the counter by one based on the input.

If *"EnableRef"* bit is TRUE, the *"OBIOEncoderCounter"* instruction is ready to receive the touch/reset input.

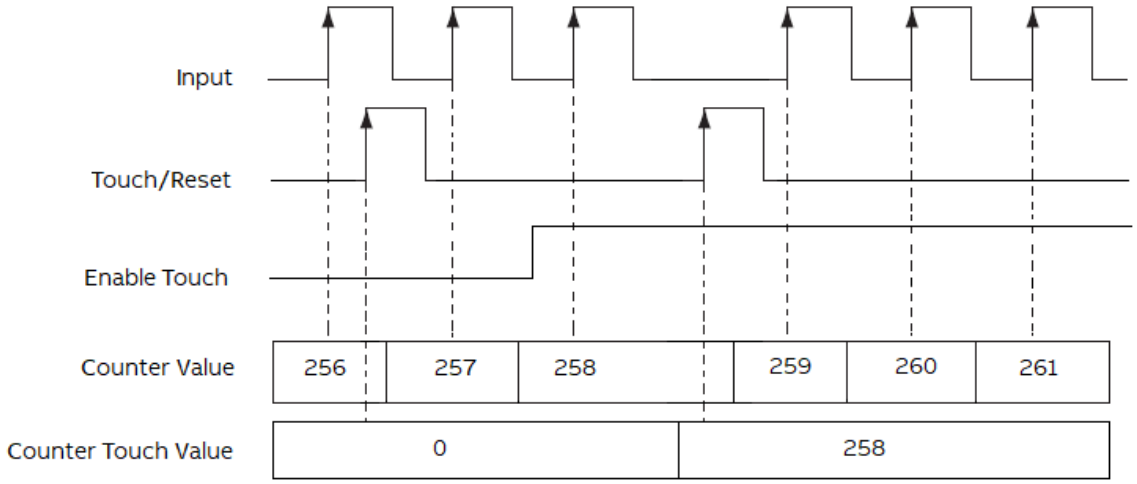
If the *"Touch/Reset"* input is TRUE, the current *"CounterValue"* will be replaced by the *"CounterValueSet"*.



If *"Enable"* is TRUE, the *"OBIOEncoderCounter"* instruction increments the counter by one based on the input.

If *"EnableTouch"* bit is TRUE, the *"OBIOEncoderCounter"* instruction is ready to receive the *"Touch/Reset"* input.

If the *"Touch/Reset"* input is TRUE, the current *"CounterValue"* will be captured and written to the *"CounterTouchValue"*.



Use the onboard I/Os as forward counter

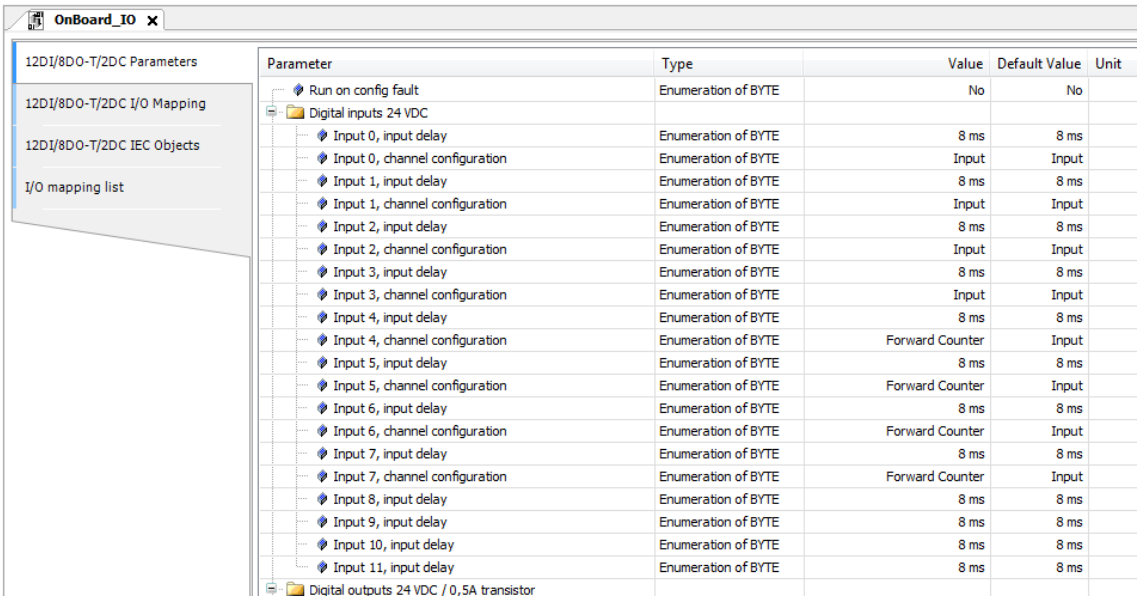
Parameter configuration

The Onboard I/O accept pulse input as forward counter.

User can configure the following input channel as forward counter.

- *"Forward Counter 0"*: Input channel 4
- *"Forward Counter 1"*: Input channel 5
- *"Forward Counter 2"*: Input channel 6
- *"Forward Counter 3"*: Input channel 7

E.g. PM50x2-x-xETH with forward counter on fast inputs I4…I7

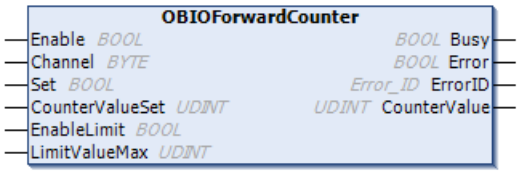


Parameter	Type	Value	Default Value	Unit
Run on config fault	Enumeration of BYTE	No	No	
Digital inputs 24 VDC				
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 0, channel configuration	Enumeration of BYTE	Input	Input	
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 1, channel configuration	Enumeration of BYTE	Input	Input	
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 2, channel configuration	Enumeration of BYTE	Input	Input	
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 3, channel configuration	Enumeration of BYTE	Input	Input	
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 4, channel configuration	Enumeration of BYTE	Forward Counter	Input	
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 5, channel configuration	Enumeration of BYTE	Forward Counter	Input	
Input 6, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 6, channel configuration	Enumeration of BYTE	Forward Counter	Input	
Input 7, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 7, channel configuration	Enumeration of BYTE	Forward Counter	Input	
Input 8, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 9, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 10, input delay	Enumeration of BYTE	8 ms	8 ms	
Input 11, input delay	Enumeration of BYTE	8 ms	8 ms	
Digital outputs 24 VDC / 0,5A transistor				

See also the following chapter: Chapter 1.6.6.2.5 "Configure the onboard I/O channel" on page 3700

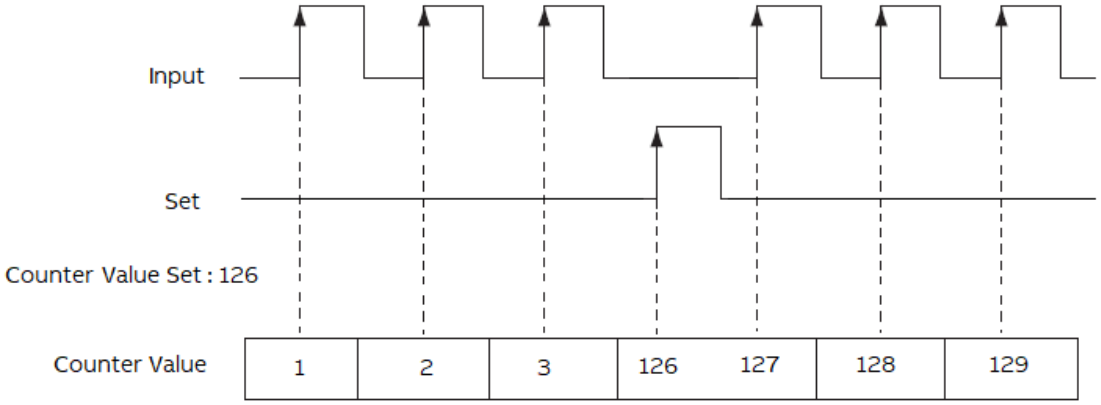
Function block

OBIOForward Counter



If *"Enable"* is TRUE, the *"OBIOForwardCounter"* instruction increments the counter by one based on the input.

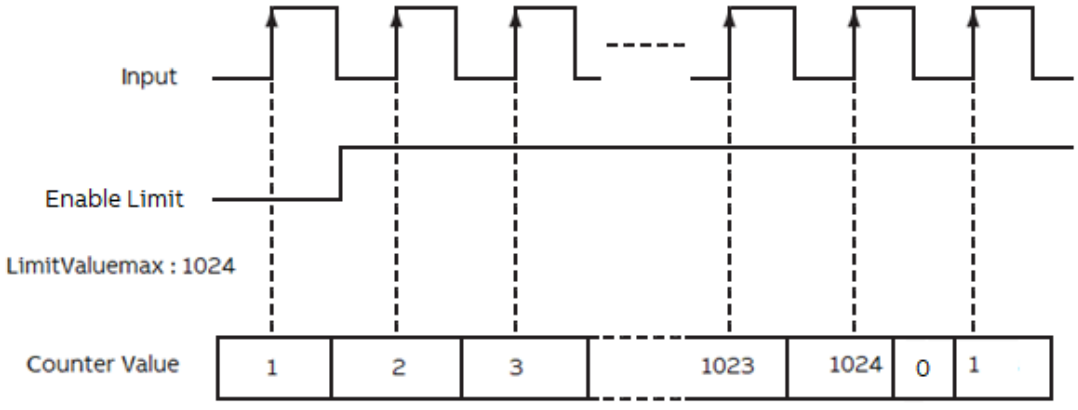
If *"Set"* bit is TRUE, the *"OBIOForwardCounter"* instruction moves the *"CounterSetValue"* to the *"CounterValue"*.



If *"Enable"* is TRUE, the *"OBIOForwardCounter"* instruction increments the counter by one based on the input.

If *"EnableLimit"* bit is TRUE, the accumulated value continues incrementing.

After *"CounterValue"* reaches the *"LimitValueMax"*, the *"OBIOForwardCounter"* instruction writes 0 to the *"CounterValue"*.



Use the onboard I/Os as interrupt input with dedicated interrupt task

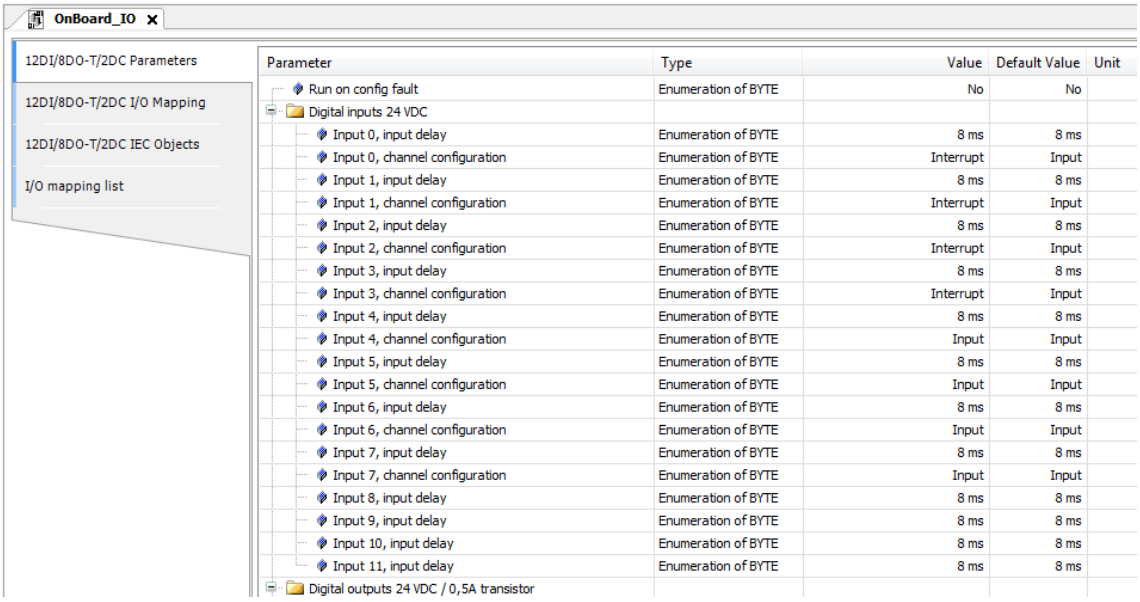
Parameter configuration

The onboard I/O input can be configured as interrupt input to trigger the interrupt task.

The user can configure the following input channel as interrupt input.

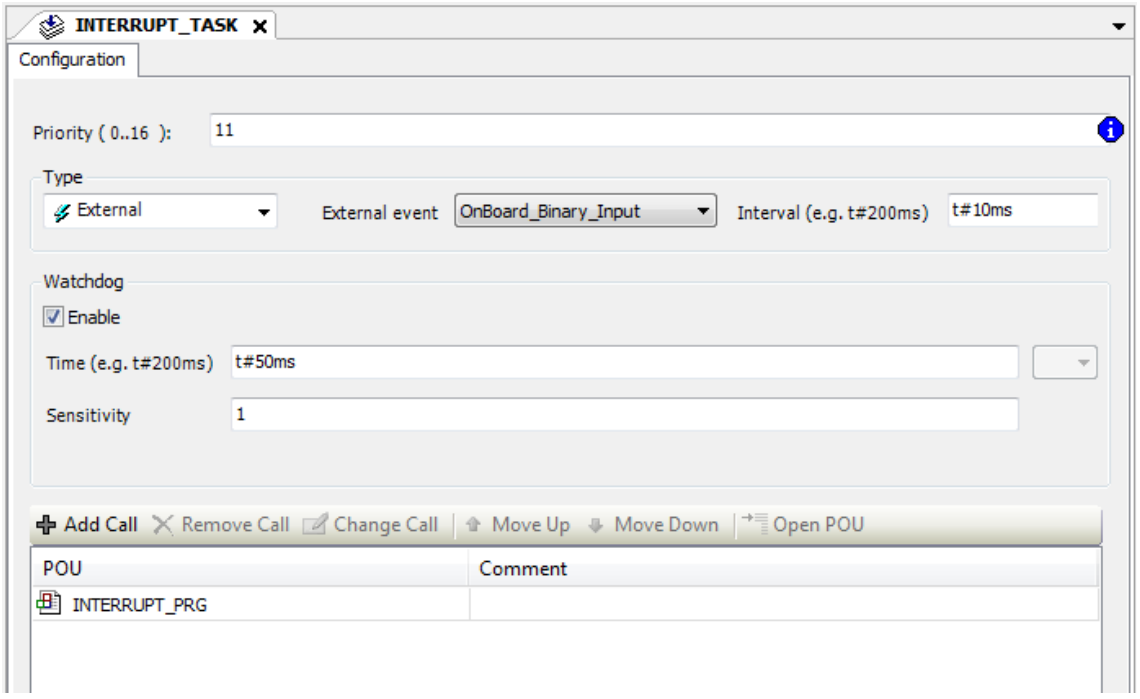
- Interrupt input 0 : Input Channel 0
- Interrupt input 1 : Input Channel 1
- Interrupt input 2 : Input Channel 2
- Interrupt input 3 : Input Channel 3

E.g. PM50x2-x-xETH with interrupt inputs on digital inputs I0…I3

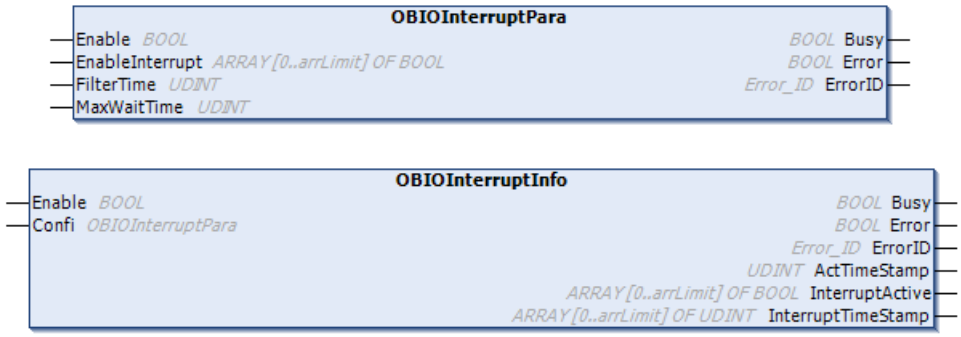


See also the following chapter: Chapter 1.6.6.2.5 “Configure the onboard I/O channel” on page 3700

After configuring the parameter, the user need to create a new task with the *“Type”* set to *“External”* and the *“External event”* set to *“OnBoard_Binary_Input”*.



Function block



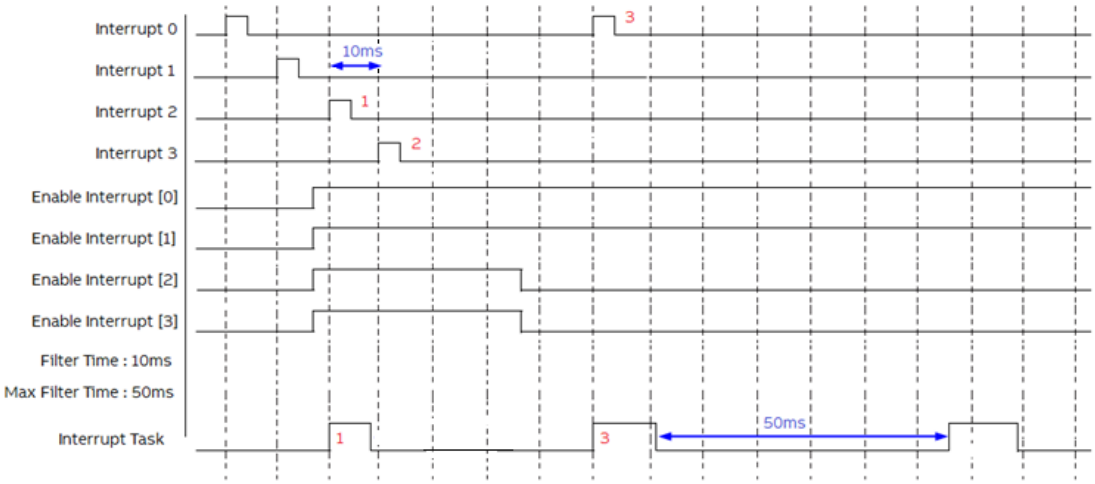
The *"OBIOInterruptPara"* instruction is configured for 4 interrupt inputs.

if *"EnableInterrupt"* bit is TRUE, the *"OBIOInterruptInfo"* instruction is ready to receive the interrupt input.

If the interrupt input is TRUE, the interrupt task will be executed.

If the second interrupt is TRUE with the interval less than 10 ms (as set), the execution of the interrupt task will be ignored.

If no interrupt occurred in 50 ms (as set), the interrupt task is executed automatically.

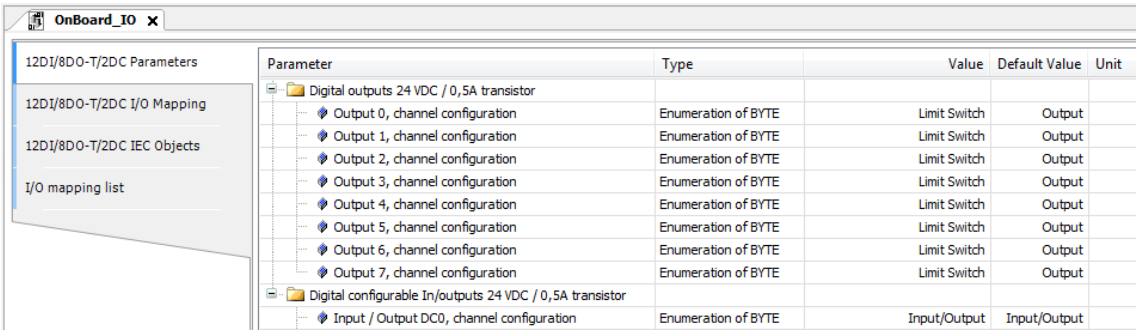


Use the onboard I/Os as output limit switch

Parameter configuration

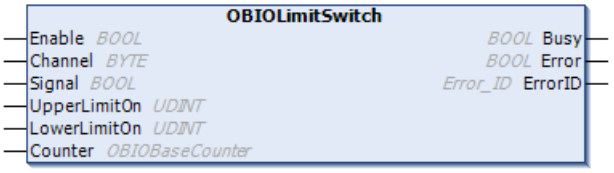
Thje user can configure the following output channel as limit switch.

- *"LimitSwitch 0"*: Output channel 0
- *"LimitSwitch 1"*: Output channel 1
- *"LimitSwitch 2"*: Output channel 2
- *"LimitSwitch 3"*: Output channel 3
- *"LimitSwitch 4"*: Output channel 4
- *"LimitSwitch 5"*: Output channel 5
- *"LimitSwitch 6"*: Output channel 6
- *"LimitSwitch 7"*: Output channel 7

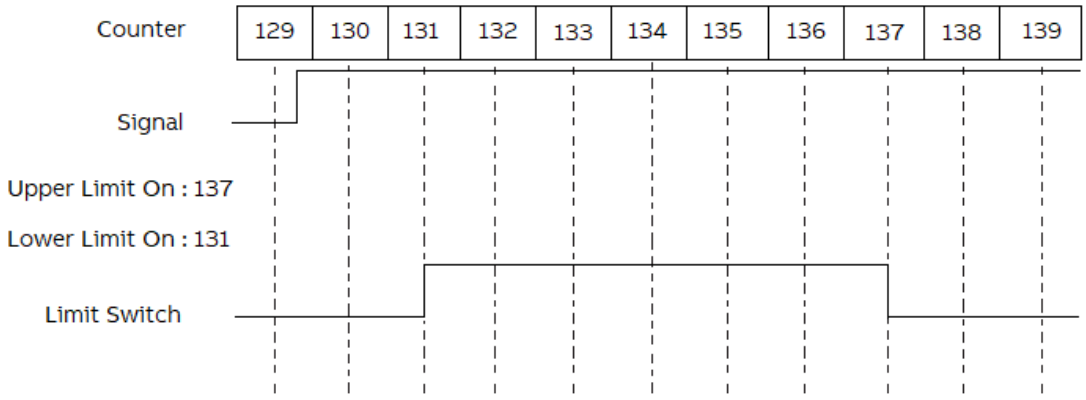


See also the following chapter: Chapter 1.6.6.2.5 "Configure the onboard I/O channel" on page 3700

Function block



If the counter value reaches the *"LowerLimitOn"* preset, it will write to the LimitSwitch output based on the signal until the *"UpperLimitOn"* preset is reached.



Use the onboard I/Os as PTO (pulse-train output) with 100 kHz frequency (max. 2 PTO using PTO HW channels)

Parameter configuration

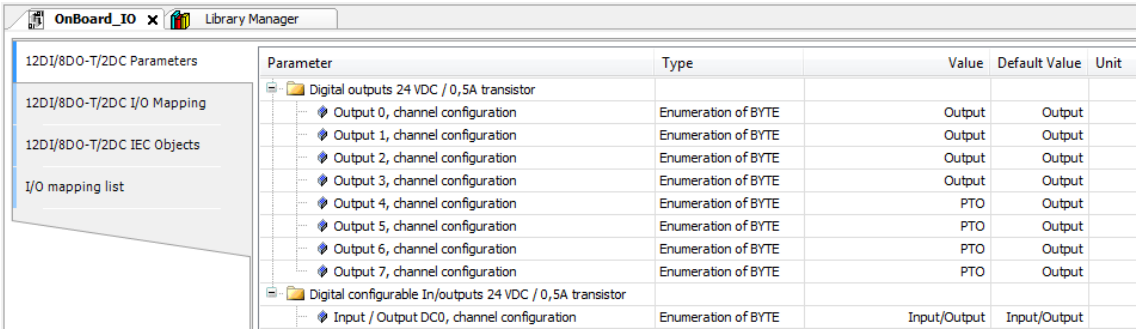
The user can configure the following output channels as PTO (pulse-train output).

- *"PTO"*: Output channel 4
- *"PTO"*: Output channel 5
- *"PTO"*: Output channel 6
- *"PTO"*: Output channel 7

If the user configures the output 4 as PTO, the output 5 is automatically configured as PTO.

If the user configures the output 6 as PTO, the output 7 is automatically configured as PTO.

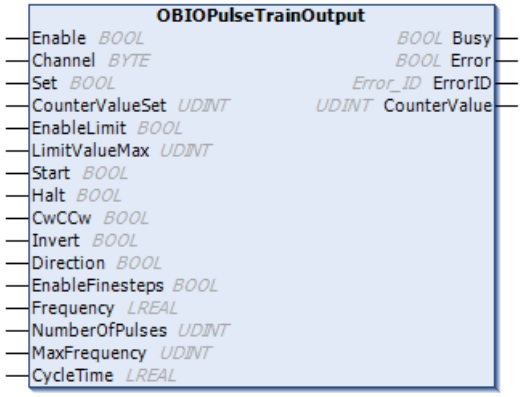
The input *“CwCCw”* of the function block *“OBIOPulseTrainOutput”* determines the output 5 and 7 as *“CounterClockWise”* or *“Direction”* if it is set as PTO.



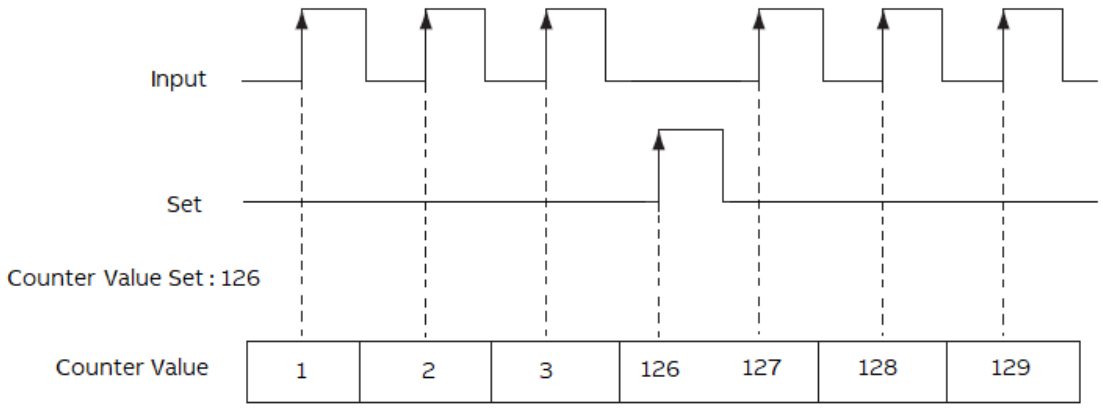
Parameter	Type	Value	Default Value	Unit
Digital outputs 24 VDC / 0,5A transistor				
Output 0, channel configuration	Enumeration of BYTE	Output	Output	
Output 1, channel configuration	Enumeration of BYTE	Output	Output	
Output 2, channel configuration	Enumeration of BYTE	Output	Output	
Output 3, channel configuration	Enumeration of BYTE	Output	Output	
Output 4, channel configuration	Enumeration of BYTE	PTO	Output	
Output 5, channel configuration	Enumeration of BYTE	PTO	Output	
Output 6, channel configuration	Enumeration of BYTE	PTO	Output	
Output 7, channel configuration	Enumeration of BYTE	PTO	Output	
Digital configurable In/outputs 24 VDC / 0,5A transistor				
Input / Output DC0, channel configuration	Enumeration of BYTE	Input/Output	Input/Output	

See also the following chapter: Chapter 1.6.6.2.5 “Configure the onboard I/O channel” on page 3700

Function block

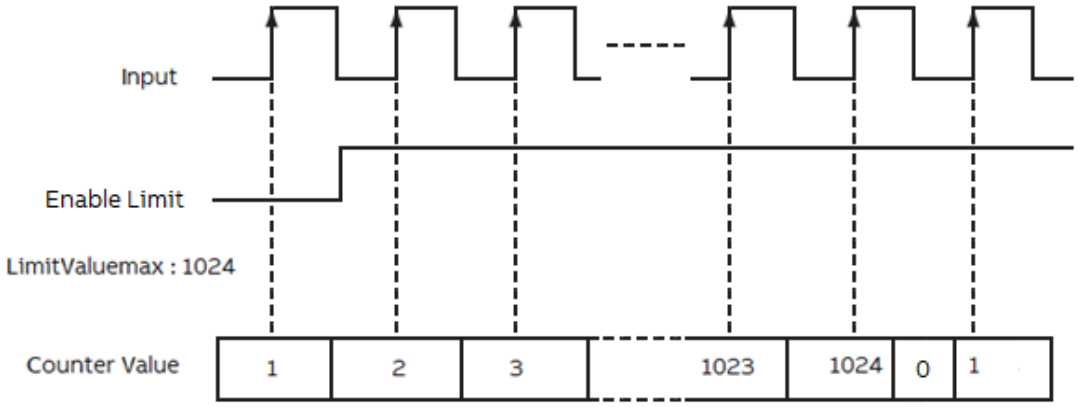


If *“Set”* bit is TRUE, the instruction moves the *“CounterSetValue”* to the *“CounterValue”*.

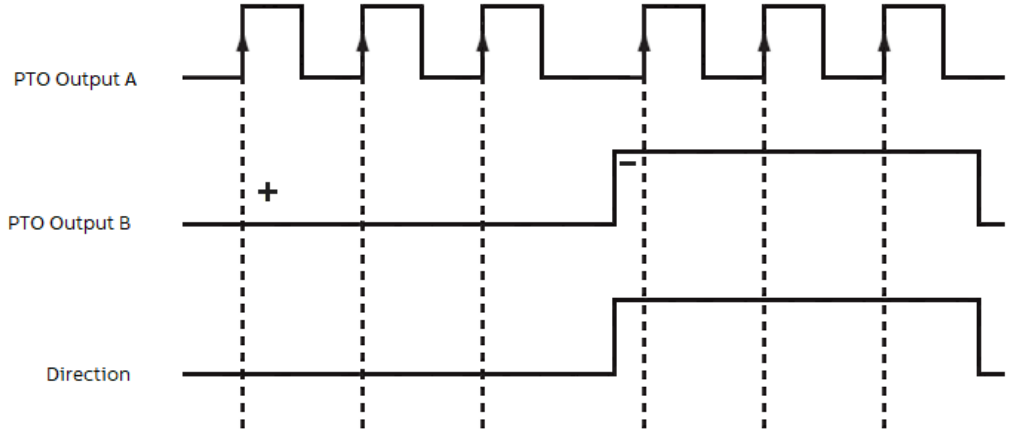


If *“EnableLimit”* bit is TRUE, the accumulated value continues incrementing.

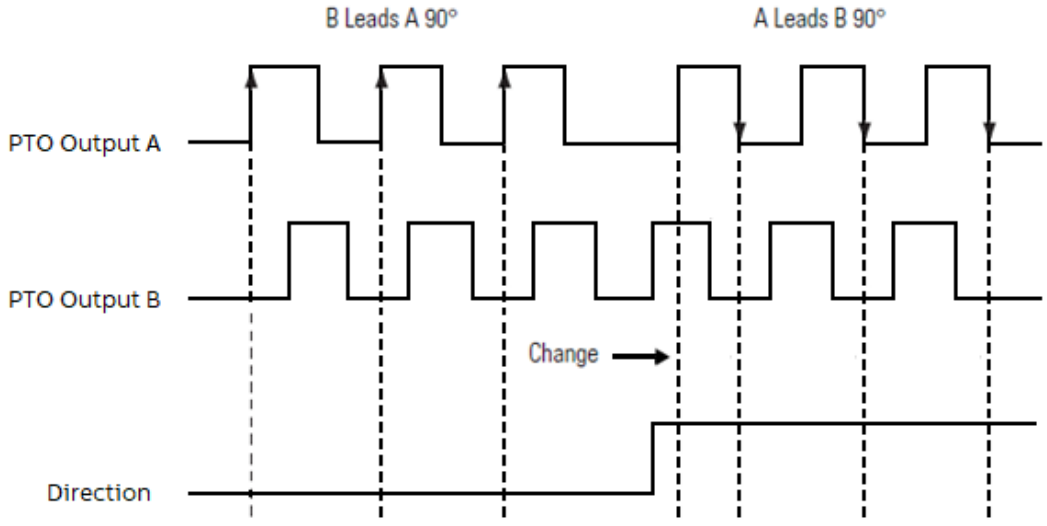
After *“CounterValue”* reaches the *“LimitValueMax”*, the instruction writes 0 to the *“CounterValue”*.



If the input *"CwCCw"* of the *"OBIOPulseTrainOutput"* is set to FALSE, the PTO output channel B is toggled based on the direction.



If the input *"CwCCw"* of the *"OBIOPulseTrainOutput"* is set to TRUE. The PTO output channel A will lead by 90° or PTO output channel B will lead by 90° depending on direction.



Use the onboard I/Os as PTO (pulse-train output) with 200 kHz frequency (max. 2 PTO using PTO HW channels) and Simple Motion OBIOMotionPTO function block

Parameter configuration

Only the Standard and Pro processor modules can be used with PTO outputs. The Basic processor modules PM5012 do not have PTO outputs.

The available PTO outputs can be used as PTO with Pulse/Direction or PTO with CW/CCW mode when the channels have been configured as PTO outputs.

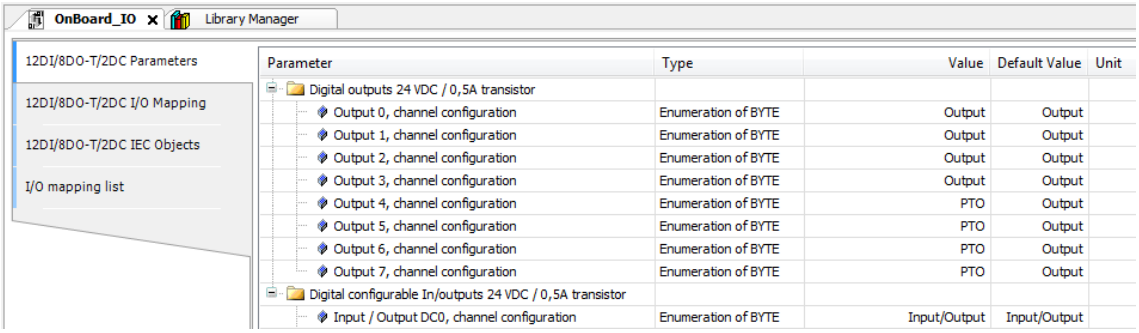
The user can configure the following output channels as PTO (pulse-train output).

- *"PTO"*: Output channel 4
- *"PTO"*: Output channel 5
- *"PTO"*: Output channel 6
- *"PTO"*: Output channel 7

If the user configures the output 4 as PTO, the output 5 is automatically configured as PTO.

If the user configures the output 6 as PTO, the output 7 is automatically configured as PTO.

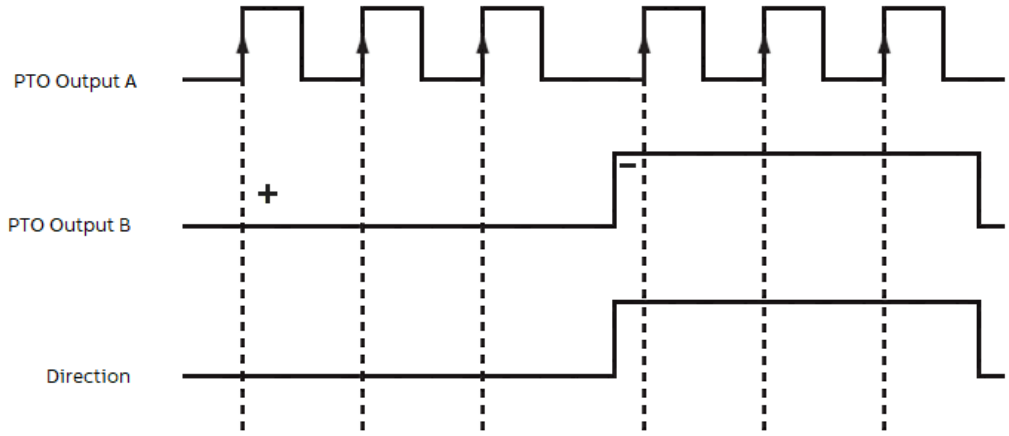
The input *"CwCCw"* of the function block *"OBIOPulseTrainOutput"* determines the output 5 and 7 as *"CounterClockWise"* or *"Direction"* if it is set as PTO.



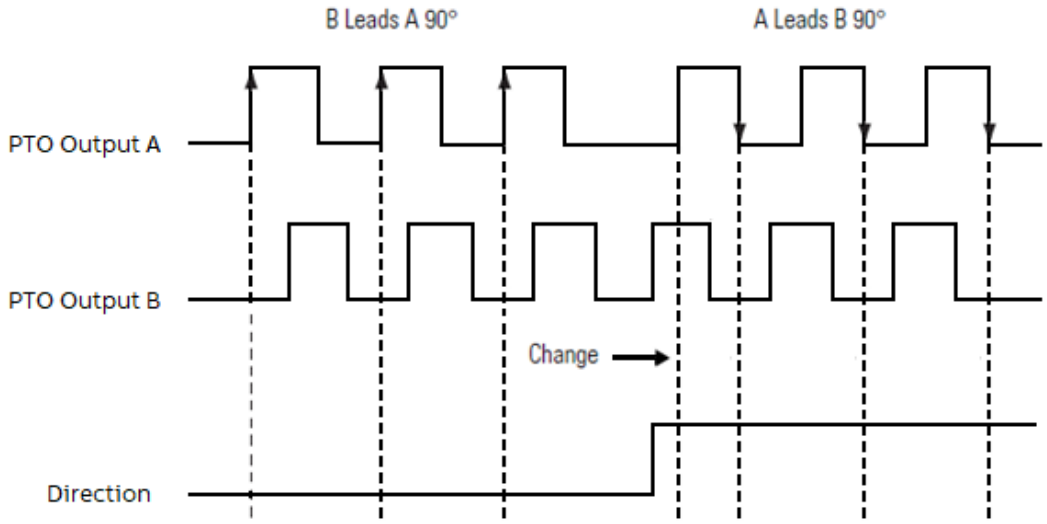
See also the following chapter: Chapter 1.6.6.2.5 "Configure the onboard I/O channel" on page 3700

Function block

If the input *"CwCCw"* of the *"OBIOMotionPTO"* is set to FALSE, the PTO output channel B is toggled based on the direction.



If the input *"CwCCw"* of the *"OBIOMotionPTO"* is set to TRUE. The PTO output channel A will lead by 90° or PTO output channel B will lead by 90° depending on direction.



Use the onboard I/Os as PTO (pulse-train output) with 100 kHz frequency (Max. 4 PTO using PWM HW channels) and Simple Motion OBIOMotionPWM function bloc

Parameter configuration

It is possible to have also up to 4 PTO channels only with Pulse/Direction mode on the AC500-eCo V3 CPU by using the fast outputs O4…O7 configured as PWM outputs and using a specific Motion function block and standard outputs for direction channel.

Only the Standard and Pro processor modules can be used with PTO (PWM) outputs. The Basic processor modules PM5012 do not have PTO outputs.

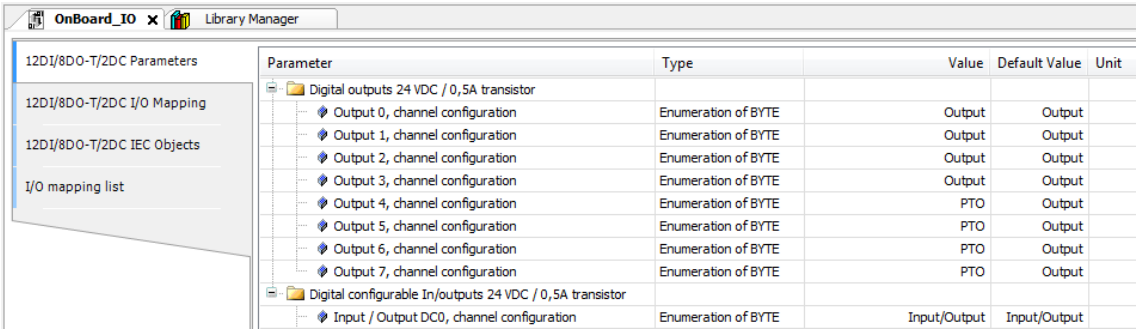
The available software PTO outputs can be used as PTO with Pulse/Direction or PTO with CW/CCW mode when the channels have been configured as PWM outputs.

The user must configure the following output channels as PWM outputs and use the *"OBIOMotionPWM"* function block.

- *"PWM"*: Output channel 4
- *"PWM"*: Output channel 5
- *"PWM"*: Output channel 6
- *"PWM"*: Output channel 7

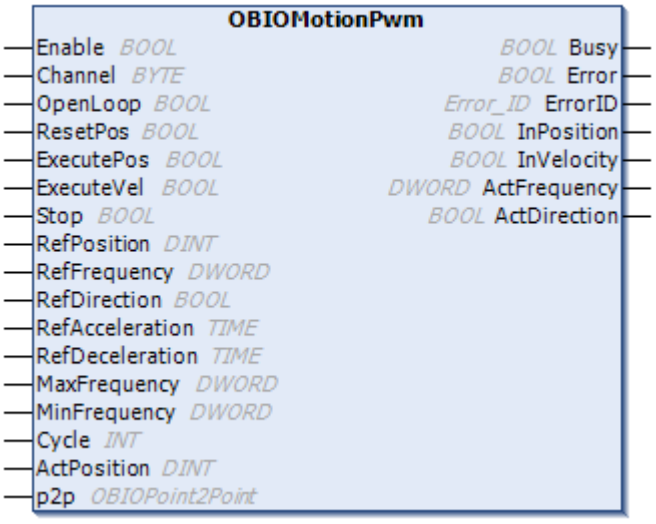
If the user configures the output 4...7 as PWM using the *"OBIOMotionPWM"* function block, up to four Software PTO can be realized offering then only the Pulse/Direction mode.

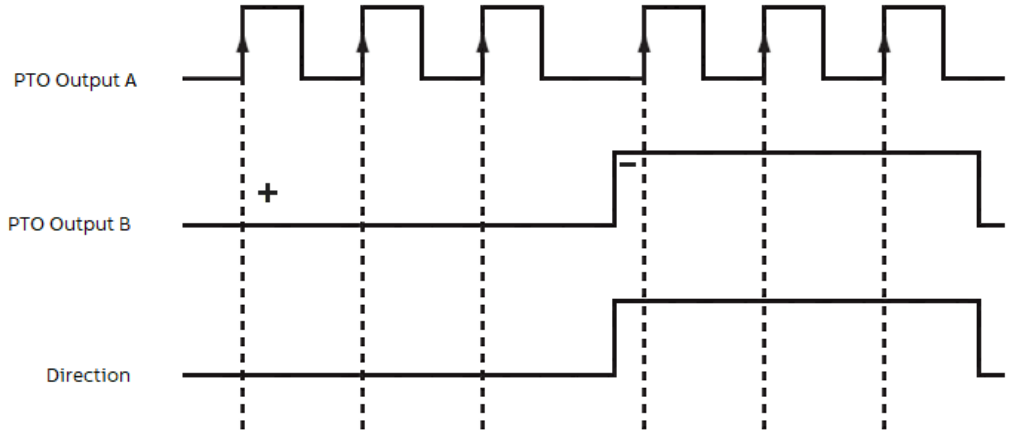
The Pulse output will always use the fast output channels O4…O7 and the direction output of the function block can be assigned to any other output e.g. O0…O3 or also outputs from a S500 I/O module.



See also the following chapter: Chapter 1.6.6.2.5 "Configure the onboard I/O channel" on page 3700

Function block



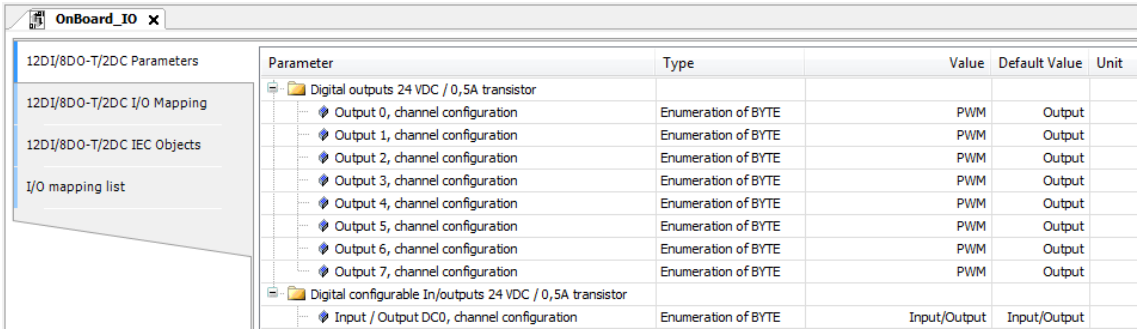


Use the onboard I/Os as output PWM (pulse-width modulation)

Parameter configuration

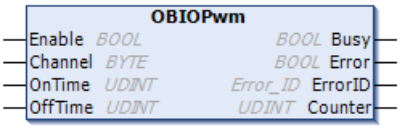
The user can configure the following output channels as PWM (pulse-width modulation).

- *"PWM 0"*: Output channel 0
- *"PWM 1"*: Output channel 1
- *"PWM 2"*: Output channel 2
- *"PWM 3"*: Output channel 3
- *"PWM 4"*: Output channel 4
- *"PWM 5"*: Output channel 5
- *"PWM 6"*: Output channel 6
- *"PWM 7"*: Output channel 7

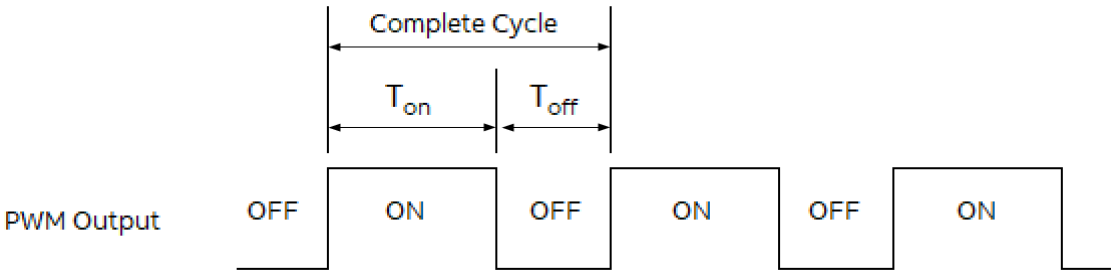


See also the following chapter: Chapter 1.6.6.2.5 "Configure the onboard I/O channel" on page 3700

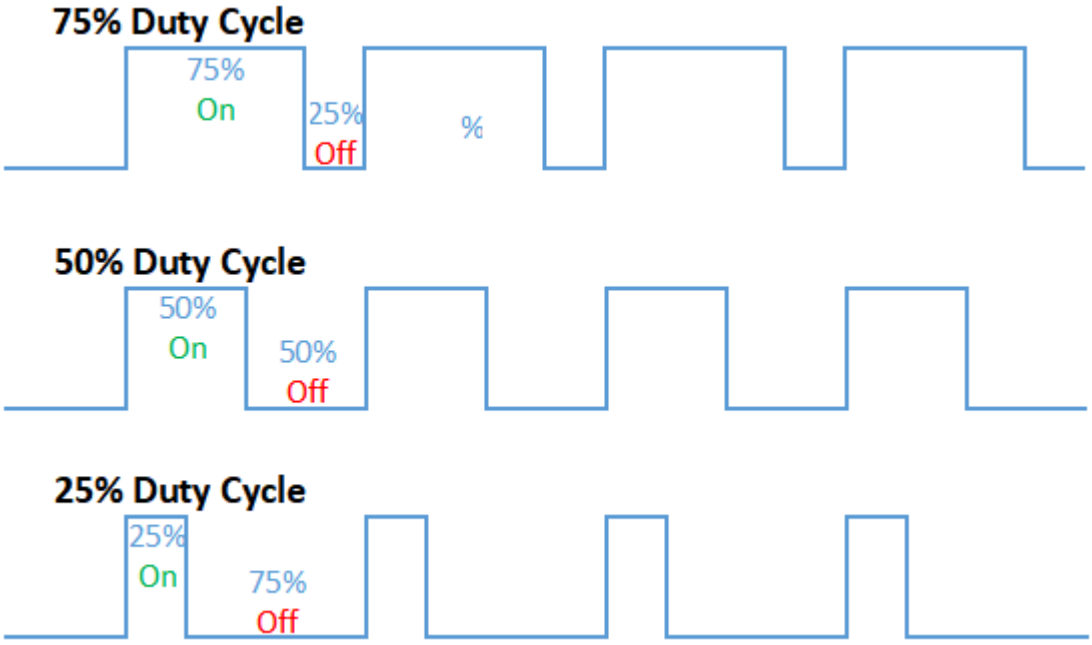
Function block



The complete cycle of the *"PWM"* is based on the *"OnTime"* and *"OffTime"*.



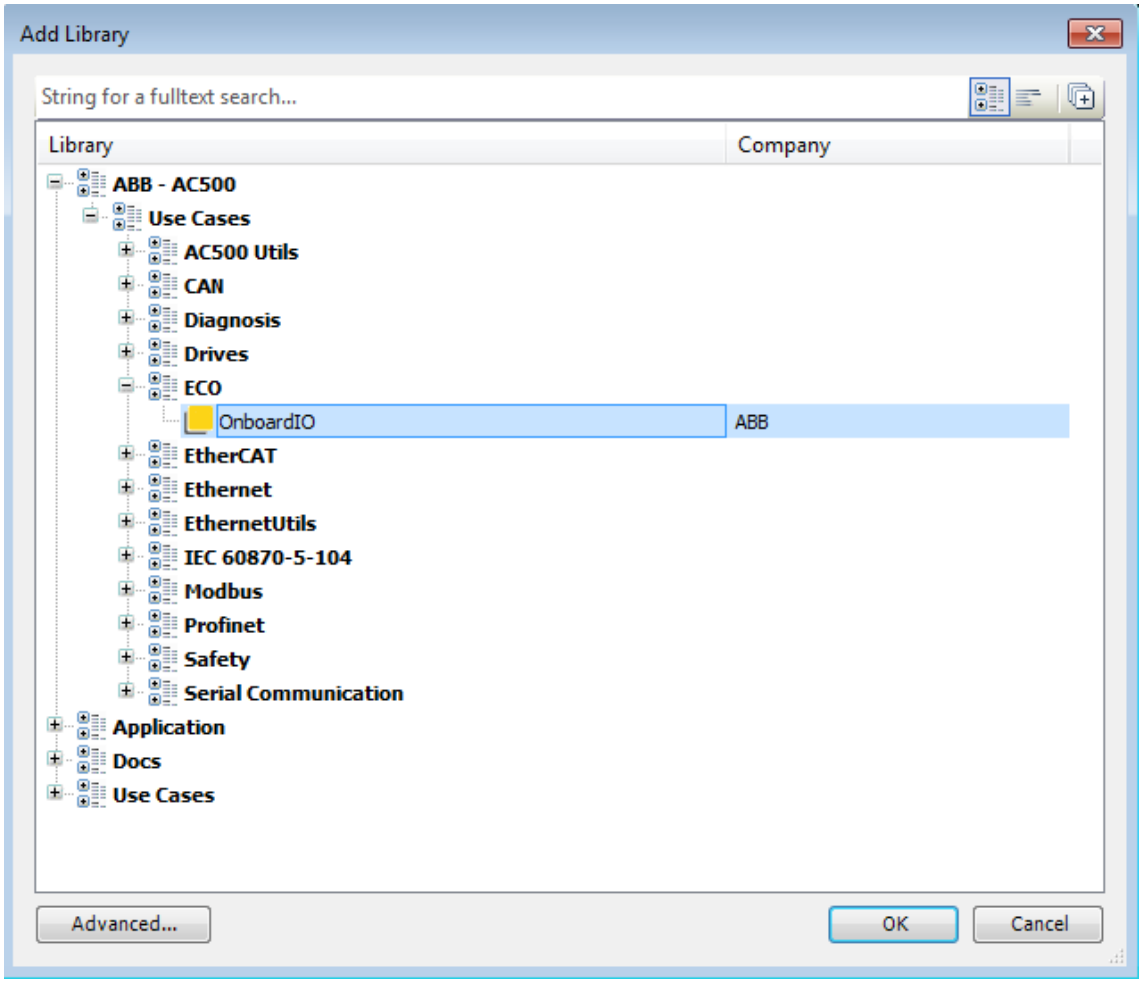
The duty cycle ratio of the *"PWM"* is based on the formula = T_{on} / (T_{on} + T_{off}).



Function block description

Function block descriptions of all V3 libraries are available in the library manager. ↳ *Chapter 1.10 "Reference, function blocks" on page 4292*

1. Under *“Application”* open *“Library Manager”*.
2. Select *“Add Library”*.



⇨ A list of all available libraries is displayed.

Libraries in folder *“ABB - AC500”* are created by ABB and tested in combination with Automation Builder.

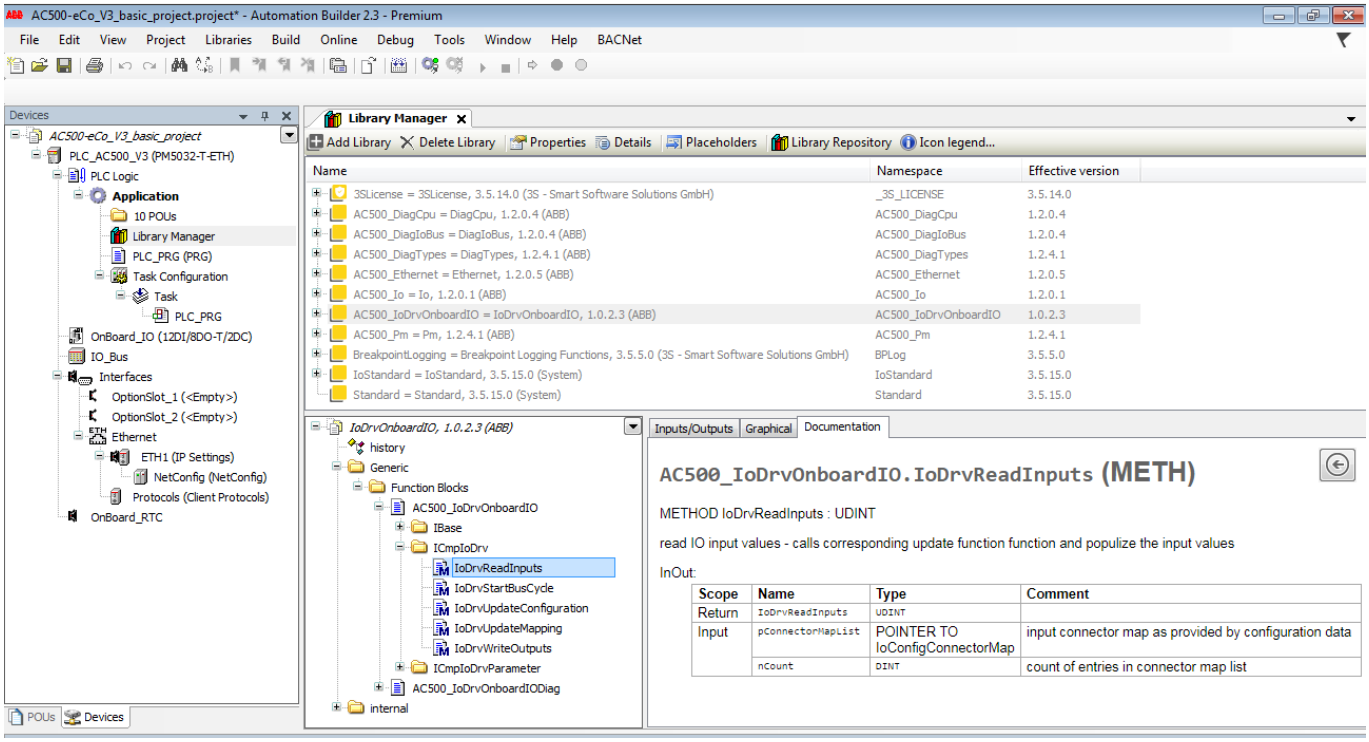
We recommend to use libraries of subfolder *“Use Cases”* for your project.

Libraries in subfolder *“Intern”* are necessary for internal procedures.

All 3S libraries distributed with Automation Builder are required by ABB libraries and have been tested in combination with AC500 and Automation Builder. Additional 3S libraries that are not distributed with Automation Builder can easily be added. There are no known major issues with using them, however, be aware that they are not tested by ABB.

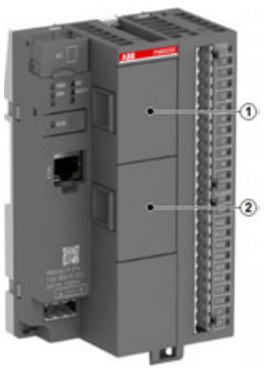
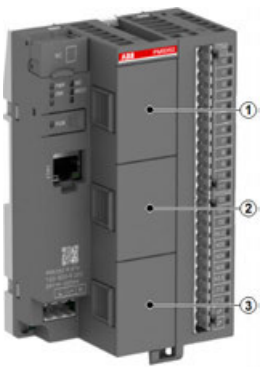
3. Add a library.
4. Choose the added library in Library Manager to access the documentation.

The function block description is shown as an example as follows.



AC500-eCo V3 option board slots for processor modules PM50xx

Depending on the processor module type, up to three option board slots are available on the CPU for different purpose like digital or analog I/O extension, serial interface or special module for specific functionality.

Option board slots	Basic CPU	Standard CPU		Pro CPU
	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
Figure • 1: Slot 1 • 2: Slot 2 • 3: Slot 3				
Option board slot 1	X	X	X	X
Option board slot 2	-	X	X	X
Option board slot 3	-	-	X	X

Usable option boards onAC500-eCo V3	Basic CPU	Standard CPU		Pro CPU
	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
TA5130-KNXPB	-	-	-	X [1])
TA5131-RTC	X[1])	-	-	-

Usable option boards onAC500-eCo V3	Basic CPU	Standard CPU		Pro CPU
	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
TA5101-4DI	X	X	X	X
TA5105-4DOT	X	X	X	X
TA5110-2DI2DOT	X	X	X	X
TA5120-2AI-UI [2])	X	X	X	X
TA5122-2AI-TC [2])	X	X	X	X
TA5123-2AI-RTD [2])	X	X	X	X
TA5126-2AO-UI [2])	X	X	X	X
TA5141-RS232I	X	X	X	X
TA5142-RS485I	X	X	X	X
TA5142-RS485	X	X	X	X

[1]) Can be used only once per CPU

[2]) In preparation, not yet available

The option board slots are not affected to one type of option board and they can be plugged and used on each slot. The only limitation is the number of slot available on the processor module. The following types of option board are available, all type can be mixed on all the slots.

Option board for COMx serial communication

Always needed for serial communication like Modbus RTU. Selection and configuration can be found into the PLC Configuration V3 documentation part: ↳ Chapter 1.6.6.2.8.3 “Attach an option board for COMx serial communication” on page 3721

Part no.	Description
1SAP 187 300 R0001	TA5141-RS232I: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 300 R0002	TA5142-RS485I: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch
1SAP 187 300 R0003	TA5142-RS485: AC500, option board for COMx serial communication, spring/cable front terminal 3.50 mm pitch

Option board for digital I/O extension

Selection and configuration can be found into the PLC Configuration V3 documentation part: ↳ Chapter 1.6.6.2.8.2 “Attach an option board for digital I/O extension” on page 3721

Part no.	Description
1SAP 187 000 R0001	TA5101-4DI: AC500, option board for digital I/O extension, 4DI 24 V DC, spring/cable front terminal 3.50 mm pitch
1SAP 187 000 R0002	TA5105-4DOT: AC500,option board for digital I/O extension, 4DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch
1SAP 187 000 R0003	TA5110-2DI2DOT: AC500, option board for digital I/O extension, 2DI 24 V DC, 2DO-T 24 V DC / 0.5 A, spring/cable front terminal 3.50 mm pitch

Option board for specific function

The TA5130-KNXPB can only be used on AC500-eCo V3 processor modules Pro PM5072-T-ETH(W).

The TA5131-RTC can only be used on AC500-eCo V3 processor modules Basic PM5012-x-ETH.

These two option boards can only be used once on one slot at a time!

Part no.	Description
1SAP 187 200 R0001	TA5130-KNXPB: AC500, option board KNX adress push button
1SAP 187 200 R0002	TA5131-RTC:AC500, real-time clock without battery, option board for AC500-eCo V3 Basic CPU

1.6.5.2 System technology of the AC500 communication modules

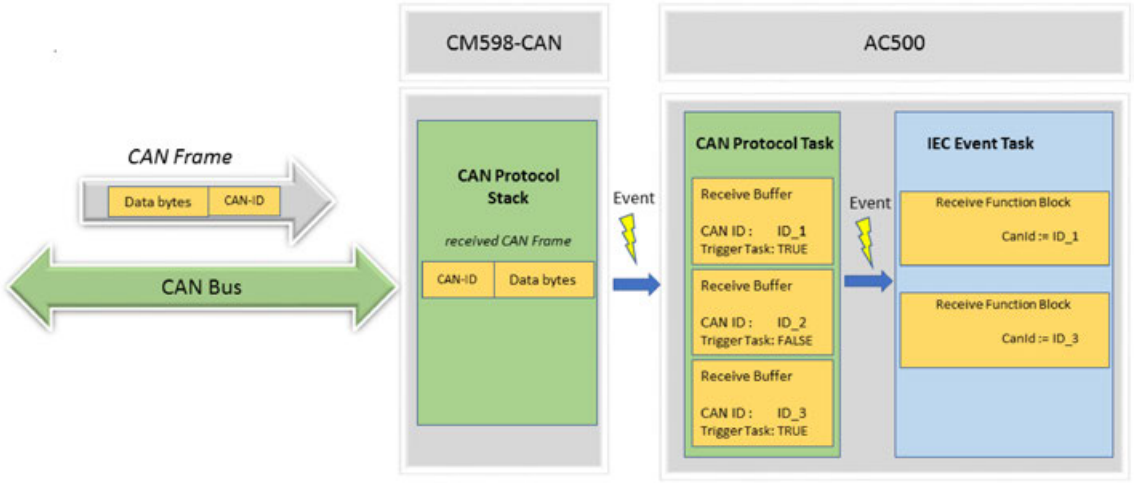
1.6.5.2.1 CANopen communication modules

Triggering of event tasks with CAN-IDs

For CM598-CAN module the execution of a PLC application task can be triggered automatically by a certain event, i.e. by incoming CAN 2.0 A or CAN 2.0 B frames. For this, the PLC application task is to be configured as external event task.

Prerequisites

- *PLC firmware version 3.2.5 and Automation Builder as of version 2.2.5.*
- *Only one PLC application task can be assigned to a communication module.*
- *Triggering of event tasks is only supported for the communication module CM598-CAN.*



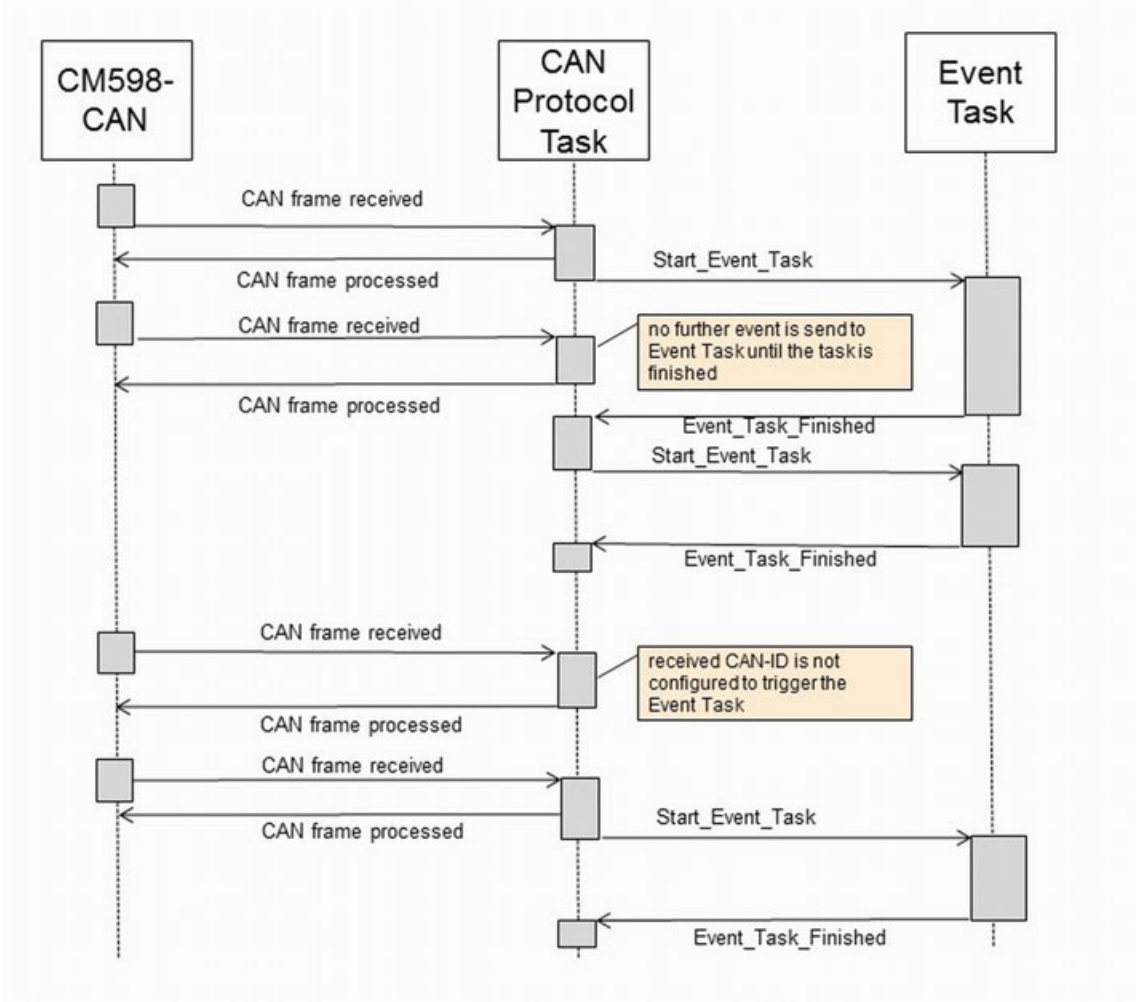
Every incoming CAN frame on a CM598-CAN module processes an event in the AC500 PLC. If the parameter "Trigger PLC Task" is set to TRUE, the CAN protocol task checks via the receive buffer configuration and the corresponding CAN-ID of the CAN frame whether a CAN frame is to be executed or not. Only those CAN-IDs that are configured in the protocol configuration will be processed. All other CAN frames will be rejected. If a CAN frame is to be processed, the CAN frame data is copied to the receive buffer and an event on the IEC event task is triggered.

> *The IEC event task will be executed for one cycle.*
>
> *The IEC event task will be triggered continuously until all associated receive buffers have been emptied. Hence, ensure that the buffers are emptied by the task, otherwise the task will run into a loop.*

Within the task the function block Cm598CanMsgRecEvt must be used to read the CAN frames from the receive buffers. The function block Cm598CanMsgRec is not suitable as it requires several task cycles for execution.

CAN frame processing

The following figure shows the sequence CAN frames processing when the triggering of event task is used.

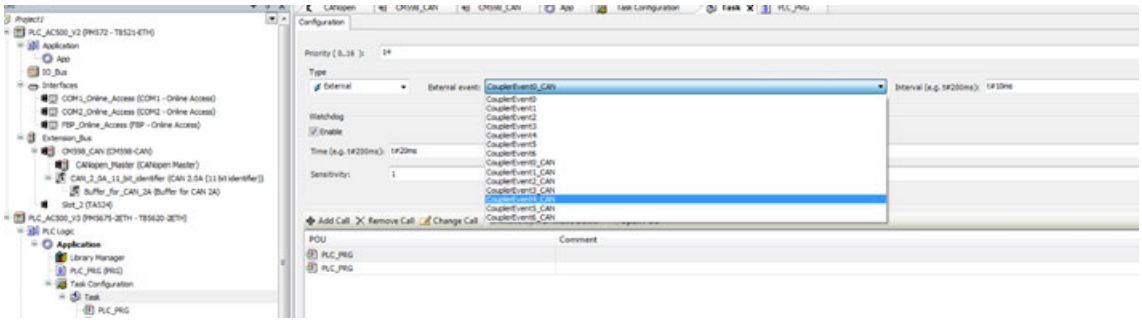


- *Only one external event task can be assigned to a CM598-CAN.*
- *There is only one common event for an external event task and all selected CAN-IDs. It must be evaluated which CAN-IDs have been received.*
- *It is possible that CAN frames are lost when necessary system resources are in use or when the CAN frames could not be processed in time due to high system load. So, the PLC application must monitor the task which consumes the events of the CAN protocol with a watchdog mechanism or something similar.*
- *Received CAN frames of the same CAN-ID are internally stored in FIFO buffers. Reading and writing of the FIFO buffers is not possible at the same time.*
- *Within an external event task the function block Cm598CanMsgRecEvt must be used to read the received CAN frames. The function block Cm598CanMsgRec is not suitable since its execution needs more than one task cycle.*
- *The CAN-IDs that are enabled to trigger an external event task must be read by the associated task. Otherwise the task is triggered again and again, and the CPU load will be high.*

Event task configuration

Add the external event task that should be executed to the task configuration of the PLC application:

1. Right-click on *“Task Configuration”*. Enter a name for the task and click *“Add object”*.
2. Right-click on the new task and append a *“Program Call”*. This contains the program code that is executed by the task.
3. Double-click on the task and setup the task parameters.



A parameter description is given in the ↳ *Chapter 1.4.1.20.2.27.1 “Tab 'Configuration'” on page 942*. Deviations are described in the following:

Parameter	Default	Value	Description
Priority	16	0..16 Value '0' indicates the highest priority	Priority of the task
Type	n.a.	External	Specifies the task type.
External Event	n.a.	CouplerEvent<slot index of the CM>_CAN	Specifies the event that triggers execution of the task.
Interval	n.a.	Cycle time	Not used

Configuration of a CM598-CAN module is desribed in the configuration chapter ↳ *Chapter 1.6.6.2.11.1 “CANopen” on page 3737*.

1.6.5.3 System technology of the communication interface modules

1.6.5.3.1 Modbus communication interface module

Overview

The Modbus TCP communication interface module CI52x-MODTCP is used as decentralized I/O module in Modbus TCP networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit.

CI521-MODTCP I/O channels properties:

- 4 analog inputs (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

Functionality

Parameter	Value
Interface	Ethernet
Protocol	Modbus TCP
Power supply	from the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the last BYTE of the IP (00h to FFh)
Analog inputs	4 (configurable via software)
Analog outputs	2 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)

CI522-MODTCP I/O channels properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

Functionality

Parameter	Value
Interface	Ethernet
Protocol	Modbus TCP
Power supply	from the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the last BYTE of the IP (00h to FFh)
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels.

The configuration of the inputs/outputs is performed by software.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Modbus TCP registers

Register layout for CI52x-MODTCP

The registers can be divided in 4 sections:

- Information data section 0x0000 to 0x0D50 (for acyclic use)
- I/O data and diagnosis section 0x0FFA to 0x2B00 (for cyclic use)
- Parameter data section 0x3000 to 0x3B00 (for acyclic use)
- Special functionality section 0x5A00 to 0x6A00 (for acyclic use)

Information data section (Acyclic data)

The information data section can be used to read out common and module specific information.

This section is read only.

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
0	Device and FW information CI	3	x
50	Production data CI	3	x
100	Device and FW information 1. EXP	3	x
125	Device and FW information 1. Hot swap terminal unit	3 *)	x
150	Production data 1. EXP	3	x
175	Production data 1. Hot swap terminal unit	3 *)	x
...	...		x
A00	Device and FW information 10. EXP	3	x
A25	Device and FW information 10. Hot swap terminal unit	3 *)	x
A50	Production data 10. EXP	3	x
A75	Production data 10. Hot swap terminal unit	3 *)	x
D00	Common device information	3	x

*) supported from CI52x firmware version V3.2.0 (device index F0)

This section can be divided again in two sections:

- The module specific section (containing information for each module CI52x-MODTCP and expansion modules and hot swap terminal units)
- The common device information block

Module specific information registers

For each module (CI52x device, expansion modules and hot swap terminal units) the following data can be read out:

- Device and FW information
  This section consists of 20 WORDs per module and contains information on each module using the following structure:

Data	DATA TYPE	Description
Module ID	WORD	The module ID of the requested module
Module name	ARRAY [1..10] OF BYTE	The module name of the requested module
Version 1st processor	ARRAY [1..4] OF BYTE	The version of the 1st processor of the requested module
Version 2nd processor	ARRAY [1..4] OF BYTE	The version of the 2nd processor of the requested module
Version 3rd processor	ARRAY [1..4] OF BYTE	The version of the 3rd processor of the requested module

Data	DATA TYPE	Description
Version 4th processor	ARRAY [1..4] OF BYTE	The version of the 4th processor of the requested module
Hardware version [1])	ARRAY [1..4] OF BYTE	The hardware version of the 4 processors
Reserved	ARRAY [1..8] OF BYTE ARRAY [1..4] OF BYTE [2])	Reserved
Number input data	WORD	Number of input data of the requested module in BYTES
Number output data	WORD	Number of output data of the requested module in BYTES

[1]) supported from CI52x firmware version V3.2.0 (device index F0)

[2]) from CI52x firmware version V3.2.0 (device index F0) “Reserved” is ARRAY [1..4] OF BYTE

- Production / Traceability data:
  This section consists of 25 WORDs per module and contains the traceability data for each module using following structure:
  – Article number: Byte 01..15
  – Index: Byte 16..17
  – Name: Byte 18..29
  – Production date: Byte 30..33
  – Key number: Byte 34..38
  – Site: Byte 39..40
  – Year: Byte 41..42
  – Serial number: Byte 41..50 (The serial number implies the year)
- Production / Traceability data from CI5x2 firmware version V3.2.0 (device index F0):
  This section consists of 26 WORDs per module and contains the traceability data for each module using following structure:
  – Article number: Byte 01..15
  – Index: Byte 16..17
  – Name: Byte 18..31
  – Production date: Byte 32..35
  – Key number: Byte 36..40
  – Site: Byte 41..42
  – Year: Byte 43..44
  – Serial number: Byte 42..52 (The serial number implies the year)

Common device information registers

Common device information block

This section consists of 80 WORDs (90 WORDs from CI52x firmware version V3.2.0 (device index F0)) and contains cluster wide information (CI52x device and connected expansion modules using the following structure:

Data	DATA TYPE	Description
Device state	BYTE	The actual state of the device: 0: STATE_PREOP (device booting) 1: STATE_OPERATION (device in operational, no bus supervision active) 2: STATE_ERROR (device detected a bus error, bus supervision active) 3: STATE_IP_ERROR (the device has a IP address error) 4: STATE_CYCLIC_OPERATION (device in operational, bus supervision active)
Parameter state	BYTE	The actual parameter state of the device: 0: PARA_STATE_NO_PARA (the device has no parameters) 1: PARA_STATE_PARA_ACTIVE (parameterization process running) 2: PARA_STATE_PARA_DONE (the uses valid parameters) 3: PARA_STATE_ERROR (The device has invalid
Module ID CI device	WORD	Module ID of the CI52x device itself
Module ID 1st expansion	WORD	Module ID of the 1st connected expansion module
Module ID 2nd expansion	WORD	Module ID of the 2nd connected expansion module
...		
Module ID 10th expansion	WORD	Module ID of the 10th connected expansion module
Expansion bus error count	DWORD	Global telegram error count over all expansion modules
Good count onboard I/O	DWORD	Telegram good count onboard I/Os
Good count 1st expansion	DWORD	Telegram good count 1st expansion module
Good count 2nd expansion	DWORD	Telegram good count 2nd expansion module
...		
Good count 10th expansion	DWORD	Telegram good count 10th expansion module
Error count onboard I/O	DWORD	Telegram error count onboard I/Os
Error count 1st expansion	DWORD	Telegram error count 1st expansion module
Error count 2nd expansion	DWORD	Telegram error count 2nd expansion module
...		
Error count 10th expansion	DWORD	Telegram error count 10th expansion module
Input address onboard I/O	WORD	Modbus TCP register address for inputs of the onboard I/Os
Input address 1st expansion	WORD	Modbus TCP register address for inputs of the 1st expansion module
Input address 2nd expansion	WORD	Modbus TCP register address for inputs of the 2nd expansion module

Data	DATA TYPE	Description
...		
Input address 10th expansion	WORD	Modbus TCP register address for inputs of the 10th expansion module
Output address onboard I/O	WORD	Modbus TCP register address for outputs of the onboard I/Os
Output address 1st expansion	WORD	Modbus TCP register address for outputs of the 1st expansion module
Output address 2nd expansion	WORD	Modbus TCP register address for outputs of the 2nd expansion module
...		
Output address 10th expansion	WORD	Modbus TCP register address for outputs of the 10th expansion module
Module ID 1st hot swap terminal unit *)	WORD	Module ID of the 1st connected hot swap terminal unit *)
Module ID 2nd hot swap terminal unit *)	WORD	Module ID of the 2nd connected hot swap terminal unit *)
...		
Module ID 10th hot swap terminal unit *)	WORD	Module ID of the 10th connected hot swap terminal unit *)

*) supported from CI52x firmware version V3.2.0 (device index F0)

I/O / Process data and diagnosis section (Cyclic data)

Table 650: The cyclic data section for CI52x-MODTCP

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
FCE *)	Module state	3,4, 23	x
FFA	Diagnosis	3,4, 23	x
1000	Inputs CI	3, 4, 23	x
1100	Inputs 1.EXP	3, 4, 23	x
...	...		x
1A00	Inputs 10.EXP	3, 4, 23	x
2000	Outputs CI	3, 23	6, 16, 23
2100	Outputs 1.EXP	3, 23	6, 16, 23
...	...		
2A00	Outputs 10.EXP	3, 23	6, 16, 23
2B00	Dummy output	3, 23	6, 16, 23

*) supported from CI52x firmware version V3.2.0 (device index F0)

This section can be divided again in three sections:

- Module state (containing the state of connected expansion modules and hot swap terminal units)
- Diagnosis data (containing diagnosis data in AC500 specific format)
- Process data (containing I/O data)

Module state

The module state section consists of 44 WORDs and contains the module state of connected expansion modules and hot swap terminal units using the following structure:

Data	DATA TYPE	Description
Module ID	WORD	Module ID of the CI52x
Expected module ID	WORD	Expected (configured) module ID of the CI52x
Module state	BYTE	The current module state of the CI52x: 0: NO_MOD (no module detected) 1: MOD_INIT (module detected, module is in initialization phase) 2: MOD_RUN (module detected and running or in failsafe state, input data are valid) 3: WRONG_MOD (wrong module detected, module ID doesn't match expected module ID) 4: MOD_REMOVED (module removed or defective on hot swap terminal unit, no communication to module possible) 5: MOD_ERROR (module defective on hot swap terminal unit, no communication to module possible) 6: MOD_LOST (lost communication to module on not hot swap capable terminal unit) 7: UNKNOWN (module detected but not configured)
Diagnosis flag	BYTE	Diagnosis flag for the CI52x: 0: NO_DIAG (no diagnosis available from CI52x I/O cards) 1: DIAG_AVAILABLE (diagnosis available for CI52x I/O cards)
Terminal unit state	BYTE	Terminal unit state for the CI52x: 0: NO_HOTSWAP_TU (not hot swap terminal unit detected) 1: HOTSWAP_TU_RUNNING (hot swap terminal unit detected and working) 2: HOTSWAP_TU_ERROR (hot swap terminal unit detected, but communication errors for hot swap terminal unit detected)
Parameter state	BYTE	Parameter state of the CI52x: 0: NO_PARA (module is in initialization phase and not ready for parameterization) 1: WAIT_PARA (module awaits parameterization) 2: PARA_RUN (parameterization running) 3: LEN_ERR (length of parameters not correct) 4: ID_ERR (module ID inside parameters not correct) 5: PARA_DONE (parameterization finished without errors)
Module ID	WORD	Module ID of the 1st connected expansion module

Data	DATA TYPE	Description
Expected module ID	WORD	Expected (configured) module ID of the 1st connected expansion module
Module state	BYTE	The current module state of the 1st connected expansion module
Diagnosis flag	BYTE	Diagnosis flag for the 1st connected expansion module 0: NO_DIAG (no diagnosis available for expansion module) 1: DIAG_AVAILABLE (diagnosis available for expansion module)
Terminal unit state	BYTE	Terminal unit state for the 1st connected expansion module
Parameter state	BYTE	Parameter state of the 1st connected expansion module
...		
Module ID	WORD	Module ID of the 10th connected expansion module
Expected module ID	WORD	Expected (configured) module ID of the 10th connected expansion module
Module state	BYTE	The current module state of the 10th connected expansion module
Diagnosis flag	BYTE	Diagnosis flag for the 10th connected expansion module
Terminal unit state	BYTE	Terminal unit state for the 10th connected expansion module
Parameter state	BYTE	Parameter state of the 10th connected expansion module

Diagnosis data

The diagnosis data section contains one diagnosis message with the following structure (according to AC500 diagnosis):

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI52x-MODTCP (e. g. error at integrated 8 DI / 8 DO)
		1 = 1st connected S500 I/O Module
		...
		10 = 10th connected S500 I/O Module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and Bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to Bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error
6	Reserved	0

If a diagnosis message is read out, the next one will be automatically filled in.

If no more diagnosis messages are available the buffer will be reset to zero.

This ensures that each diagnosis message can be delivered to the Modbus TCP client/slave and no diagnosis will be lost.

I/O data

The I/O data section can use two different formats according to the module parameter “I/O Mapping Structure” (see ↬ *Chapter 1.6.3 “Device specifications” on page 2430* for details).

- Fixed I/O mapping
  In case of fixed I/O mapping each module has a predefined register range for each Inputs and Outputs.
- Dynamic I/O mapping
  In case of dynamic I/O mapping the mapping is build according to the actual configuration.

The dummy output at the end of the I/O data section can be used to retrigger the bus supervision and has no effect on the HW outputs.

Fixed I/O mapping

In case of fixed I/O mapping the following predefined register table is used:

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
1000	Inputs CI	3, 4, 23	x
1100	Inputs 1.EXP	3, 4, 23	x
...	...		x
1A00	Inputs 10.EXP	3, 4, 23	x
2000	Outputs CI	3, 23	6, 16, 23
2100	Outputs 1.EXP	3, 23	6, 16, 23
...	...		
2A00	Outputs 10.EXP	3, 23	6, 16, 23
2B00	Dummy output	3, 23	6, 16, 23

If a certain expansion module has no inputs or outputs the corresponding registers remain empty.

Dynamic I/O mapping

In case of dynamic mapping only the start addresses of inputs and outputs are predefined:

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
1000	Inputs CI	3, 4, 23	x
...	...		x
2000	Outputs CI	3, 23	6, 16, 23
...	...		
2B00	Dummy output	3, 23	6, 16, 23

The register addresses of the connected expansion modules are calculated dynamically based on the number of inputs and outputs of the previous modules (each module starts directly on the next register after the previous module).

The register addresses of each module can be read out via the common device register (see ↳ *Chapter 1.6.5.3.1.2.2.2 “Common device information registers” on page 3606*).

Comparative example

The difference between fixed I/O mapping and dynamic I/O mapping is shown in the following table.

For this comparison a cluster with CI522, AX522, DC532, AX521, DC523, DC532, AO523, AI523, DI524, AX522 and DC523 is used.

Fixed Mapping					Dynamic Mapping			
Register (hex)	**Description**	**Type**	**Data**		**Register (hex)**	**Description**	**Type**	**Data**
1000	Inputs CI	8 DC, 8 DI, FC	4 BYTE + 4 WORD		1000	Inputs CI	8 DC, 8 DI, FC	4 BYTE + 4 WORD
1100	Inputs AX522	8 AI	8 WORD		1006	Inputs AX522	8 AI	8 WORD
1200	Inputs DC532	16 DI, 16 DC	4 BYTE		100E	Inputs DC532	16 DI, 16 DC	4 BYTE
1300	Inputs AX521	4 AI	4 WORD		1010	Inputs AX521	4 AI	4 WORD
1400	Inputs DC523	24 DC	3 BYTE		1014	Inputs DC523	24 DC	3 BYTE
1500	Inputs DC532	16 DI, 16 DC	4 BYTE		1016	Inputs DC532	16 DI, 16 DC	4 BYTE
1600	Inputs AO523	---	---		---	Inputs AO523	---	---
1700	Inputs AI523	16AI	16 WORD		1018	Inputs AI523	16AI	16 WORD
1800	Inputs DI524	32 DI	4 BYTE		1028	Inputs DI524	32 DI	4 BYTE
1900	Inputs AX522	8 AI	8 WORD		102A	Inputs AX522	8 AI	8 WORD
1A00	Inputs DC523	24 DC	3 BYTE		1032	Inputs DC523	24 DC	3 BYTE
2000	Outputs CI	8 DC, 8DO, FC	4 BYTE + 8 WORD		2000	Outputs CI	8 DC, 8DO, FC	4 BYTE + 8 WORD
2100	Outputs AX522	8 AO	8 WORD		200A	Outputs AX522	8 AO	8 WORD
2200	Outputs DC532	16 DC	2 BYTE		2012	Outputs DC532	16 DC	2 BYTE
2300	Outputs AX521	4 AO	4 WORD		2013	Outputs AX521	4 AO	4 WORD
2400	Outputs DC523	24 DC	3 BYTE		2017	Outputs DC523	24 DC	3 BYTE
2500	Outputs DC532	16 DC	2 BYTE		2019	Outputs DC532	16 DC	2 BYTE
2600	Outputs AO523	16 AO	16 WORD		201A	Outputs AO523	16 AO	16 WORD
2700	Outputs AI523	---	---		---	Outputs AI523	---	---
2800	Outputs DI524	---	---		---	Outputs DI524	---	---
2900	Outputs AX522	8 AO	8 WORD		202A	Outputs AX522	8 AO	8 WORD
2A00	Outputs DC523	24 DC	3 BYTE		2032	Outputs DC523	24 DC	3 BYTE

Process data structure CI521-MODTCP

When commissioning a CI521 module with byte order "big endian" in combination with a V3 PLC.

Table 651: I/O data (Inputs 19 BYTEs)

Signal	DATA TYPE	Description
AI0	WORD	Input value of the 1st analog input
AI1	WORD	Input value of the 2nd analog input
AI2	WORD	Input value of the 3rd analog input
AI3	WORD	Input value of the 4th analog input
Additional reserve byte	BYTE	reserved, not used
DI	BYTE	Input value of the DI channels
Fast counter actual value counter 1	DWORD	*Chapter 1.6.5.1.12.1 "Fast counters in AC500 devices" on page 3570*
Fast counter actual value counter 2	DWORD	
Fast counter state counter 1	BYTE	
Fast counter state counter 2	BYTE	

Table 652: I/O data (Outputs 23 BYTEs)

Signal	DATA TYPE	Description
AO0	WORD	Output value of the 1st analog output
AO1	WORD	Output value of the 2nd analog output
Additional reserve byte	BYTE	reserved, not used
DO	BYTE	Output value of the DO channels
Fast counter start value counter 1	DWORD	*Chapter 1.6.5.1.12.1 "Fast counters in AC500 devices" on page 3570*
Fast counter end value counter 1	DWORD	
Fast counter start value counter 2	DWORD	
Fast counter end value counter 2	DWORD	
Fast counter control counter 1	BYTE	
Fast counter control counter 2	BYTE	

Process Data Structure CI522-MODTCP

When commissioning a CI522 module with byte order "big endian" in combination with a V3 PLC.

Table 653: I/O data (Inputs 12 BYTEs)

Signal	DATA TYPE	Description
DI	BYTE	Input value of the DI channels
DC	BYTE	Input value of the DC channels
Fast counter actual value counter 1	DWORD	↳ *Chapter 1.6.5.1.12.1 “Fast counters in AC500 devices” on page 3570*
Fast counter actual value counter 2	DWORD	
Fast counter state counter 1	BYTE	
Fast counter state counter 2	BYTE	

Table 654: I/O data (Outputs 20 BYTEs)

Signal	DATA TYPE	Description
DO	BYTE	Output value of the DO channels
DC	BYTE	Output value of the DC channels
Fast counter start value counter 1	DWORD	↳ *Chapter 1.6.5.1.12.1 “Fast counters in AC500 devices” on page 3570*
Fast counter end value counter 1	DWORD	
Fast counter start value counter 2	DWORD	
Fast counter end value counter 2	DWORD	
Fast counter control counter 1	BYTE	
Fast counter control counter 2	BYTE	

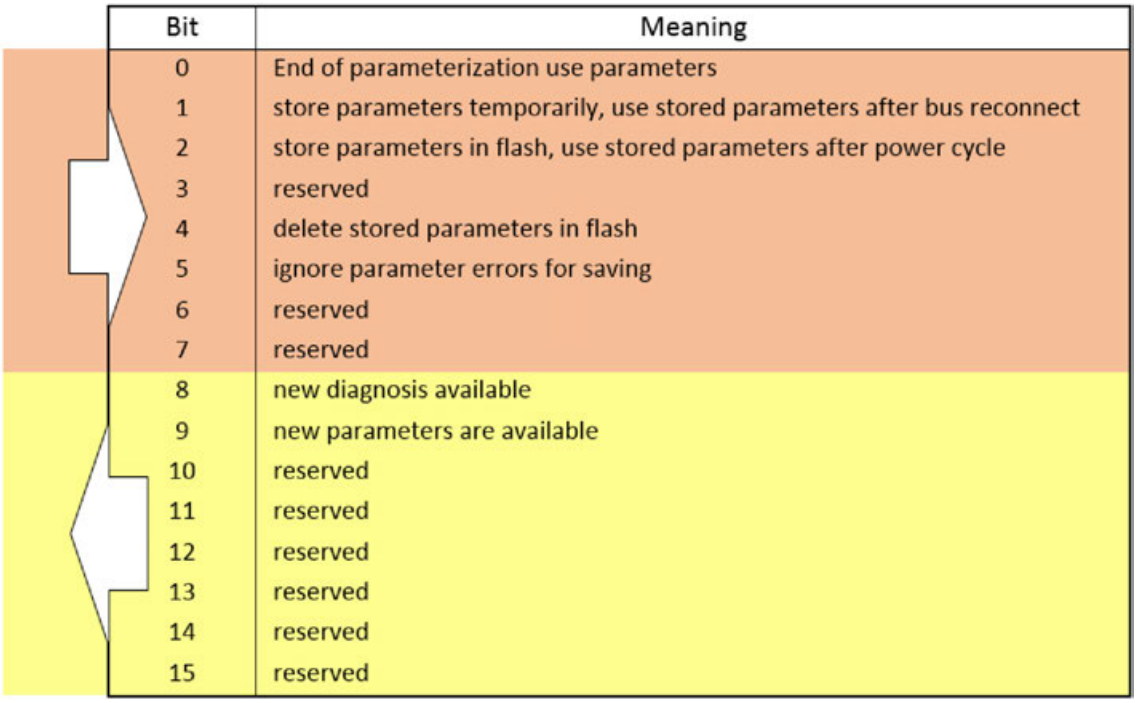
Parameter data (Acyclic data)

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
3000	Parameters CI	3	6, 16
3080	Stored parameters CI	3	x
3100	Parameters 1. EXP	3	6, 16
3180	Stored parameters 10. EXP	3	x
...			
3A00	Parameters 10. EXP	3	6, 16
3A80	Stored parameters 10. EXP	3	x
3B00	controlword/statusword	3	6, 16

For each connected module the following parameter data are defined (the parameters are represented as ARRAY OF BYTE):

- Actual used parameter for each module
  In these sections the actual parameters are stored. This section is also used to write parameters to the module (For a description on how to parameterize see *Chapter 1.6.5.3.1.3.2 “Parameterization” on page 3622*).
- Stored parameters for each module
  If the module has stored nonvolatile parameters these can be read out using the corresponding registers.

The controlword/statusword is used to trigger a parameterization process. The single bits have the following meaning:



Bit	Meaning
0	End of parameterization use parameters
1	store parameters temporarily, use stored parameters after bus reconnect
2	store parameters in flash, use stored parameters after power cycle
3	reserved
4	delete stored parameters in flash
5	ignore parameter errors for saving
6	reserved
7	reserved
8	new diagnosis available
9	new parameters are available
10	reserved
11	reserved
12	reserved
13	reserved
14	reserved
15	reserved

The direction of the first 8 bits is client to server (master to slave).

The direction of the second 8 bits is server to client (slave to master). A description of the bits can be found in chapter behavior *Chapter 1.6.5.3.1.3.2 “Parameterization” on page 3622*.

The parameter register sections (actual and stored parameters) have the structure as explained in the of the corresponding module *Chapter 1.6.3 “Device specifications” on page 2430*.

Short description of the CI521-MODTCP parameters

Parameter	Single parameter index	Description	Additional Info
0		Module ID (high Byte)	Fixed, must be 16#1C
1		Module ID (low Byte)	Fixed, must be 16#E8
2		Ignore Module	Reserved, must be 0
3		Length of following parameter block	Fixed, must be 16#3F
4	0	Error LED / Failsafe	See *Chapter 1.6.3 “Device specifications” on page 2430*

Parameter	Single parameter index	Description	Additional Info
5	1	Master IP Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
6		Master IP Byte 1	
7		Master IP Byte 2	
8		Master IP Byte 3	
9	2	Master IP 1 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
10		Master IP 1 Byte 1	
11		Master IP 1 Byte 2	
12		Master IP 1 Byte 3	
13	3	Master IP 2 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
14		Master IP 2 Byte 1	
15		Master IP 2 Byte 2	
16		Master IP 2 Byte 3	
17	4	Master IP 3 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
18		Master IP 3 Byte 1	
19		Master IP 3 Byte 2	
20		Master IP 3 Byte 3	
21	5	Master IP 4 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
22		Master IP 4 Byte 1	
23		Master IP 4 Byte 2	
24		Master IP 4 Byte 3	
25	6	Master IP 5 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
26		Master IP 5 Byte 1	
27		Master IP 5 Byte 2	
28		Master IP 5 Byte 3	
29	7	Master IP 6 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
30		Master IP 6 Byte 1	
31		Master IP 6 Byte 2	
32		Master IP 6 Byte 3	
33	8	Master IP 7 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
34		Master IP 7 Byte 1	
36		Master IP 7 Byte 2	
36		Master IP 7 Byte 3	
37	9	Timeout	Timeout for bus supervision in 10ms steps if set to 0 no bus supervision is active
38	10 (read only)	I/O Mapping Structure	See *Chapter 1.6.3 “Device specifications” on page 2430*
39	11	Reserved	Reserved, must be 0
40	12	Reserved	Reserved, must be 0

Parameter	Single parameter index	Description	Additional Info
41	13	Reserved	Reserved, must be 0
42	14	Check supply	See *Chapter 1.6.3 "Device specifications" on page 2430*
43	15	Analog data format	Reserved, must be 0
44	16	Input delay	See *Chapter 1.6.3 "Device specifications" on page 2430*
46	17	Fast counter	
46	18	Short circuit detection	
47	19	Behavior binary outputs at com. fault	
48	20	Substitute value binary outputs	
49	21	Overvoltage monitoring	
50	22	Behavior analog outputs	
51	23	Channel Config AI0	
52	24	Check Channel AI0	
53	25	Channel Config AI1	
54	26	Check Channel AI1	
55	27	Channel Config AI2	
56	28	Check Channel AI2	
57	29	Channel Config AI3	
58	30	Check Channel AI3	
59	31	Channel Config AO0	
60	32	Check Channel AO0	
61	33	Substitute value AO0 (high Byte)	
62		Substitute value AO0 (low Byte)	
63	34	Channel Config AO1	
64	35	Check Channel AO1	
65	36	Substitute value AO1 (high Byte)	
66		Substitute value AO1 (low Byte)	

Short description of the CI522-MODTCP parameters

Parameter	Single parameter index	Description	Additional Info
0		Module ID (high Byte)	Fixed, must be 16#1C
1		Module ID (low Byte)	Fixed, must be 16#ED
2		Ignore Module	Reserved, must be 0
3		Length of following parameter block	Fixed, must be 16#2F
4	0	Error LED / Failsafe	See *Chapter 1.6.3 "Device specifications" on page 2430*

Parameter	Single parameter index	Description	Additional Info
5	1	Master IP Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
6		Master IP Byte 1	
7		Master IP Byte 2	
8		Master IP Byte 3	
9	2	Master IP 1 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
10		Master IP 1 Byte 1	
11		Master IP 1 Byte 2	
12		Master IP 1 Byte 3	
13	3	Master IP 2 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
14		Master IP 2 Byte 1	
15		Master IP 2 Byte 2	
16		Master IP 2 Byte 3	
17	4	Master IP 3 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
18		Master IP 3 Byte 1	
19		Master IP 3 Byte 2	
20		Master IP 3 Byte 3	
21	5	Master IP 4 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
22		Master IP 4 Byte 1	
23		Master IP 4 Byte 2	
24		Master IP 4 Byte 3	
25	6	Master IP 5 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
26		Master IP 5 Byte 1	
27		Master IP 5 Byte 2	
28		Master IP 5 Byte 3	
29	7	Master IP 6 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
30		Master IP 6 Byte 1	
31		Master IP 6 Byte 2	
32		Master IP 6 Byte 3	
33	8	Master IP 7 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 3624*)
34		Master IP 7 Byte 1	
36		Master IP 7 Byte 2	
36		Master IP 7 Byte 3	
37	2	Timeout	Timeout for bus supervision in 10ms steps if set to 0 no bus supervision is active
38	3 (read only)	I/O Mapping Structure	See *Chapter 1.6.3 “Device specifications” on page 2430*
39	4	Reserved	Reserved, must be 0
40	5	Reserved	

Parameter	Single parameter index	Description	Additional Info
41	6	Reserved	
42	7	Check supply	See *Chapter 1.6.3 "Device specifications" on page 2430*
43	8	Input delay	
44	9	Fast counter	See *Chapter 1.6.3 "Device specifications" on page 2430*
46	10	Short circuit detection	
46	11	Behavior binary outputs at com. fault	
47	12	Substitute value binary outputs (high byte)	
48		Substitute value binary outputs (low byte)	
49	13	Voltage feedback monitoring	
50	14	Overvoltage monitoring	

Parameters of connected expansion modules

The parameters of the connected expansion modules are represented as byte array (the parameters valid for "CPU" in the *Chapter 1.6.3 "Device specifications" on page 2430* of the corresponding module are used):

Parameter	Description	Additional Info
0	Module ID (high byte)	Fixed, see *Chapter 1.6.3 "Device specifications" on page 2430* of corresponding module (the module ID of FBP is used)
1	Module ID (low byte)	Fixed, see of corresponding module (the module ID of FBP is used) *Chapter 1.6.3 "Device specifications" on page 2430*
2	Ignore module	Reserved must be 0
3	Length of following parameter block	Fixed, see *Chapter 1.6.3 "Device specifications" on page 2430* of corresponding module
4...	The rest of the parameter are described in the corresponding module	

Special functionality

This section contains special services like firmware update or single parameterization.

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
4000	Firmware download	3	16
4100	Firmware download state	3	x
5000	Write single parameterization of CI	x	16
5100	Write single parameterization of 1. EXP	x	16
...			
5A00	Write single parameterization of 10. EXP	x	16
6000	Read single parameterization of CI	3	16
6100	Read single parameterization of 1. EXP	3	16
...			
6A00	Read single parameterization of 10. EXP	3	16

Behavior

IP address assignment

The delivery IP address of the CI52x-MODTCP is 192.168.0.xx (xx is the hardware address switch position of the device.

The devices support BOOTP, DHCP and fixed IP address setting (these can be set individual or together). If BOOTP and DHCP are enabled the following priority takes place:

- If DHCP configuration fails, the device will fall back to BOOTP.
- In case of a BOOTP failure, the fixed IP address will be used.

A new IP address (or changing of BOOTP and DHCP) can be set in two different ways:

- With the address switches of the corresponding module
- With the ↬ *Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675*

Using the address switches

With the address switches only the last byte of the IP address can be changed.

The IP address can only be set via the address switches in case of factory default or in case of the last byte of the IP address is set to zero with the ↬ *Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675*. The not allowed IP addresses are mapped as followed:

- Address switch position 255 is mapped to fixed IP 192.168.0.254 independent of other stored settings (by IP Configuration Tool).

  This is a backup so the module can always get a valid IP address and can be configured by the IP Configuration Tool.
- Address switch position 0 is mapped to last byte equal 1 and DHCP enabled.

Using the IP configuration tool

With the *Chapter 1.6.6.2.2.4.2 “Configuration of the IP settings with the IP configuration tool” on page 3675* a network scan can be executed, and the found devices can be assigned with new settings, e.g. enable BOOTP or DHCP and set a new fixed IP. If the last byte of the IP address of the CI52x-MODTCP devices is set to 0 with the IP Configuration Tool the address switch position is used instead (see *Chapter 1.6.5.3.1.3.1.1 “Using the address switches” on page 3621*).

Parameterization

The parameterization is done via the corresponding registers explained in the Modbus TCP registers *Chapter 1.6.5.3.1.2.4 “Parameter data (Acyclic data)” on page 3615*.

In addition to that the parameters can be directly transferred via Automation Builder (see documentation of Automation Builder for that).

There are two different parameter sections with different behavior.

Actual used parameters

After startup this section contains the following data:

- Default parameters (only module id and parameter length set all others zero) if no valid stored parameters are available (no or invalid parameters stored).
- Actual used / stored parameters if valid parameters are stored nonvolatile.

These parameters can be read out and changed by reading or writing of the corresponding registers, but will not be used automatically after writing them, the use of new written parameters has to be triggered by writing the parameter control word with the corresponding bits set (see below).

Stored parameters

This section always contains a copy of the nonvolatile stored parameters, if no parameters are stored nonvolatile this sections will be 0.

Controlword/statusword parameter

This parameter can be used to trigger and save new parameters.

The direction of the first 8 bit is client to server (master to slave). The direction of the second 8 bits is server to client (slave to master).

Bit	Description	
0	Use parameters / start parameterization	If this bit is set the CI Device starts the parameterization with the parameters in the actual parameters registers.
1	Store parameters volatile	If this bit is set the CI device will use the parameters temporarily, which means after a bus error detection and reconnection the parameters will be used again. **This bit should always be set.** **This bit is only evaluated when bit 0 is set.**
2	Store parameters nonvolatile	If this bit is set the CI device will store the parameters nonvolatile, which means after a power cycle the stored parameter data will be used again. **This bit is only evaluated when bit 0 is set.**
3	Reserved	-
4	Delete nonvolatile stored parameters	If this bit is set the CI device will delete its nonvolatile stored parameters. **This bit is only evaluated when bit 0 is set**.
5	Ignore parameter error for nonvolatile parameter storage	If this bit is set a parameter error during nonvolatile storage of parameters will be ignored, and the parameters will be stored. **This bit can only be set in combination with bit 0 and bit 2.**
6	Reserved	-
7	Reserved	-
8	New diagnosis available	The device will set this bit if new diagnosis data are available in the diagnosis data section.
9	New parameters available	The device will set this bit if new parameters are available in the actual parameter data section and these were not activated by setting bit 0 in the control word.
10...15	Reserved	-

Cyclic I/O data exchange

The I/O data can be exchanged cyclic by the master by reading, writing the corresponding registers.

I/O data exchange is only possible after successful parameterization of the device.

For writing of outputs **bus failure detection** can be activated by setting the corresponding parameter. This bus failure detection is described in the following chapter.

Bus failure detection

If the parameter "*"timeout"*" in the module parameters of the CI52x-MODTCP is set, the module will supervise the Modbus TCP "write telegrams".

After the first "write telegram" the bus will be supervised. If no new "write telegram" arrives at the CI52x-MODTCP within the configured time, the module will detect a bus failure and switch off its outputs or switch them to the configured failsafe state (see module parameter CI521 ↳ *Chapter 1.6.3.7.4.1.7 "Parameterization" on page 3176* and CI522 ↳ *Chapter 1.6.3.7.4.2.7 "Parameterization" on page 3206* for details).

Configurable write restriction

With the module parameters “Master IP”- “Master IP 7” it is possible to set write restrictions on the CI52x-MODTCP device.

If none of the parameters is set, all masters / clients in the network have read and write rights on the CI52x-MODTCP device and its connected expansion modules.

If at least one parameter is set only the configured masters / clients have write rights on the CI52x-MODTCP device.

All other masters / clients still have read access to the CI52x-MODTCP device.

Diagnosis behavior

Each diagnosis message signals if this error is coming or going , so it is possible to create a list in the master of actual pending diagnosis.

Diagnosis messages will be transferred again after a bus failure detection and reconnection.

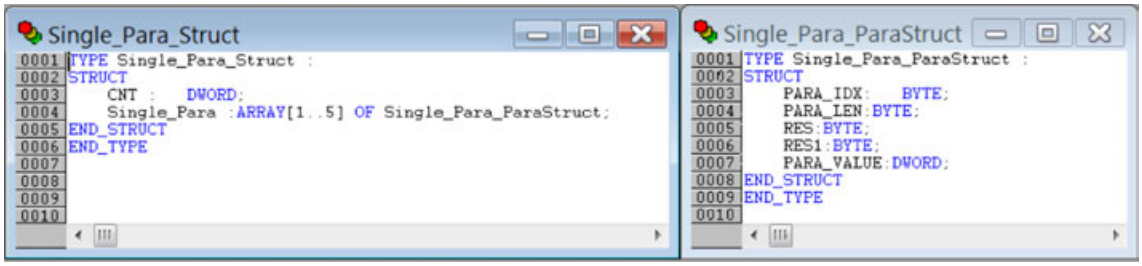
Diagnosis messages can be read out with function code 3,4,23. Function codes 3 and 4 can always read out diagnosis messages, function code 23 can only read out after successful parameterization of the device. See also table *Chapter 1.6.5.3.1.2.3.2 “Diagnosis data” on page 3611*.

Single parameterization

The single parameterization services can be used to read or write parameters during run time of device without the need of triggering a new parameterization process.

For indexes used for single parameterization services see parameter lists in section Modbus TCP registers of this document.

The read and write parameterization services are explained below, for each module (CI52x-MODTCP and connected expansion modules) a different section for read and write is defined see chapter Modbus TCP registers in this document). Both services are using the following data structure:



The length of the read / write service depends on the count of parameters that should be transferred (length = 4+ count*8).

Reading of single parameters

The read single parameterization works in two steps:

- Writing of a request list containing the indexes that should be read using the structure explained above.
  Only CNT and PARA_IDX has to be set.
  Up to 5 parameters can be requested with one telegram.
  The length of the write service depends on the count of parameters that should be transferred (length = 4+ count*8).
- Reading of the parameters list with the same length then the previous write request.
  If the internal reading process inside the CI52x-MODTCP device is done the data will be read out.
  If the internal reading process inside the CI52x-MODTCP device is not yet finished the read service will be rejected with Modbus TCP exception code 6 (device busy).

Writing of single parameters

For writing of single parameters only one step is necessary, the parameters are transferred with one write request using the structure described above.

The length of the write service depends on the count of parameters that should be transferred (length = 4+ count*8).

In case of write of single parameters the following values have to be set:

- CNT: number of parameters to be set
- And for each parameter:
  Parameter index
  Parameter length
  New parameter value

Written single parameters are not stored volatile and not stored nonvolatile. That means after a bus reconnection or power cycle the written parameters will be discarded.

Commissioning example

Set IP Address:

- The setting of the IP address is the first step to integrate the CI52x-MODTCP devices into a running system.
- The setting of the IP address of the CI52x-MODTCP devices is described in the chapter *Chapter 1.6.5.3.1.3.1 “IP address assignment” on page 3621* in this document.

Set Parameters (optional read parameters):

- The second step in configuring the CI52x-MODTCP devices is to set the module and channel parameters.
- A read of parameters is optional but can be used the get the module IDs and the parameter length.
- The reading and or writing of parameters is described in chapter *Chapter 1.6.5.3.1.3.2 “Parameterization” on page 3622*.

Set Control Word:

- After setting the parameter data these have to be activated by writing the control word.
- The meaning and usage of the control word is described in chapter *Chapter 1.6.5.3.1.3.2 “Parameterization” on page 3622*.

Exchange data:

- After setting and activating the parameters the CI52x-MODTCP device is ready for data exchange.
- The registers for data exchange are described in chapter *Chapter 1.6.5.3.1.2.3 “I/O / Process data and diagnosis section (Cyclic data)” on page 3608*.

Hot swap

With hot swap for AC500 and S500 it is possible to exchange expansion modules (with same type) during run time.

Preconditions for using hot swap

Information about preconditions for using hot swap see *“Hot swap” on page 3523*.

Compatibility of hot swap

	Modbus remote I/O
I/O module on TU5xx-H connected to I/O bus master	CI521-MODTCP or CI522-MODTCP
Required version of I/O bus master	Module index as of F0 Firmware as of V3.2.3
Fieldbus master when used as remote I/O with AC500 V3	Any AC500 V3 CPU with on-board Ethernet
When used as remote I/O on third party controller (PLC or DCS)	No limitation known

Hot swap behavior

The following table describes the behavior in case of I/O attached to communication interface module for Modbus TCP, CI521-MODTCP or CI522-MODTCP.

Hot Swap Behavior	Modbus TCP remote I/O
Start-up behavior with missing or damaged I/O module on hot swap terminal unit TU5xx-H	Remote I/O station is not starting As of device index F4 and Automation Builder Version 2.4.1 it is possible to configure the startup in case of missing modules on hot swap terminal units. If configured, the remote I/O station is starting up with missing or damaged I/O module, if the module is plugged later or replaced it will be automatically parameterized and I/O data will be exchanged. As the Automation Builder checks that all modules are available during configuration process, it is necessary that all I/O modules are available and in working order during configuration via Automation Builder. As the parameters are stored nonvolatile inside the CI52x devices later one the parameters have effect for power cycle or reconnection operations.
Start-up behavior with wrong I/O module type on any terminal unit	Remote I/O station is not starting
Diagnosis of presence of hot swap terminal unit	Information is available in Modbus registers of the communication interface module which can be accessed by the application program As of device index F4 and Automation Builder Version 2.4.1 it is possible to configure a list of required hot swap terminal units. If a required hot swap terminal unit is missing (normal one plugged) this will not prevent a normal operation but a diagnosis message will be generated for the corresponding slot.
Diagnosis of hot swap capability of I/O module mounted on hot swap terminal unit	Information can be obtained by reading Modbus registers in the communication interface module. Those Modbus registers contain: ● Diagnosis in case that a not hot-swappable I/O module is plugged on a hot swap terminal unit ● Diagnosis In case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration ● Production data and version index of the modules
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	Diagnosis is available in Modbus registers in the communication interface module
Input state in process image of controller while module is pulled or module is not operational	Input = ZERO
Diagnosis after plugging the I/O module on the hot swap terminal unit	Diagnose "diagnosis gone" is available in Modbus registers in the communication interface module

System behavior

If an expansion module is removed or defective during run time, the input data of this module will be set to “0” and the module state will be set to the corresponding value (see *Chapter 1.6.5.3.1.2.3 “I/O / Process data and diagnosis section (Cyclic data)” on page 3608*). A diagnosis message will be created in that case (see hardware description of *Chapter 1.6.3.7.4.1 “CI521-MODTCP” on page 3156* / *Chapter 1.6.3.7.4.2 “CI522-MODTCP” on page 3196* for diagnosis messages).

In case a module is replaced, the new module will automatically be parameterized with the last parameters of the removed module (if single parameters were written to the previously removed module, this parameters will be ignored).

During pulling or plugging of a certain module, all other module will continue to operate with one limitation: The reaction time of modules connected to the right of the affected module will be bigger in that case (up to 50 ms).

If the bus failure detection is active for CI52x and failsafe is configured (see *Chapter 1.6.5.3.1.3.3 “Cyclic I/O data exchange” on page 3623*) the following behavior applies if a module is removed and replugged during failsafe condition:

- Last value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will remain at its last value.
  - If the module is removed and plugged again, the output will remain off, and not be kept its last value, as the last value of the new module is “0” in that case.
- Substitute value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again.
- Substitute value for x seconds configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value for the configured time.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again, and the configured time starts again.

Mandatory rules for hot swapping

Mandatory rules for hot swapping:

- Between two pull and / or plug operations of I/O modules a pause of at least 1 second must be observed.
  - That means if a module is pulled or plugged there has to be at least a break of 1 second before the next module is pulled or plugged.
- At boot up of CI52x all configured expansion modules have to be physically available.
  - Start up with missing modules is not supported.
- In the application program it is possible to detect if a hot swap terminal unit is mounted in a specific position on the I/O bus. The information is available in the common device information registers. These can be accessed when the version of the communication interface module supports hot swap.
  - This has to be checked by application:

    Best way for checking if a hot swap terminal unit is available or not, is reading out the common device information registers (see *Chapter 1.6.5.3.1.2.2 “Information data section (Acyclic data)” on page 3605*). If the CI52x rejects this read out the CI52x doesn’t support hot swap at all.

1.6.5.3.2 PROFINET communication interface module

Hot swap

With hot swap for AC500 and S500 it is possible to exchange expansion modules (with same type) during run time.

Preconditions for using hot swap

Information about preconditions for using hot swap see *“Hot swap” on page 3523*.

Compatibility of hot swap

	PROFINET remote I/O
I/O module on TU5xx-H connected to I/O bus master	CI501-PNIO or CI502-PNIO
Required version of I/O bus master	Module index as of F0 Firmware as of V3.2.10
Fieldbus master when used as remote I/O with AC500 V3	Not supported
When used as remote I/O on third party controller (PLC or DCS)	Note: alarms must be acknowledged by fieldbus master. GSDML as of version GSDML-V2.3-ABB-S500-CI501-PNIO-20180822.xml or GSDML-V2.3-ABB-S500-CI502-PNIO-20180822.xml needed for full scope of vendor specific diagnosis.

Hot swap behavior

The following table describes the behavior in case of I/O attached to communication interface module for PROFINET, CI501-PNIO or CI502-PNIO.

Hot Swap Behavior	PROFINET remote I/O with third party controller
Start-up behavior with missing or damaged I/O module on hot swap terminal unit TU5xx-H	Remote I/O station is not starting As of device index F1 and Automation Builder Version 2.4.1 it is possible to configure the startup in case of missing modules on hot swap terminal units. If configured, the remote I/O station is starting up with missing or damaged I/O module, if the module is plugged later or replaced it will be automatically parameterized and I/O data will be exchanged.
Start-up behavior with wrong I/O module type on any terminal unit	Remote I/O station is not starting
Diagnosis of presence of hot swap terminal unit	Information is available either: ● via acyclic services or ● as cyclic state information in the process image As of device index F1 and Automation Builder Version 2.4.1 it is possible to configure a list of required hot swap terminal units. If a required hot swap terminal unit is missing (normal one plugged) this will not prevent a normal operation but a diagnosis message will be generated for the corresponding slot.
Diagnosis of hot swap capability of I/O module mounted on hot swap terminal unit	Diagnosis is transmitted as vendor specific PROFINET channel diagnosis: ● Diagnosis in case that a not hot-swap-pable I/O module is plugged on a hot swap terminal unit ● Diagnosis in case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration Production data and version index of the modules is accessible via acyclic services
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	PROFINET channel diagnosis is generated together with standard "pull alarm" which must be acknowledged
Input state in process image of controller while module is pulled or module is not operational	Input = ZERO In addition a standard PROFINET state information is transmitted saying "Inputs not valid"
Diagnosis after plugging of the I/O module on the hot swap terminal unit	PROFINET channel diagnosis is generated together with standard "plug alarm" which must be acknowledged

System behavior

If an expansion module is removed or defective during run time, the input data of this module will be set to “0” and the module state will be set to the corresponding value. A diagnosis message will be created in that case (see hardware description of *Chapter 1.6.3.7.5.2 “CI501-PNIO” on page 3224* / *Chapter 1.6.3.7.5.3 “CI502-PNIO” on page 3263* for diagnosis messages).

In case a module is replaced, the new module will automatically be parameterized with the last parameters of the removed module (if single parameters were written to the previously removed module, this parameters will be ignored).

During pulling or plugging of a certain module, all other module will continue to operate with one limitation: The reaction time of modules connected to the right of the affected module will be bigger in that case (up to 50 ms).

If the bus failure detection is active for CI50x and failsafe is configured the following behavior applies if a module is removed and replugged during failsafe condition:

- Last value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will remain at its last value.
  - If the module is removed and plugged again, the output will remain off, and not be kept its last value, as the last value of the new module is “0” in that case.
- Substitute value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again.
- Substitute value for x seconds configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value for the configured time.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again, and the configured time starts again.

Mandatory rules for hot swapping

Mandatory rules for hot swapping:

- Between two pull and / or plug operations of I/O modules a pause of at least 1 second must be observed.
  - That means if a module is pulled or plugged there has to be at least a break of 1 second before the next module is pulled or plugged.
- At boot up of CI50x all configured expansion modules have to be physically available.
  - Start up with missing modules is not supported.
- In the application program it is possible to detect if a hot swap terminal unit is mounted in a specific position on the I/O bus. The information is available in the process data area or can be read out via acyclic read. These can be accessed when the version of the communication interface module supports hot swap.
  - This has to be checked by application:
    Best way for checking if a hot swap terminal unit is available or not, is checking the corresponding information inside the process image.

1.6.6 Configuration in Automation Builder for AC500 V3 products

1.6.6.1 General settings

This chapter describes the device configuration of AC500 product family with Automation Builder. Basic information on Automation Builder handling can be found in the *Chapter 1.2 “Getting started” on page 11*.

1.6.6.1.1 Project handling

What is a project?

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

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

See also

- ↳ *Chapter 1.4.1.20.2.1 “Object 'Application'” on page 819*
- ↳ *Chapter 1.4.1.20.2 “Objects” on page 818*
- ↳ *Chapter 1.4.1.20.4 “Dialogs” on page 1149*
- ↳ *Chapter 1.4.1.20.3.4.13 “Command 'Project information'” on page 1007*
- ↳ *Chapter 1.4.1.5 “Protecting and Saving Projects” on page 197*

Creating a new project

1. Select *“File ➔ New Project”*.

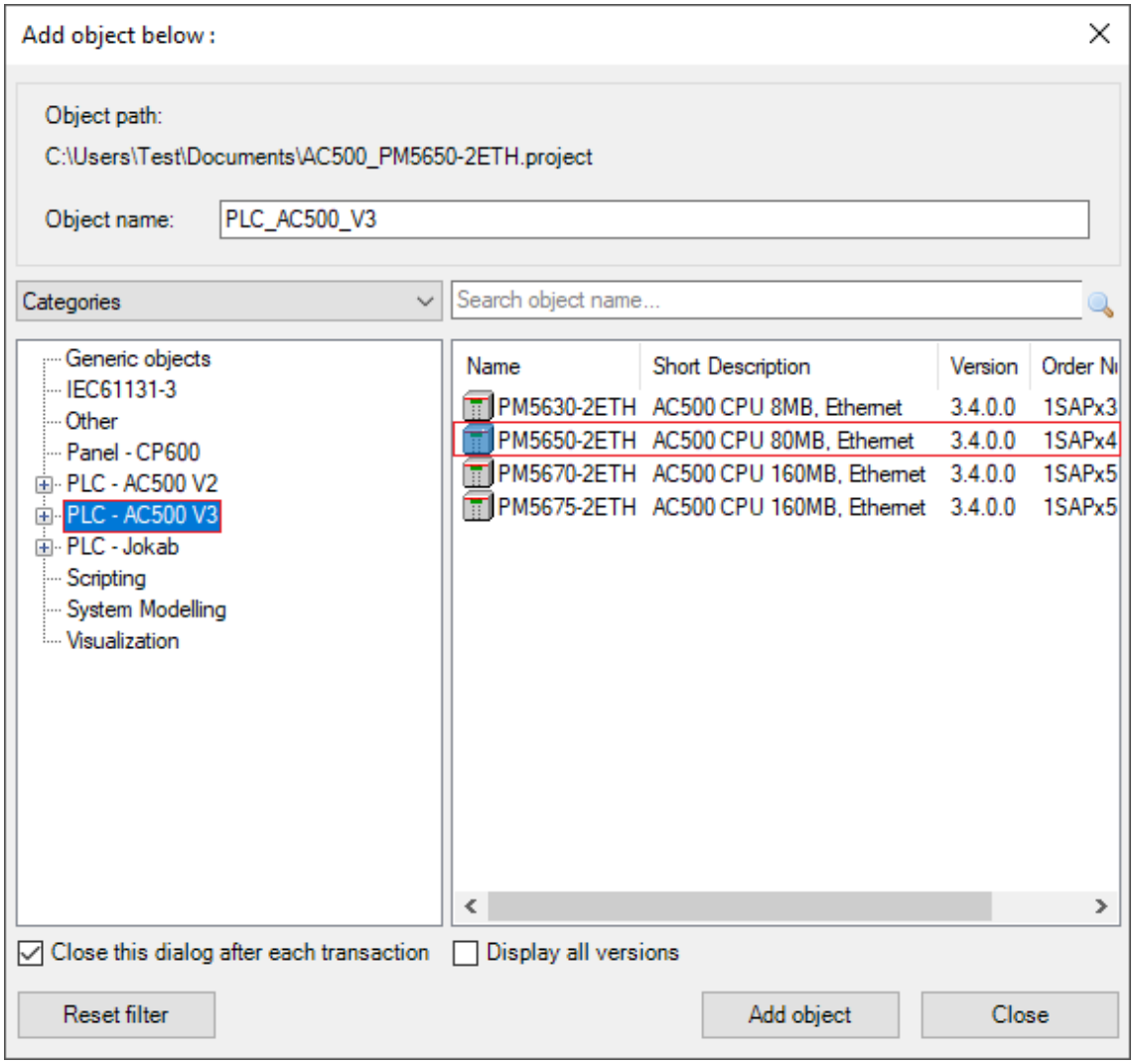
   If the used Automation Builder version is not the latest version, an information is displayed.

   - Select *“Change to newest installed version”* to create a project with the latest installed version of Automation Builder.
   - Select *“Continue to work with version: XXX”* to create a project in the current software version.

2. Select *“AC500 project”*, enter a project name and specify the storage location for the new project.

   With *“Empty project”* a project without a PLC is created.

3. Select the device type for the new project and click *[Add device]*.



⇨ A new project is created and can be configured.

Opening an existing project

NOTICE!

Risk of damaging Automation Builder projects!

Projects created with Automation Builder are incompatible with CODESYS V2.3.9.x. Do not open projects with CODESYS V2.3.9.x as this can cause corrupted Automation Builder projects.

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

Opening a project

1. Select *"File ➔ Open Project"*.
   ⇨ The *"Open Project"* dialog appears.
2. Select a previously saved project from the file system.
   ⇨ Automation Builder switches to the version of the project and opens the project.

Exporting and importing a project

Configuration of a complete PLC or of single devices can be reused within the same project by copy-and-paste the desired nodes in the device tree.

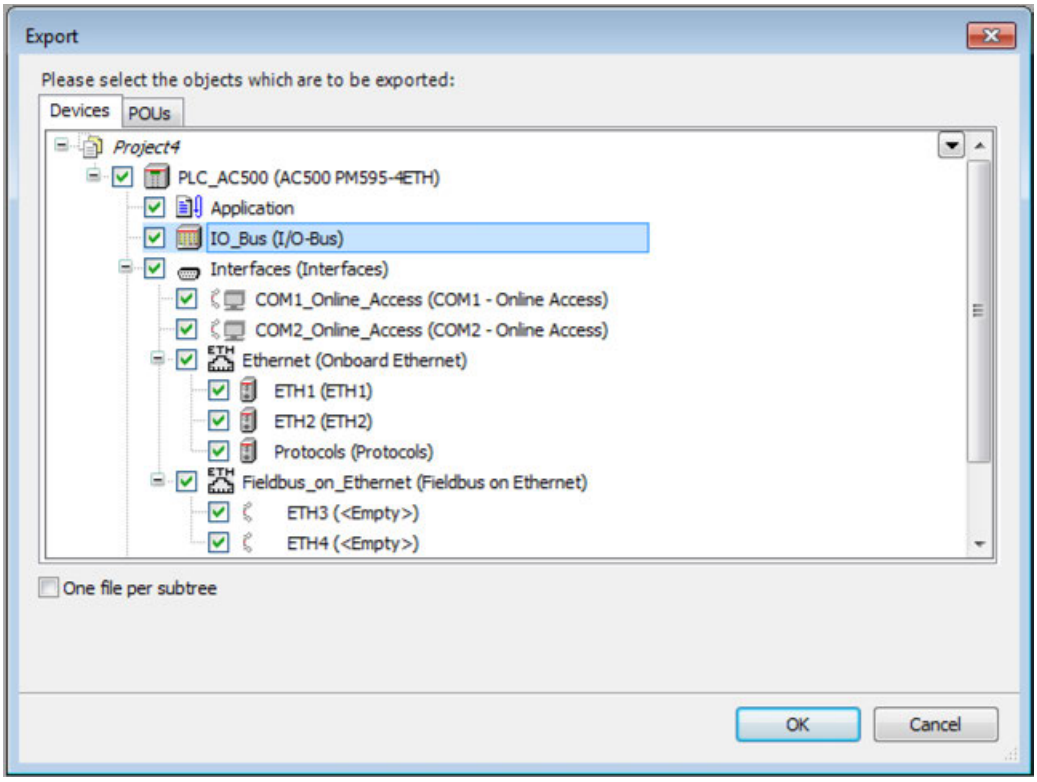
In order to reuse a PLC configuration cross-over projects, the project configuration can be exported and imported afterwards into another project.

> *An exported project configuration can only be imported to a project with the same Automation Builder version. If the versions are not the same, the import fails with an error message.*

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

Project export

From the menu, select *"Project ➔ Export ➔ Project"*. Select the objects to be exported. The configuration of the selected items will be added to an export file (*.export).



"One file per subtree": If this option is activated, all objects belonging to the same subtree will be exported into the same export file, otherwise a separate file will be created for each particular object.

Project import

For importing a project a basic and an advanced function is available.

Basic project import: Users with a basic or a standard Automation Builder license can perform a **basic project import**. Command: *"Project ➔ Import ➔ Project"*.

A previously exported project configuration is imported into the current project. With this, the current project configuration is overwritten.

In order to supplement the current project with the project configuration of a previously exported project, use the compare function. Command: "Project ➔ Compare".

Chapter 1.6.6.1.1.6 "Comparing projects" on page 3640

Advanced project import: Users with a premium Automation Builder license can perform an **advanced project import**. Command: *"Project ➔ Import ➔ Project with compare"*. This command allows to compare two projects, to check on differences and to adapt single parts of the project configuration easily.

Basic project import

1. From the menu, select *"Project ➔ Import ➔ Project"*.

A previously exported project configuration is imported into the current project. With this, the current project configuration is overwritten.

In order to supplement the current project with the project configuration of a previously exported project, use the compare function. Command: "Project ➔ Compare".

Chapter 1.6.6.1.1.6 "Comparing projects" on page 3640

2. Select the export file from the file system and click *[Open]* to import the project configuration.

An exported project configuration can only be imported to a project with the same Automation Builder version. If the versions are not the same, the import fails with an error message.

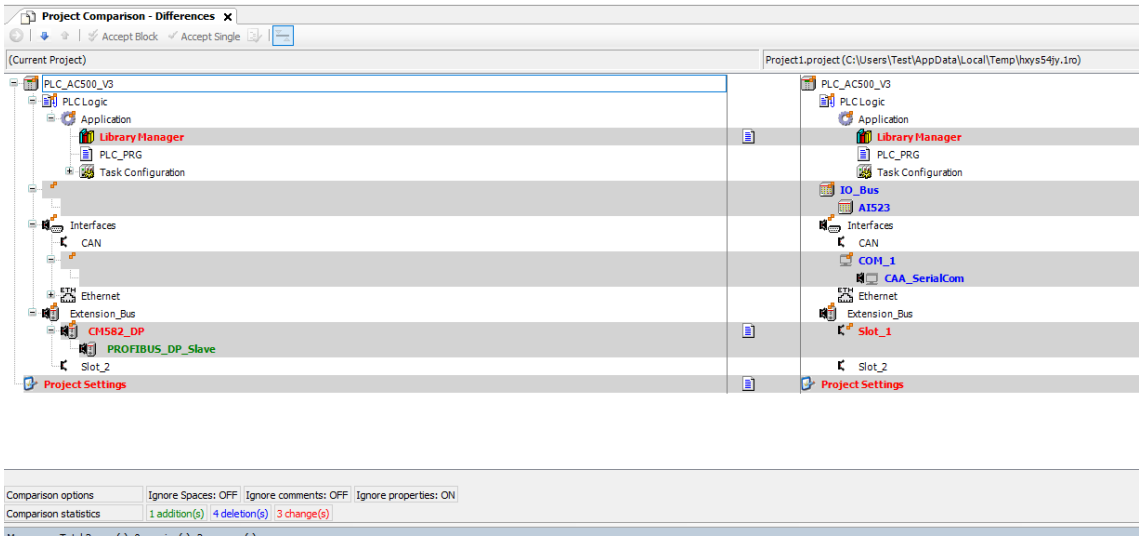
Advanced project import

Perform an advanced project import in order to compare two projects, to check on differences and to adapt single parts of a previously exported project configuration easily.

1. From the menu, select *"Project ➔ Import ➔ Project with compare"*.
2. Select the export file from the file system and click *[Open]* to import the project configuration.

   ⇨ The project import is started.

3. Once the project file is imported, a compare view is displayed. The left pane represents the current project, the right pane represents the imported project.



⇨ Differences between the current project and the imported project are highlighted in red color.

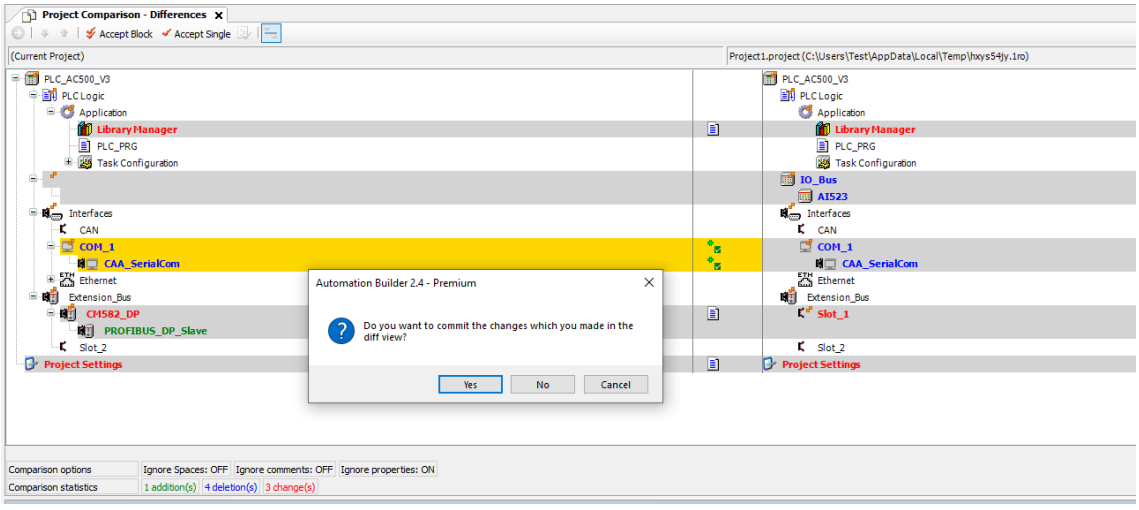
⇨ Additional modules in the current project that are not available in the imported project are highlighted in green color.

⇨ Additional modules in the the imported project or deleted modules in the current project are highlighted in blue color.

⇨ A summary of all differences within the projects is given in the *"Comparison statistics"* under the device tree.

4. Every highlighted item of both projects can be handled individually and can either be transferred to the current project or skipped.

- *[Accept Block]*: All items of the selected node are transferred to the current project with one click. Use this function for example to copy all nodes of a PLC configuration from the imported project to the current project (select *"I/O_Bus"* node).
- *[Accept Single]*: Only a single item from a node is transferred to the current project. Use this function for example to copy certain I/O modules from the imported project to the current project.

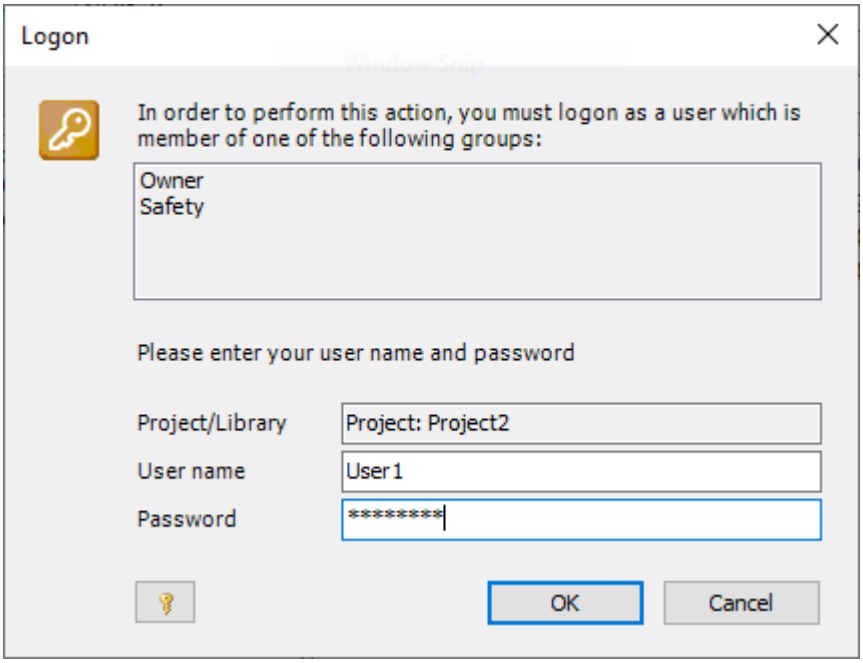


⇨ All accepted items are highlighted in the current project in yellow color.

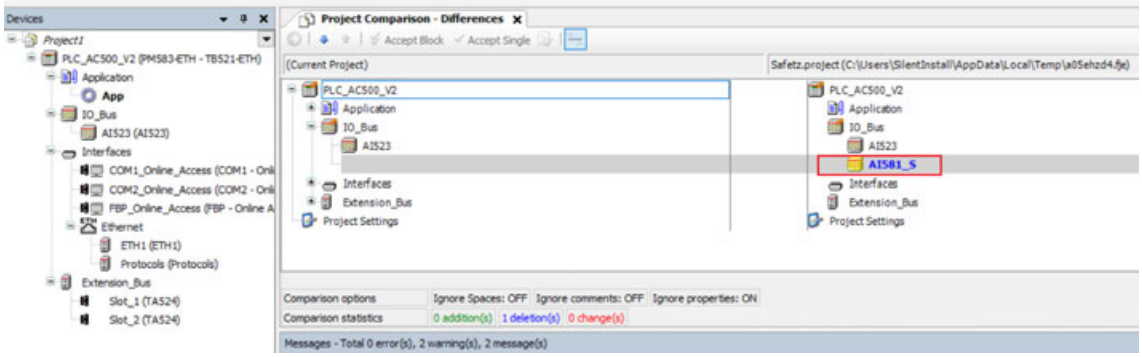
⇨ To undo a selection, again, click *[Accept Block]* or *[Accept Single]*.

⇨ To accept all changes on the current project, close the *"Project Comparison - Differences"* tab and confirm the prompted dialog.

5. If in the import project the PLC contains an AC500-S safety module, a security check is performed which requires user authentication:



6. After a successful user authentication the AC500-S safety modules are added to the compare view and can be imported to the current project.



Upgrading/ updating a project to a new Automation Builder version or profile

When upgrading or updating Automation Builder a previously configured project can be converted in order to be used in a new Automation Builder version or with a new Automation Builder profile.

Definition:

*Automation Builder upgrade: changing over to a major Automation Builder version (e.g. from version 2.**3**.1 to version 2.**4**.1).*

*Automation Builder update: changing over to a minor Automation Builder version (e.g. from version 2.4.**0** to version 2.4.**1**).*

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

↳ *Chapter 1.6.6.6 “Converting an AC500 V2 project to an AC500 V3 project” on page 3993*

Before the upgrade/update

Project archive

*Create a project archive before updating Automation Builder. Project archives contain all project data, including data that is not stored with a *.project file, e.g. device description files for third party devices.*

Chapter 1.6.6.1.1.7.1 “Creation of an archive ” on page 3642

RobotStudio station

RobotStudio integration has been discontinued as of Automation Builder 2.1.0. It is recommended to externally store the link to the RobotStudio station and to remove the RobotStudio station object prior to the upgrade.

Automation Builder profile

To use the Automation Builder profile of an older project, the old profile must have been installed. The installation of older Automation Builder profiles can be activated in the device dialog during the upgrade process.

Upgrading/ Updating a project

1. With opening a project Automation Builder automatically detects the project version. In case of an outdated project version a dialog is prompted.

   ⇨ If the update is confirmed, the project is automatically updated to the latest Automation Builder version.

   Automation Builder updates the complete project (complete device tree) to the latest version. Success messages, warnings and errors are described in the section *“All messages”*.

   ⇨ If the update is declined, the project is closed unchanged.

   In order to initiate a project update or upgrade later on, select *“Project ➔ Update Project”*.

   ⇨

To keep an older project, it must be opened with the same Control Builder Plus/ Automation Builder version the project has been created. For this, the appropriate Control Builder Plus/ Automation Builder profil must be selected.

In this mode, new Automation Builder features cannot be used.

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

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

AC500 V2 libraries

2. When upgrading Automation Builder, new available AC500 V2 system libraries are installed automatically. In difference to AC500 V3 libraries the AC500 V2 libraries are not versioned. Hence, after an Automation Builder update login to a PLC might only be possible after a rebuild and with an online change. This might be required although the application has not been changed and the previous version profile is still in use.

   To avoid this, add the AC500 V2 libraries to the Automation Builder project. The procedure on how to add a AC500 V2 (system) library to a project is described exemplarily.

   ↳ *Chapter 1.4.1.16.2 “Adding a Library to the Application” on page 450*

Migrate third party devices

3. During the project upgrade, an option for migration of third party devices can be selected. If this option was not selected during the upgrade procedure, migration can be initiated manually after an Automation Builder upgrade in order to migrate all third party devices to the project.

   ↳ *Chapter 1.6.6.1.5 “Migration of third party devices” on page 3658*

4. Exception, for the CANopen device CM598-CN:

   > *Usually, when upgrading Automation Builder or an existing project, new AC500 V2 system libraries are installed automatically and older library versions are removed.*
   >
   > *As an exception, for the CANopen device CM598-CN both library versions are available in the Library Manager due to compatibility reasons. However, coexistence of a new library version and an older library version is not possible. In order to avoid compile errors remove the older library version.*

Login

5. After the Automation Builder upgrade login to the PLC from Automation Builder: right-click *“Application ➔ App”* and select *“Login [PLC_AC500_V2]”*.

   ⇨ The firmware on the devices is upgraded.

   > *Depending on the currently installed firmware versions, a login from CODESYS V2.3 might be impossible prior to the firmware update.*

Updating PLC devices

To update all devices of a PLC project, right-click the PLC node and select *“Update objects”*. In the dialog enable *“update subtree”* option to update all sub-objects. Otherwise only the processor module object is updated.

To update a specific device only, the command *“Update objects”* can be executed individually at the specific node.

I/O mapping export and import

Export I/O mapping

To exchange information on I/O mapping only, data can be exported as .csv file. This allows maintenance of I/O data outside Automation Builder, e.g. in MS Excel.

Right-click the *“Processor Module”* node or *“I/O_Bus”* node in the device tree and select *“Export -> IO mapping”*. To export the I/O Mapping for the complete project, e.g. with more than one configured processor modules, I/O data of the complete project can be exported *“Project -> Export -> I/O mapping”*.

Import I/O mapping

A previously exported .csv file can be imported to the project: *“Project -> Import -> I/O mapping”*.

Comparing projects

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

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

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

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

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

Creating a comparison view

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

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

8. Close the comparison view and answer the query whether the changes made are to be saved with *"Yes"*.

   ⇨ The changes become effective in the project.

Opening the detailed compare view

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

1. Double-click the line of the aligned POU versions.

   ⇨ The compare view switches to the detailed compare view of the POU. The modified code lines are highlighted in gray and written in red.

2. Click  .

   ⇨ Code lines with changes (red) are extended by two lines: an line with insert (left, green) and a line with delete (right, blue).

3. Click  again.

   ⇨ The code line is marked again as modified.

4. Move the mouse pointer to the code line marked as modified and click *"Accept Single"*.

   ⇨ The code line from the reference project is activated for acceptance into the current project.

5. Click  .

   ⇨ The project compare view opens for the entire project. It is write-protected (read-only) to prevent you from activating differences for acceptance. The link highlighted in yellow above the tree view also indicates this.

6. Click the link: *"Project compare view is read only because there are uncommitted changes in another view. Click here to switch to the modified view."*

   ⇨ The detailed compare view opens again. The unconfirmed changes are highlighted in yellow.

7. Click ✕ in the tab of the view and confirm that the changes should be saved.

   ⇨ The detail project view is closed and the POU is overwritten. Now it corresponds to the POU of the reference project. The project view is active again so that you can continue working with project compare.

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

See also

- *Chapter 1.4.1.4 "Comparing projects" on page 195*
- *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*
- *Chapter 1.6.6.1.1.6.1 "Creating a comparison view" on page 3640*

Project archive

Automation Builder supports the creation and the import of project archive files. Archive files contain all relevant project data including the PLC configuration, the project files of the CODESYS and all device descriptions. This allows exchanging Automation Builder projects without taking care of the target environment.

Creation of an archive

☑ The following steps describe the creation archive file from an Automation Builder project:

1. Select *“File ➔ Project Archive ➔ Save/Send Archive”*.
2. Select the information which should be included in the archive file from the list box.

Section/Control	Parameter	Description
Information selection list box	Options	Not supported
	Referenced devices	The referenced devices can be selected by expanding the "Referenced devices" item of the list box. It is strongly recommended to include all devices in the project archive to maintain consistency.
Additional files	-	Not supported
Comment	-	Opening a control window which allows the input of a comment to the project archive.
Save	-	Opening a dialog window to determine the path and the file name of the project archive and storing it to the file system.
Send	-	Not supported
Cancel	-	Canceling the operation and closing the dialog window.

With *[Comment]* additional information can be added to the project archive, for example to add a brief description or some information concerning the project.

3. Proceed with *[Save...]*.

It is strongly recommended to keep the default settings.

Section “Options” of the list box is not support. Do not enable this option.

Extraction of an archive

The currently loaded project will be closed automatically when extracting the selected project archive. It is recommended to open a new instance of Automation Builder before starting the extraction process.

☑ The following steps describe the extraction of an archive file and the import to Automation Builder.

1. Select *"File ➔ Project Archive ➔ Extract Archive"*.
2. Select the desired project file and click *[Open]*.

Section/ Control	Parameter	Description
Locations	Extract into the same folder where the archive is located	The project archive will be extracted to the same path where the archive is located.
	Extract into the following folder	Path to which the project archive should be extracted.
	Button ...	Opening a folder selection dialog which allows selecting the desired path.
Contents	Items	Select the items which should be extracted.
	Comment	Displaying comments included inside the Project archive file.
	Extract	Triggering the extraction process. Automation Builder extracts the archive and creates a project from out the archive. After creating the project Automation Builder checks the version of the project. If the project version and the activated Automation Builder version is not identical the workflow is the same as described in "Opening an Existing Project".
	Cancel	Closing the Extract Project Archive dialog and canceling the extraction process.

1.6.6.1.2 User and access rights management

User and access rights

The 'User Management' provide functions for defining user accounts and configure the access rights within a project. The rights to access project objects via specified actions are assigned only to user groups, not to a single user account. So each user must be member of a group.

User management

Before setting up users and user groups, notice the following: The configuration of users and groups is done in the Project Settings dialog *Chapter 1.6.6.1.2.3 "Project Settings - Users and groups" on page 3646*.

- Automatically there is always a group "Everyone" and by default primarily each defined user or other groups are members of this group. Thus each user account at least automatically is provided with defined default settings. Group "Everyone" cannot be deleted, just renamed, and no members can be removed from this group.
- Also automatically there is always a group "Owner" containing one user "Owner". Users can be added to or removed from this group, but at least one user must remain. This group also cannot be deleted and always has all access rights. Thus it is not possible to make a project unusable by denying the respective rights to all groups. Both group and user "owner" might be renamed.
- When starting the programming system resp. a project, primarily no user is logged on the project. But then the user optionally might log on via a defined user account with user name and password in order to have a special set of access rights.

Notice that each project has its own user management!
So, for example to get a special set of access rights for a library included in a project, the user must separately log on to this library. Also users and groups, set up in different projects, are not identical even if they have identical names.

CAUTION!
The user passwords are stored irreversibly!
If a password gets lost, the respective user account gets unusable. If the "Owner"-password gets lost, the entire project might get unusable!

Access right management

User management in a project is only useful in combination with the access right management. Notice the following:

- In a new project basically all rights are not yet defined explicitly but set to a default value. This default value usually is: "granted".
- In the further run of working on the project each right can be explicitly granted or denied resp. set back to default. The access right management of a project is done in the Permissions dialog *"Permissions" on page 3645*.
- Access rights on objects get "inherited". If an object has a "father" object (example: if an action is assigned to a program object, that is inserted in the structure tree below the program, then the program is the "father" of the action object) , the current rights of the father automatically will become the default settings of the child. Father-child relations of objects concerning the access rights usually correspond with the relations shown in the POUs or Devices tree and are indicated in the Permissions dialog by the syntax "<father object>.<child object>".

Example

Action ACT is assigned to POU object PLC_PRG. So in the POUs window ACT is shown in the objects tree indented below PLC_PRG. In the Permissions dialog ACT is represented by "PLC_PRG.ACT" indicating that PLC_PRG is the "father" of ACT. If the "modify" right would be denied explicitly for PLC_PRG and a certain user group, the default value of the "modify" right for ACT automatically also would be "denied".

User management commands

The 'User Management UI' plug-in provides commands for command category 'User Management'.

These are used for:

- Configuration of access rights on the project objects
- Logging on or off to/from the project via a defined user account in order to get the access rights which are associated to this account

The configuration of user accounts and groups is done in the Project Settings subdialog User Management *Chapter 1.6.6.1.2.3 "Project Settings - Users and groups" on page 3646*.

By default the following commands are part of submenu 'User Management' in the 'Project' menu: Logon, Logoff, Permissions.

Logon

Symbol: 

This command opens the Logon dialog for logging on to a project or library via a defined user account.

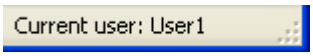
Logging on with a certain user account means to log on with those object access rights which are granted to the group which the user belongs to. The configuration of user accounts and groups is done in the Project Settings subdialog User Management.

To log on select the project or an included library from the selection list in the Project/Library field. Enter User name and Password of a valid user account, noticing that each project or library has an own user and access rights management. Log on with OK.

If already another user is logged on the project, this one will be logged out automatically by the new log-on action.

When you are logged on to a project or library and try to perform an action for which you have no right, automatically a Logon dialog will be opened, giving the possibility to log on with another user account provided with the appropriate rights.

The status bar always displays which user currently is logged on the project.



Logoff

Symbol: 

This command logs off the currently logged on user. If no user had been logged on to the currently opened project or to a referenced library an appropriate message will appear when trying to log off.

If the user currently is logged on to more than one project or referenced library (not necessarily with the same user account) a Logoff dialog will appear when trying to log off.

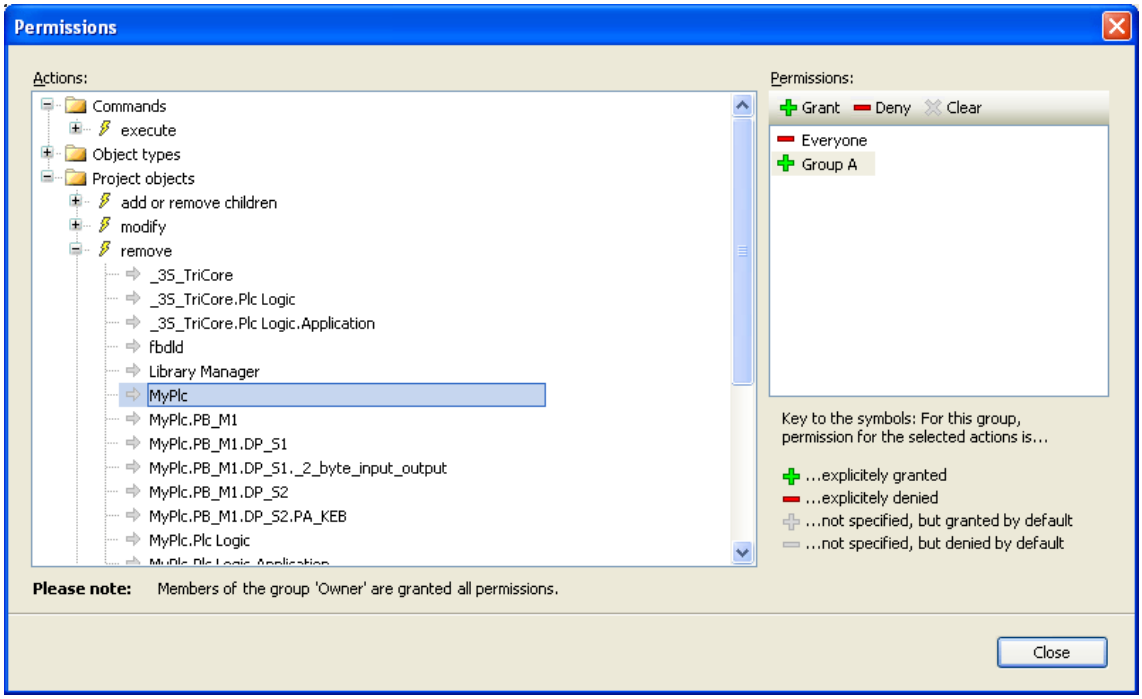
From the Project/Library selection list choose those project/library for which you want to log off. The name of the Current user is displayed just for information.

The status bar always displays which user currently is logged on the project.

Permissions

This command opens the Permissions dialog, where the rights to work on objects or to perform commands in the current project can be configured.

Any changes made in this dialog will be applied immediately.



The Actions window displays all possible rights, that is all actions which might be performed on any object of the current project.

The tree is structured in the following way:

- Top-level see the names of some categories, which have been set up just for the purpose of optical structuring the rights management.
  They are grouping concerning the execution of Commands, the configuration of User accounts and Groups, the creation of Object Types, the viewing, editing, removing and handling of child objects of Project Objects.
- Below each category node there are nodes for the particular actions which might be performed on the command, user account, group, object type or project object. These nodes also only have optical function. Possible Actions:
  - execute (execution of a menu command)
  - create (creating a new object in the current project)
  - add or remove children (adding or removing of "child" objects to an existing object)
  - modify (editing an object in an editor)
  - remove (deleting or cutting an object)
  - view (viewing an object in an editor)

Below each action node find the possible targets, that is project objects, of the respective action.

The Permissions window provides a list of all currently available user groups (except the "Owner" group) and a toolbar for configuring rights to a group.

Select the group and configure it´s permissions.

Left to each group name one of the following icons indicates the currently assigned permission concerning the target which is currently selected in the Actions window:

- : The action(s) for the target(s) currently selected in the Actions window are granted for the selected group.
- : The action(s) for the target(s) currently selected in the Actions window are denied for the selected group.
- : The right to perform the action(s) which are currently selected for the selected target(s) in the Actions window, has not been granted explicitly, but is granted by default, for example because the corresponding right has been granted to the "father" object. (Example: The group has got the right for object "myplc", thus it by default it also has got it for object "myplc.pb_1".) Basically this is the default setting for all rights which not explicitly have been configured.
- : The right to perform the action(s) which are currently selected for the selected target(s) in the Actions window, has not been denied explicitly, but is denied by default, for example in case because the corresponding right has been assigned to the "father" object.

If currently multiple actions are selected in the Actions window, which do not have unique settings referring to the currently selected group, no icon will be displayed.

To configure the rights for a group select the desired action(s) and target in the Actions window and the desired group in the Permissions window. Then use the appropriate button in the toolbar of the Permissions window:

Grant: Explicit granting.

Deny: Explicit denying.

Clear: The currently granted right for the action(s) currently selected in the Actions window will be deleted, that is set back to the default.

Project Settings - Users and groups

The Project Settings dialog in category 'Users and Groups' provides three subdialogs for the user management for the current project: Users, Groups, Settings. For a general description on users and access rights management see help page *Chapter 1.6.6.1.2.1 “User and access rights” on page 3643*.

Users dialog

The currently registered users are listed in a tree structure. The ownerships of each user is displayed and each user is a member of a group by default ↬ *Chapter 1.6.6.1.2.1 "User and access rights" on page 3643*.

Define a new user account

1. Click *"Add"* to open the *"Add User"* dialog.
2. Define the user credentials and click OK to set up the new user. If there are incorrect entries (no login name, password mismatch, user already existing) you will get an appropriate error message.

Modify a user account

Click *"Edit"* to open the *"Edit User"* dialog. The entry fields are the same as in the *"Add User"* dialog. The password fields however - for security reasons - will show 32 * characters. After having modified the desired entries close the dialog with OK to get applied the new settings.

Remove user accounts

Enable the entries to be removed in the Users list and click *"Remove"*. Note that you will get no further inquiry! An error message appears if you try to delete all users from a group. At least one entry must remain.

Groups dialog

Add a group

The currently available groups are displayed in a tree structure. A member also might be a group.

1. Click *"Add"* to open the *"Add Group"* dialog.
2. Define a name for the new group and enable all entries (single users or groups) which should be members of the new group.
3. Click OK to set up the new group. If there are incorrect entries (no name defined, group already existing, in Members having selected a group which would cause a "group cycle", you will get an appropriate error message.

Modify a group

Click *"Edit"* to open the *"Edit User"* dialog. The entry fields are the same as in the 'Add Group' dialog (see above). After having modified the desired entries close the dialog with OK to get applied the new settings.

Remove groups

Enable the entries to be removed in the groups tree and click *"Remove"*. Note that you will get no further inquiry! The members of the deleted groups will remain unmodified. An error message appears if you try to delete the groups "Everyone" and/or "Owner".

Settings dialog

The following basic options and settings concerning the user accounts can be made:

- Maximum number of authentication trials: If activated, the user account will be set invalid after the specified number of trials to log in with a wrong password. If not activated, the number of erroneous trials is unlimited. Default: option activated, number of trials: 3; permissible values: 1-10.
- Automatically log out after time of inactivity: If activated, the user account will be logged out automatically after the specified number of minutes of inactivity (no user actions via mouse or keyboard registered in the programming system). Default: option activated, time: 10 minutes; permissible time values: 1-180 minutes.

1.6.6.1.3 Later change-over of a target system

Changing the processor module type

In a project, you can change the target system by changing the type of processor module or terminal base type. If possible, the device configuration of fieldbusses and interfaces is kept and switched over to the device configuration of the new module.

Target change options:

- between platforms: from V2 platform to V3 platform (and vice versa)
- between module types: from AC500 (standard) to AC500-eCo (and vice versa)
- a combination of changed platform and changed module type

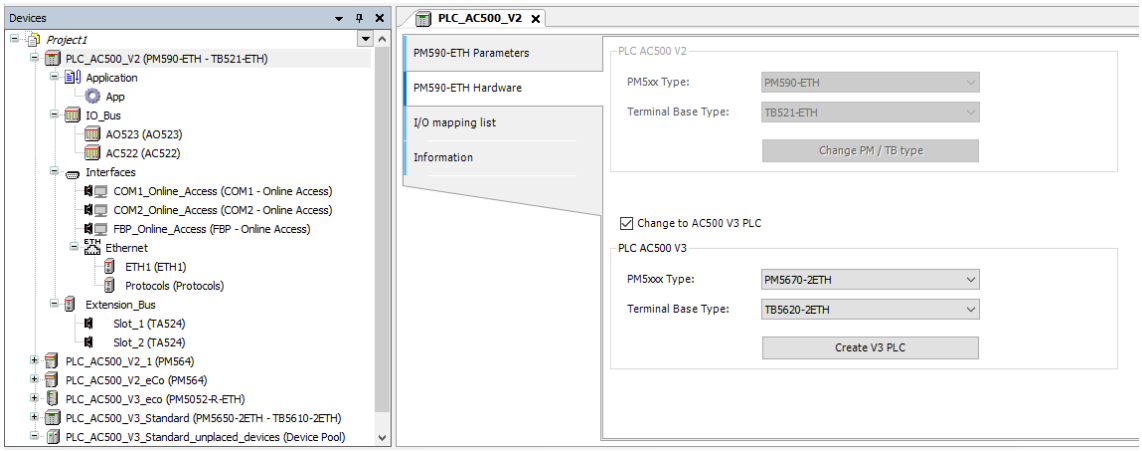
Target change from a V2 processor module to a V3 processor module

Target change options:

- AC500 V2 processor module ➜ AC500 V3 processor module
- AC500 V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500 V3 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node and open the *“PM5<...> Hardware”* tab.
3. Enable *“Change to AC500 V3 PLC”* and select the desired V3 processor module from the *“PM5xx Type”* drop-down list.



4. Click *[Create V3 PLC]*.
   ⇨ The new V3 processor module is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

> *In case of a target change from AC500-eCo V2 to AC500-eCo V3, the I/O bus and Ethernet configuration is kept.*

Target change from a V3 processor module to another V3 processor module

Target change options:

- AC500 V3 processor module ➜ AC500 V3 processor module
- AC500 V3 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V3 processor module ➜ AC500 V3 processor module
- AC500-eCo V3 processor module ➜ AC500-eCo V3 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V3 <...>* node and open the *“PM5<...> Hardware”* tab.
3. Select the desired V3 processor module from the *“PM5xx Type”* drop-down list.

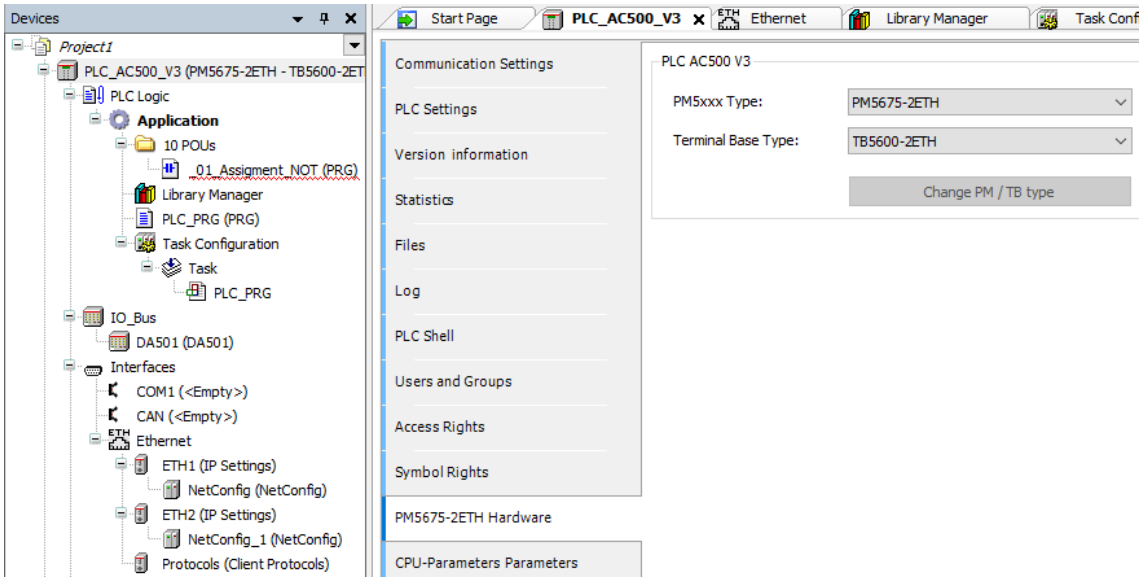
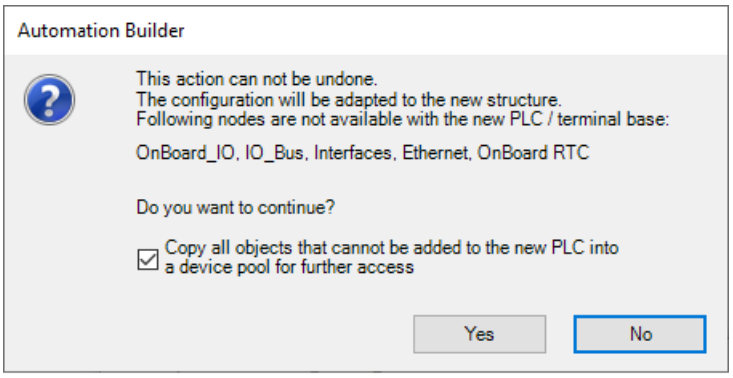


Fig. 309: Change_Hardware_V3

4. Ensure the correct *"Terminal Base Type"* is selected and click *[Change PM / TB type]*.

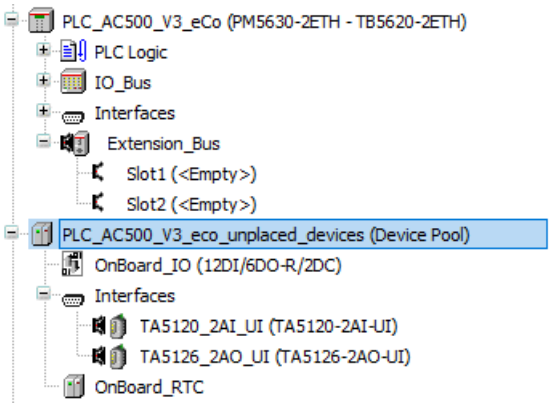
   ⇨ If possible, the device configurations from the previous processor module will be kept and switched over to the new processor module.

   The device configurations that cannot be kept are listed in a prompted information dialog.



   By default, all device configurations which cannot be switched over will be copied to a "device pool" section in the navigation tree (option "Copy all objects that cannot be added to the new PLC into a device pool for further access"). If required, this backuped configuration can be used in another project or in another processor module configuration.

   If the checkbox is deactivated all device configurations that cannot be switched will be lost after the execution of the target change.



Target change from AC500-eCo V3 to AC500 V3

The configuration of the onboard I/Os, the option board slots and the onboard RTC cannot be changed-over to the new module.

Target change from AC500 V3 to AC500-eCo V3

The configuration of COM1, CAN and the I/O bus cannot be changed-over to the new module. Depending on the selected target, also the I/O bus configuration and ETH2 configuration cannot be switched.

ETH1 configuration is kept even if the configured protocols are not allowed for the selected AC500-eCo V3 PLC. In this case error messages are displayed in the messages window.

Libraries which are not used anymore are not deleted with the target change. Libraries of option boards are kept in the Library Manager even if no longer available at the target module.

Customer libraries

CODESYS for AC500 V2 products contains different types of libraries:

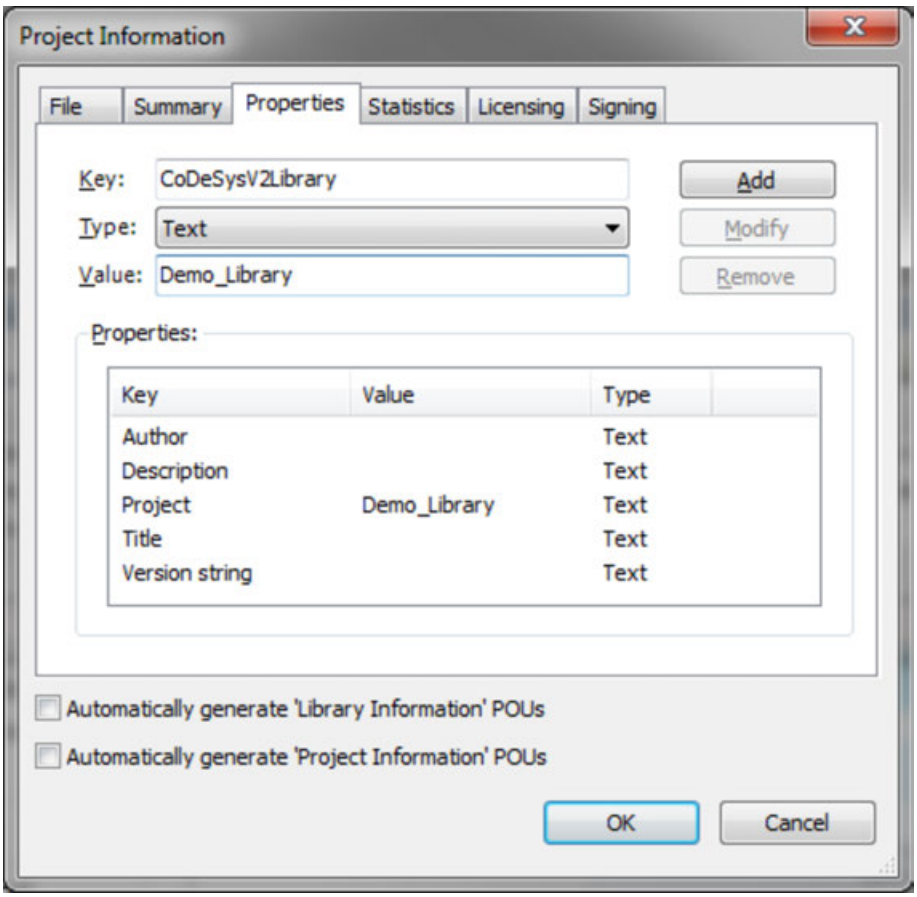
- Standard CODESYS libraries
- Specific AC500 libraries
- Customer libraries

In general, the Standard CODESYS libraries and the AC500 libraries are automatically converted during a target change from AC500 V2 to AC500 V3. Those libraries that cannot be converted (e. g. because there is no matching in V3) are created automatically in the V3 Library Manager and must be manually deleted by the user after the target change.

The customer libraries have to be converted manually using the Library Converter integrated into the Automation Builder installation:

1. In Automation Builder click *“File ➔ Open project”*.
2. Select the CODESYS library for AC500 V2 products which has to be converted.
3. After conversion of the library, open the view POUs in the device navigator and double-click *“Project Information”*.
4. To have the library automatically available in the V3 project, enter “Company”, “Title” and “Version” in the specific fields of the dialog.

   Then, open the *“Properties”* tab. For the target change the new *“Key”* "CoDeSysV2Library" has to be added. Under *“value”*, enter the name of the CODESYS library and click the *“Add”* button.



Click *"File ➔ Save project"* and install into the library repository.

1.6.6.1.4 Firmware identification and update

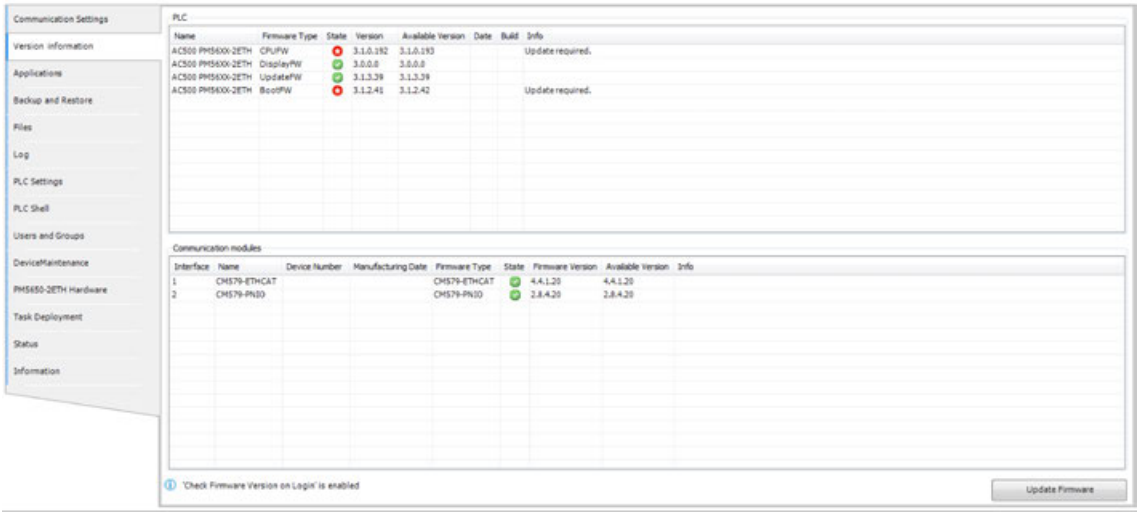
> *Without direct access to the internet, a firmware update with the memory card is also possible.* *Chapter 1.6.7.2 "Memory card in AC500 V3" on page 3999*

Version information

Information on the firmware versions of the processor modules or communication modules, is provided on the *"Version information"* tab.

Remarks:

- The *"Version information"* tab displays the version identified on the device and the version provided with Automation Builder.
- The firmware on the devices must match to the Automation Builder version. Upgrade or downgrade to version supplied with Automation Builder is recommended (especially for CPUs) to ensure correct functionality.
- The firmware type can be changed to the type required by the hardware configuration for devices that support changing the firmware type. E.g., the onboard field bus communication modules of PM595 that may be used as PROFINET, Ethernet or EtherCAT communication module.



State icons

	Firmware version on device matches version supplied with Automation Builder.
	Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.
	Only for communication modules if CPU firmware must be updated first. This happens when CPU firmware has version below 2.5.0.0. Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.

	Identified device is different from configured device, thus no firmware update is possible. Happens only for Communication Modules.
No icon	Firmware of device is not updateable or no newer firmware than the initial version is available.

The [Update Firmware] button to download the new firmware is only enabled if there is updateable firmware.

AC500 V3 firmware installation and update

The PLC firmware can be updated via Automation Builder.

This is also necessary for commissioning V3 CPUs.

A very new CPU has no pre-installed firmware. To guarantee the authenticity of delivered AC500 firmware, V3 CPUs are delivered with a boot loader only. You need to download a valid firmware to the CPU. After download, the functionality of the CPU is given.

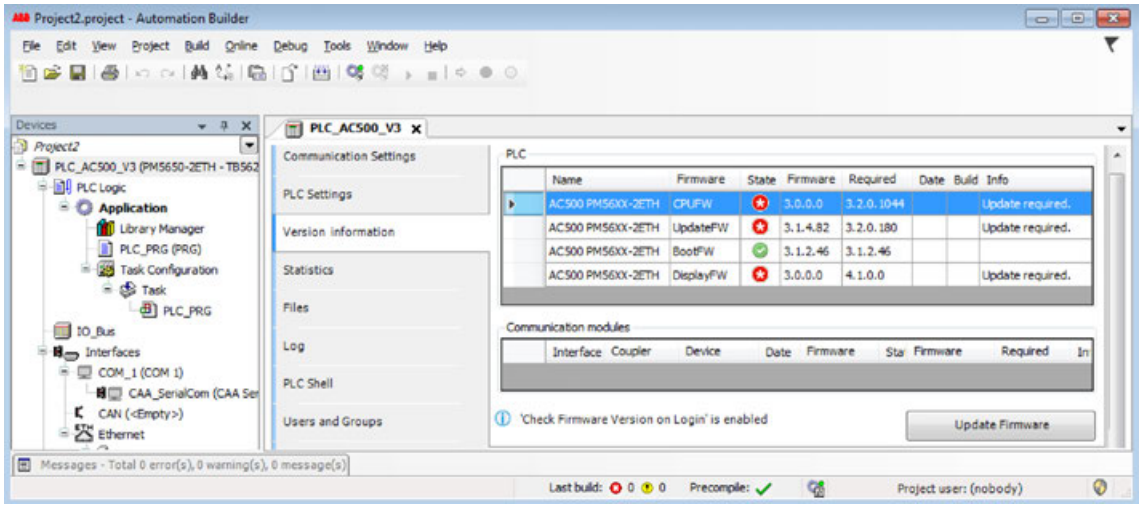
☑ An Automation Builder project with an AC500 V3 CPU is open.

☑ CPU is in "stop" mode or shows **uPdAtE** (update) on the display.

☑ After update the CPU shows either **donE** or **StoP** on the display

☑ For new modules: IP address is set. (The default IP address is 192.168.0.10)

1. Double-click CPU *"PLC_AC500_V3"*.
2. Select *"Version information"*.



3. Select *"Update Firmware"*.

   ⇨ While the update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

4. Wait for the PLC to finish the update.

   A completed update is indicated by a message on the display. Either **donE**, or **StoP**.

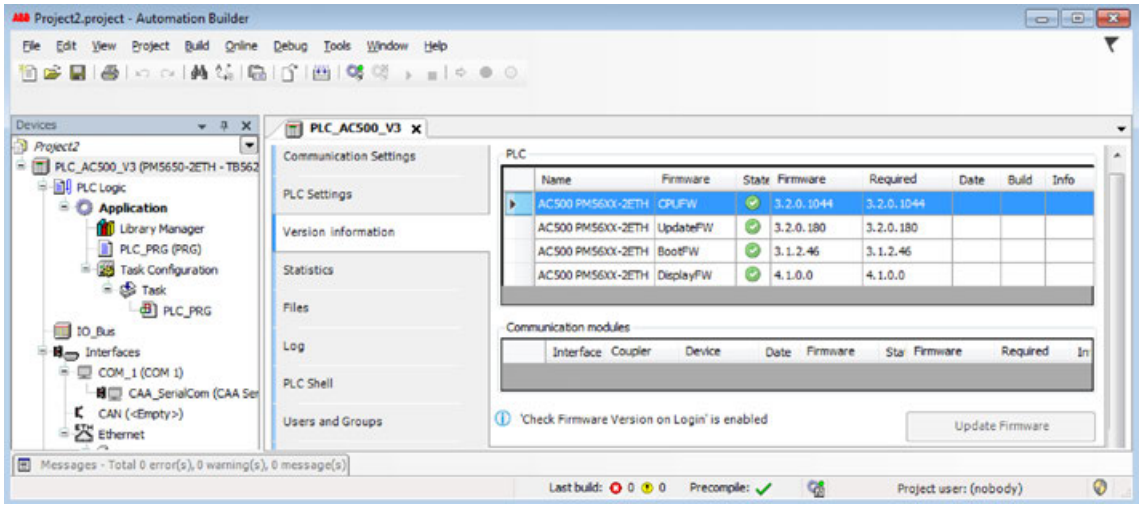
> **NOTICE!**
> Do not disconnect the power supply during the update process! The PLC could be damaged.

   ⇨ **StoP** indicates a restart has been performed by the CPU. When **donE** is displayed sometimes it is necessary to re-boot the CPU manually, e.g., by powering-off. Manual re-boot might be, e.g., for some older CPU versions or if downgrading to an older firmware version according to application settings.

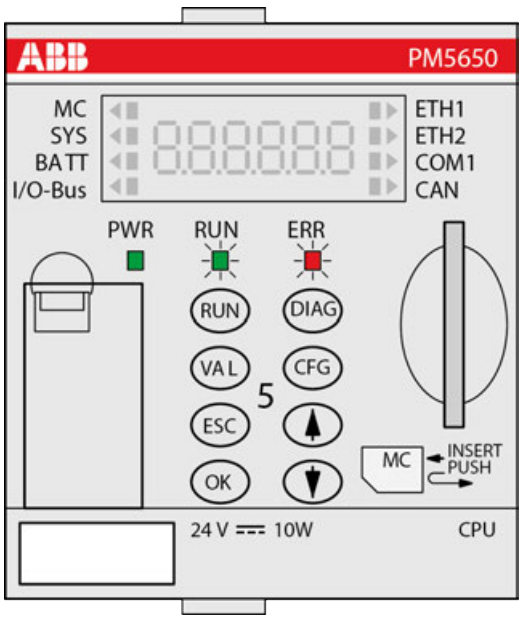
   The CPU display shows "stop" after re-boot. The update process is finished.

5. If necessary, refresh the version information by switching to another tab and back.

   ⇨ Successful firmware update:



Behavior of LEDs during firmware update



LED	LED flashes	Status
RUN and ERR	Toggling	Update pending
RUN	Flashing slow	Done successful
ERR	Flashing slow	Done failed

AC500-eCo V3 firmware installation and update

The PLC firmware can be updated via Automation Builder.

> *This is also necessary for commissioning AC500-eCo V3 CPUs.*

A very new CPU has no pre-installed firmware. To guarantee the authenticity of delivered AC500-eCo firmware, V3 CPUs are delivered with a boot loader only. You need to download a valid firmware to the CPU. After download, the functionality of the CPU is given.

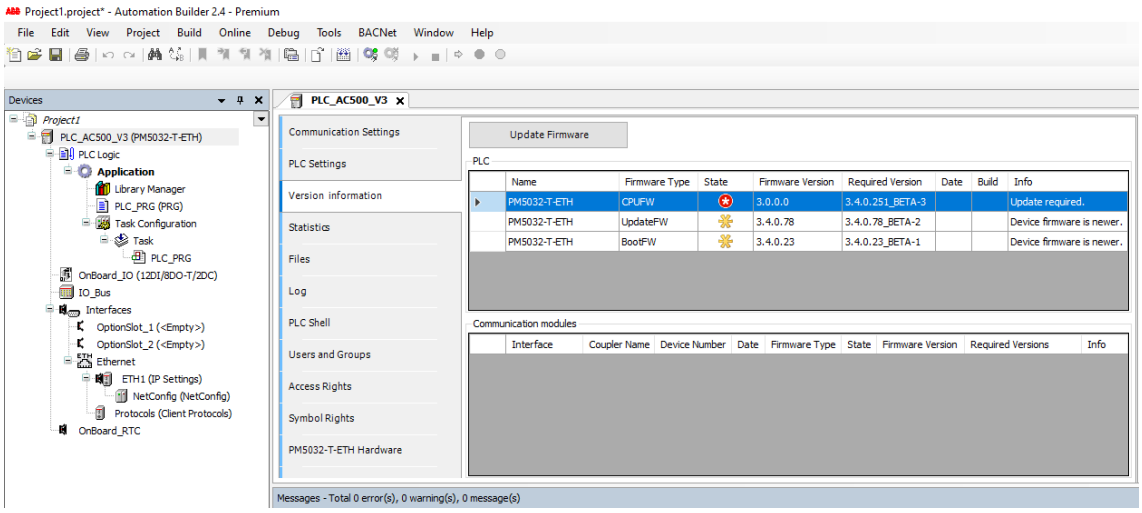
☑ An Automation Builder project with an AC500-eCo V3 CPU is open.

☑ CPU is in "stop" mode without firmware.

☑ The power LED is ON.

☑ For new modules: IP address is set. (The default IP address is 192.168.0.10)

1. Double-click CPU *"PLC_AC500_V3"*.
2. Select *"Version information"*.



3. Select *[Update Firmware]*.

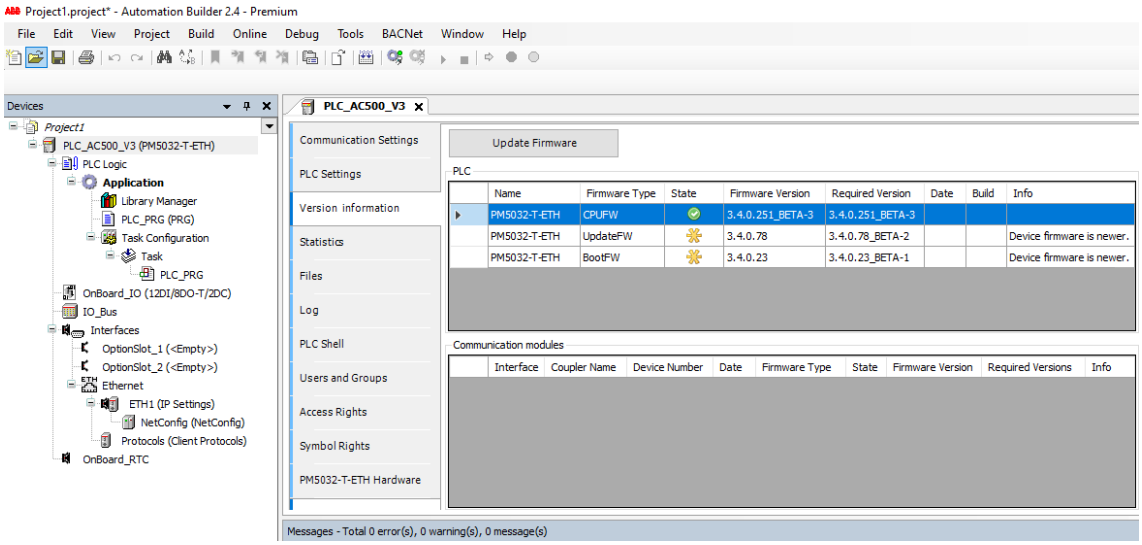
   ⇨ While the update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

4. Wait for the PLC to finish the update.

> **NOTICE!**
> Do not disconnect the power supply during the update process! The PLC could be damaged.

5. If necessary, refresh the version information by switching to another tab and back.

   ⇨ Successful firmware update:



Behavior of LEDs during firmware update

- CPU without firmware, only the power LED is on.
- While the firmware update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

LED	LED flashes	Status
RUN and ERR	Toggling	Update pending
RUN	Flashing slow	Done successful
ERR	Flashing slow	Done failed

- CPU with installed firmware, only the power LED is on.
- If the CPU is running, then the RUN LED is on.
- If the CPU is in STOP mode, the RUN LED is off.

Update CI52x-Modbus firmware

Requirement: A firmware update file is available, e.g. AC500_CI52x_Firmware_V3.2.8.bin.

> *The CI52x Modbus firmware update is only available in the Automation Builder IP Configuration Tool.*

Installation of the IP configuration tool

1. In Automation Builder click *"Tools ➔ Installation Manager"* to start the Installation Manager.
2. Close any other running instances of Automation Builder. Then, click *"Modify"* in the Installation Manager.
3. Select the option *"IP Configuration Tool"* from the list and start the installation of the IP Configuration Tool.

Firmware update procedure

1. In the IP Configuration Tool click *“Scan”* to initialize a device scan.
2. From the list select the CI52x-MODTCP device(s) which shall be updated and click *“FW Update”*.
3. Select the firmware update file (e.g. AC500_CI52x_Firmware_V3.2.8.bin) to initialize a signature check and start the update procedure.
4. After the update, click *“Scan”* again to retrieve the firmware version of the device.

Troubleshooting

After the IP Configuration Tool has been installed, the firmware update of the CI devices can be initialized. If the CI firmware update fails, check the troubleshooting hints and follow the instructions.

General hints

- Close all unused applications on the update PC and do not open Automation Builder or any other applications during the firmware update.
- Stop the communication between AC500 PLC and the CI52x devices and disconnect the Ethernet connection of the update PC and the CI Modbus device(s).
- Do not close the IP Configuration Tool during a firmware update and do not switch off a CI Modbus device during the firmware update.

During a firmware update the operation of the device(s) is stopped. After the update, all outputs are set to zero.

Erroneous firmware update

Error	Solution
Error 1: Package Timeout Due to a primitive firmware update protocol a fast and stable network connection is required. Otherwise the update packages cannot be transferred within the requested time and a timeout occurs.	Locate the PC on which the update is performed as near as possible to the stationed CI Modbus devices. Avoid network switches.
Error 2: Unable to read device status After the firmware update the IP Configuration Tool reads out the status of the updated device in order to check if the update was successful.	Rescan and repeat the update. If this doesn't work, power cycle the device and retry the update.
Error 3: IP is not unique If more than one device hold the same IP address, a firmware update is not possible as the update command is IP based.	Correct the IP address, rescan and repeat the update. If this doesn't work, power cycle the device and retry the update.
Error 4: Internal Error An internal error on the CI52x Modbus device occurred during the firmware update.	Rescan and repeat the update. If this doesn't work, power cycle the device and retry the update.
Error 5: Cannot connect to device The TCP communication is not sufficient for a connection. Increase the connection quality.	See Error 1: Package Timeout.

Signature check failed

After the selection of the firmware file (*.bin) a signature check is performed. If either the firmware file or the signature file is corrupt, the signature check fails. In the event of an erroneous signature check, perform the following steps:

- Ensure the signature file is stored in the same directory as the firmware file.
- Check the file names. The name of the signature file must be the same as the firmware file + attached ".sig".

File names	Name of the firmware file: c:\AC500\AC500_CI52x_Firmware_V3.2.8.bin
	Correct name of the signature file: c:\AC500\AC500_CI52x_Firmware_V3.2.8.bin.sig
	Wrong name of the signature file: c:\AC500\AC500_CI52x_Firmware_V3.2.8.sig

Indeterminate device firmware version

If the firmware version of the device cannot be determined, an error occurs. In this case, check that the device and the update PC are located in the same subnet and ping the device. If the ping is successful you can use the IP Configuration Tool to retrieve the device firmware version.

PC	Device	Result
192.168.14.71 / 255.255.255.0	192.168.14.10 / 255.255.255.0	OK
192.168.**10**.71 / 255.255.**255**.0	192.168.**14**.10 / 255.255.**255**.0	ERROR
192.168.10.71 / 255.255.0.0	192.168.14.10 / 255.255.0.0	OK

1.6.6.1.5 Migration of third party devices

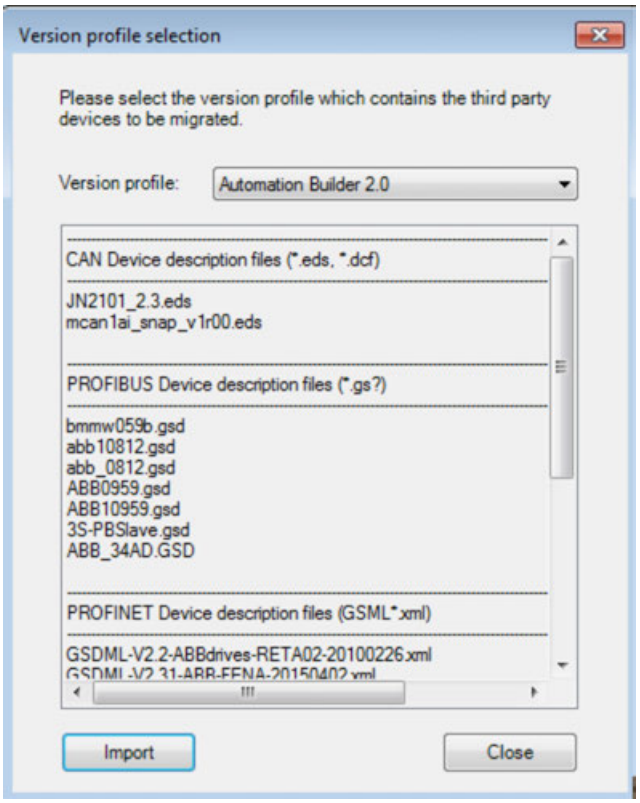
After an update of Automation Builder the device repository contains only ABB devices. The third party devices which were installed into previous versions of Automation Builder are not automatically installed in the newest version profile. This has to be triggered by the user.

The feature “Migrate third party devices” is available as of Automation Builder 2.1.1.

1. Click *“Tools”* in the main menu of Automation Builder.
2. Click *“Migrate third party devices”* in the drop-down list.
   ⇨ The window *Version profile selection* appears.

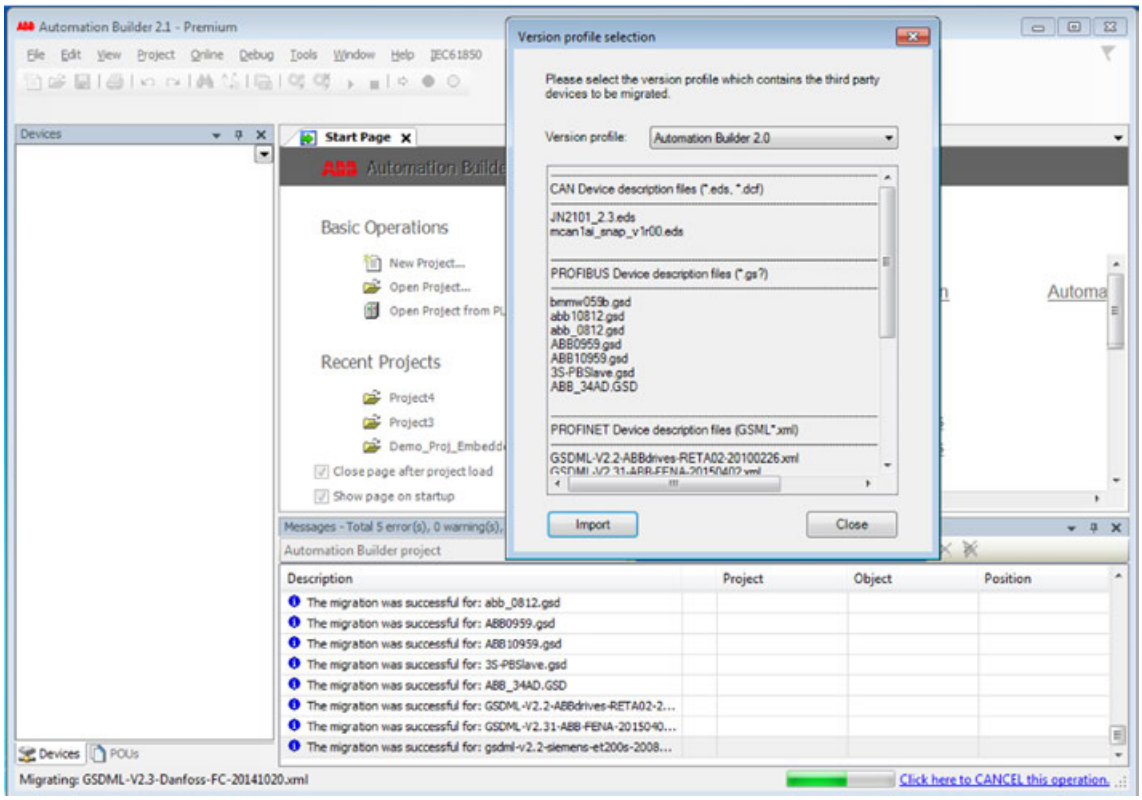
3. Select a version profile in the drop-down list containing previous Automation Builder / Control Builder Plus profiles. The active profile does not appear in the list.

   ⇨ After selection of a previous version profile, all the third party devices which have been installed inside this version profile are listed.



   It is not possible to select or deselect some third party devices. Importing will affect all the third party devices which are listed in the list view.

4. Select *[Import]*.



⇨ During the migration the message window displays success or failure of device migration.

> *In case of failure during the migration the affected third party device description has to be installed manually via main menu "Tools ➔ Device Repository ➔ Install".*

In the status bar, the third party device which is on *Migrating:* <...> is displayed on the left side.

The import operation can be cancelled by clicking the *"Click here to CANCEL this operation"* link on the right side of the status bar. This becomes effective when the migration of the just migrating third party device is finished.

5. To close the dialog select the *[Close]* button of the *Version profile selection*.

1.6.6.1.6 Advanced IO device handling

Automation Builder provides the Advanced IO Device Handling feature for configuring identical IO device types at multiple instances.

This feature is supported by the following commands that works with IO devices only.

- Generate DUT
- Map to Existing DUT
- Release DUT mapping

These commands work on individual nodes and on CI (communication interface) level nodes.

Generating DUT

Each device generates two DUTs. One for the input and one for the output. Some devices contain only input or output type. In such cases, the device generates only one DUT of the relevant type.

- Right-click on the desired IO device and select *"Generate DUT"* to generate a DUT for an IO device.

The following example shows how to generate DUTs at CI level node.

- In the device tree, right-click on a master node such as PNIO_Controller and select *"Generate DUT"* to create DUTs for the child nodes.
- The DUTs of child nodes are generated in *"Application* ➔ *App* ➔ *IO_Device_Generated_Items"* folder.
- Generated DUT considers channels with BYTE datatype as members. If channels with BYTE datatype are not present in the given hierarchy, it adds the members with another higher datatype.
- Channels with BOOL datatype are not considered.

Mapping to existing DUT

This command is enabled for the IO device when the IO device is not mapped and when DUTs of matching size (calculated based on device channel list) are available in *"Application* ➔ *App* ➔ *IO_Device_Generated_Items"* folder.

1. Right-click on an IO device and select *"Map to Existing DUT"*.
   ⇨ Enter Instance Name dialog is displayed.
2. Enter the instance name which satisfies IEC naming validations and unique name in global scope.
3. Click *"OK"* to create a global variable associated with the mappings in DI (PRG).
   If you want to view mapped instances, double-click *"DI (PRG)"*.

With the 'Map to Existing DUT' command:

- Any device can be mapped only to one input DUT and one output DUT. If you have already mapped an input DUT, only the output DUT is shown in the options list and vice-versa.
- Mapping is also supported at CI level nodes. To create global variables for CI level nodes, the address of the first child is considered.

Releasing DUT mapping

This command is enabled on an IO device only when an IO device is mapped either to input, output or both DUTs. You can use this command to release (or revert) mappings and to delete global variables created during 'Map to Existing DUT'.

Right-click on an IO device and select *"Release DUT Mapping"*. The mapped DUT instance is deleted.

Using DUT variables in CODESYS application

1. In the Automation Builder project, double-click *"Application"* to launch CODESYS application.
   ⇨ CODESYS application is launched. CODESYS application contains mapped DUT instances.
2. Double-click *"PLC_PRG"* to create DUT variables.
3. Add DUT variables based on mapped DUTs.

For further information on mapping DUTs, see section *Chapter 1.6.6.1.6.2 "Mapping to existing DUT" on page 3661*.

For example, in the PLC_PRG, add analog I/O and digital I/O. If you insert a dot at a position where an identifier should be inserted, then a selection list is open, offering all the input and output variables which are found in the project.

After adding DUT variables, rebuild the program in CODESYS application using *"Project ➔ Rebuild"*.

Support for CI level node

The user can create DUTs for the entire hierarchy of CI level node (for example, IO_BUS), by right-clicking on the desired CI level node and by selecting *"Generate DUT"*. Further, all the DUTs are generated in *"Application ➔ App ➔ IO_Device_Generated_Items"* folder.

- The command generates DUT for the node itself and also for all child nodes.
- The DUT generated for the CI level node contains generated DUTs for the child nodes as their members.
- For every execution, the command checks, if any new child node is added and generates DUT.

If you delete child nodes in CI level node (for example, IO_BUS), the DUTs generated for these child nodes are not deleted automatically. You should delete the DUTs manually in the *"Application ➔ App ➔ IO_Device_Generated_Items"* folder if desired.

Configuration check

Configuration check for size is enabled to ensure that all devices are mapped with DUTs of the correct size. In case of any changes in the mapped DUT, configuration check verifies the size of the DUT. If it fails, an error message is displayed in Automation Builder messages window and does not allow to launch the application. This check can be performed in *"Create configuration data"*.

1.6.6.2 PLC devices and components

1.6.6.2.1 Device repository

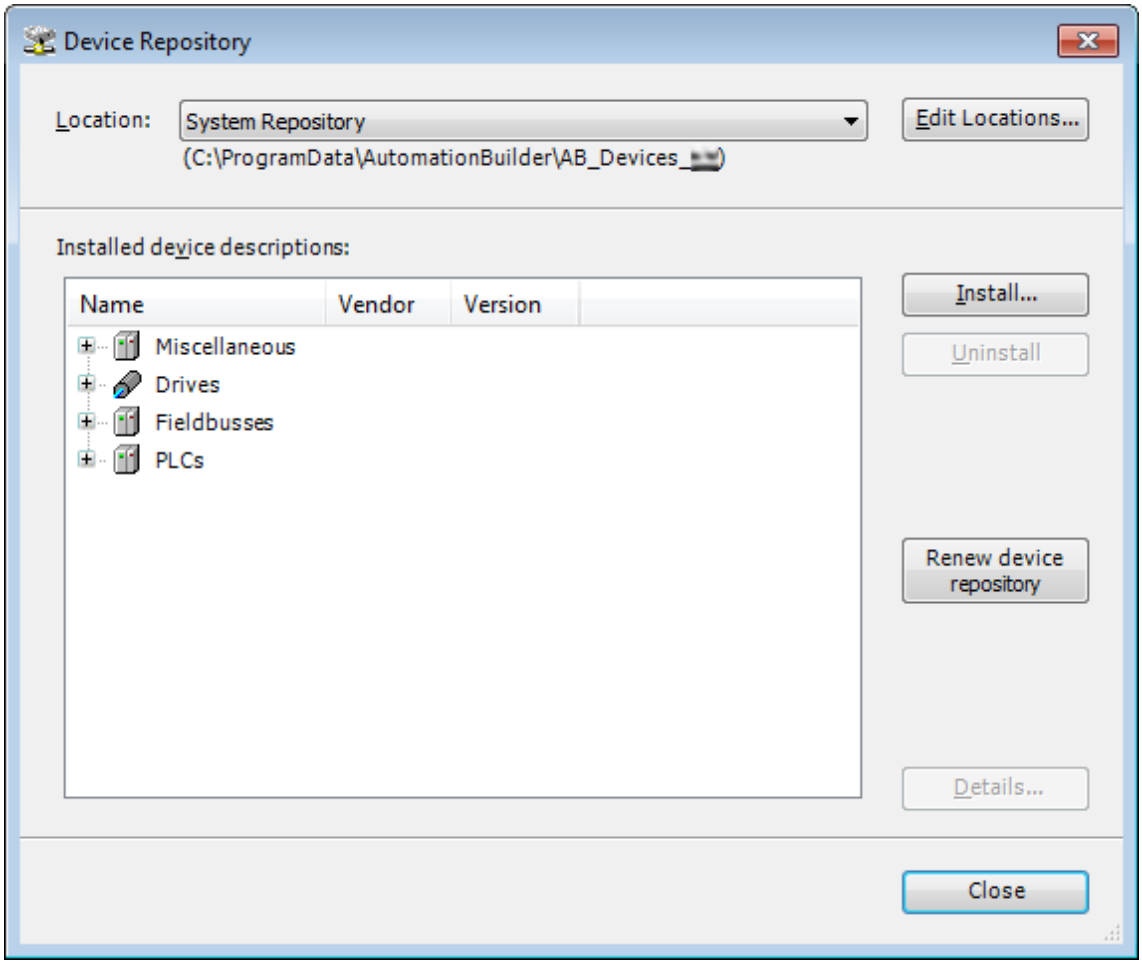
The Device Repository of Automation Builder manages the pool of devices that can be used in the PLC configuration.

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

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

Dialog device repository

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



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

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

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

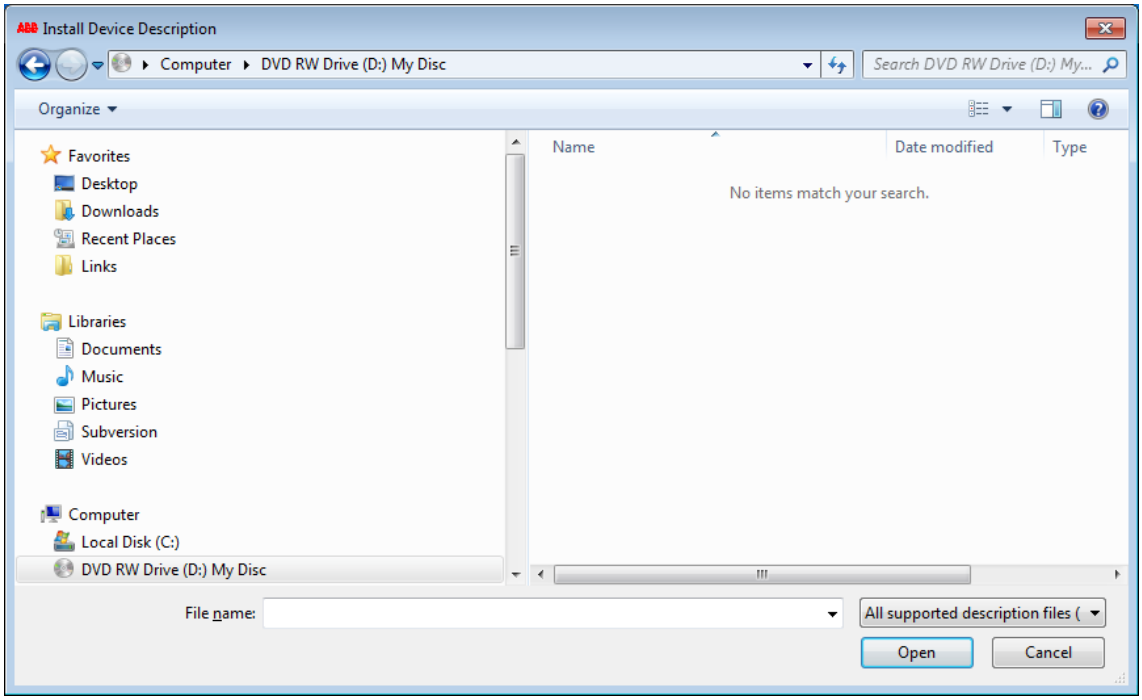
[Details]: Provides technical details on the selected device.

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

Installing devices

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

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



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

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

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

Uninstalling devices

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

The device is removed from the list.

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

1.6.6.2.2 PLC start-up

A fast online program modification of the user program is possible without interrupting the running operation. If data areas should be saved during power OFF/ON, they can be stored in the flash EPROM. An optional battery saves data in the RAM.

Initialization of AC500 V3 CPU

To initialize an AC500 V3 CPU, you need to download the firmware.

A new CPU has no pre-installed firmware, it is delivered with a boot loader only. You need to download a valid firmware to the CPU. See *Chapter 1.6.6.1.4.2 " AC500 V3 firmware installation and update" on page 3653*. After download, the functionality of the CPU is given.

PLC runtime licensing

The use of some libraries and devices require the PLC to have a runtime license. If you purchased such a license, activate the license *Chapter 1.6.6.2.2.2.1 "Activating a runtime license via license key" on page 3665*.

If you want to test device functionality or library features in advance, you can activate a demo license in advance *Chapter 1.6.6.2.2.2.2 "Activating a demo license" on page 3669*.

The license status of a PLC can be displayed at any time *Chapter 1.6.6.2.2.2.5 "View license information" on page 3672*.

NOTICE!

After removing a Wibu memory card (which holds the AC500 runtime license), the PLC system moves into 'Stop' mode after 24 h.

Ensure to insert the Wibu memory card at the time.

Activating a runtime license via license key

The use of some libraries and devices require the PLC to have a runtime license.

☑ PC and PLC are connected. In case of no connection, perform the activation via memory card *Chapter 1.6.6.2.2.2.3 "Licensing via memory card" on page 3669*.

☑ There is a connection to the Internet. In case of no connection, perform the activation on another PC with internet connection *Chapter 1.6.6.2.2.2.1.1 "Activation without internet connection" on page 3666*.

1. Right-click on the PLC and select *"PLC runtime licensing"* from the *"Runtime Licensing"* menu.

   ⇨ A wizard starts. Follow the instructions.

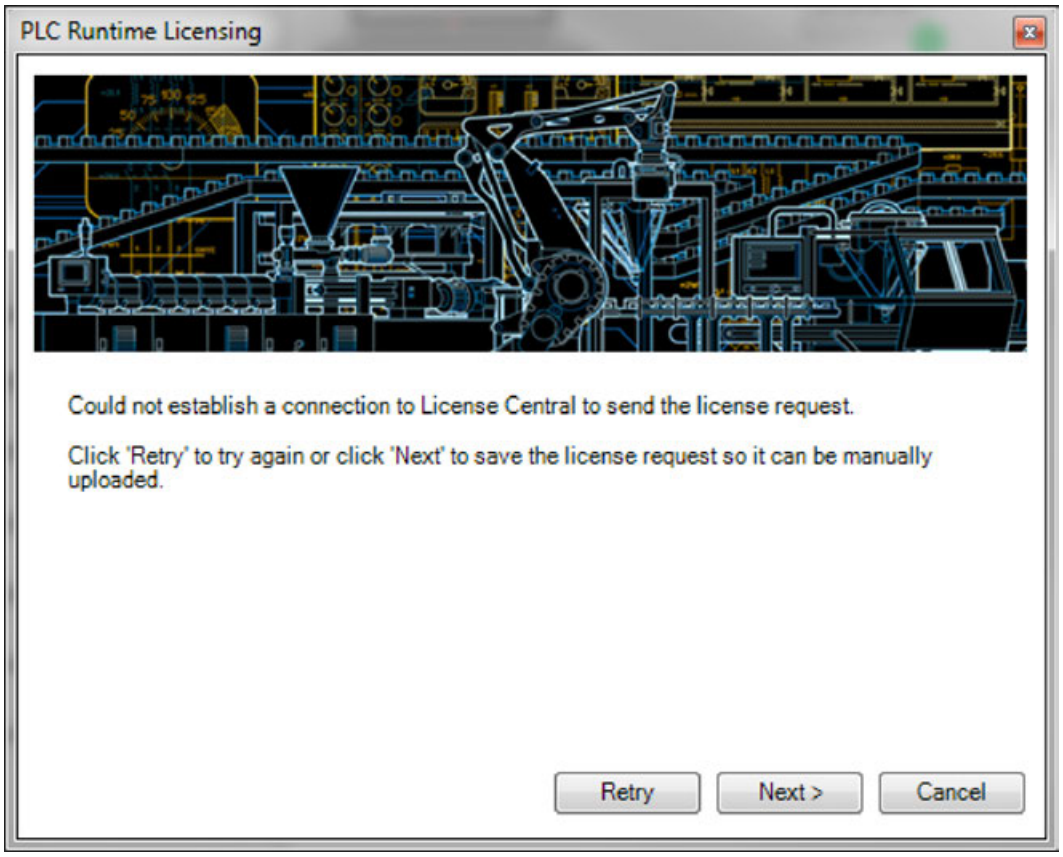
2. Enter the license activation key and select *"Next"* to finish the licensing procedure.

   ⇨ The license is activated on the PLC device.

   If the license shall be used on another PLC device, the installed license can be returned *Chapter 1.6.6.2.2.2.4 "Returning a license" on page 3671*.

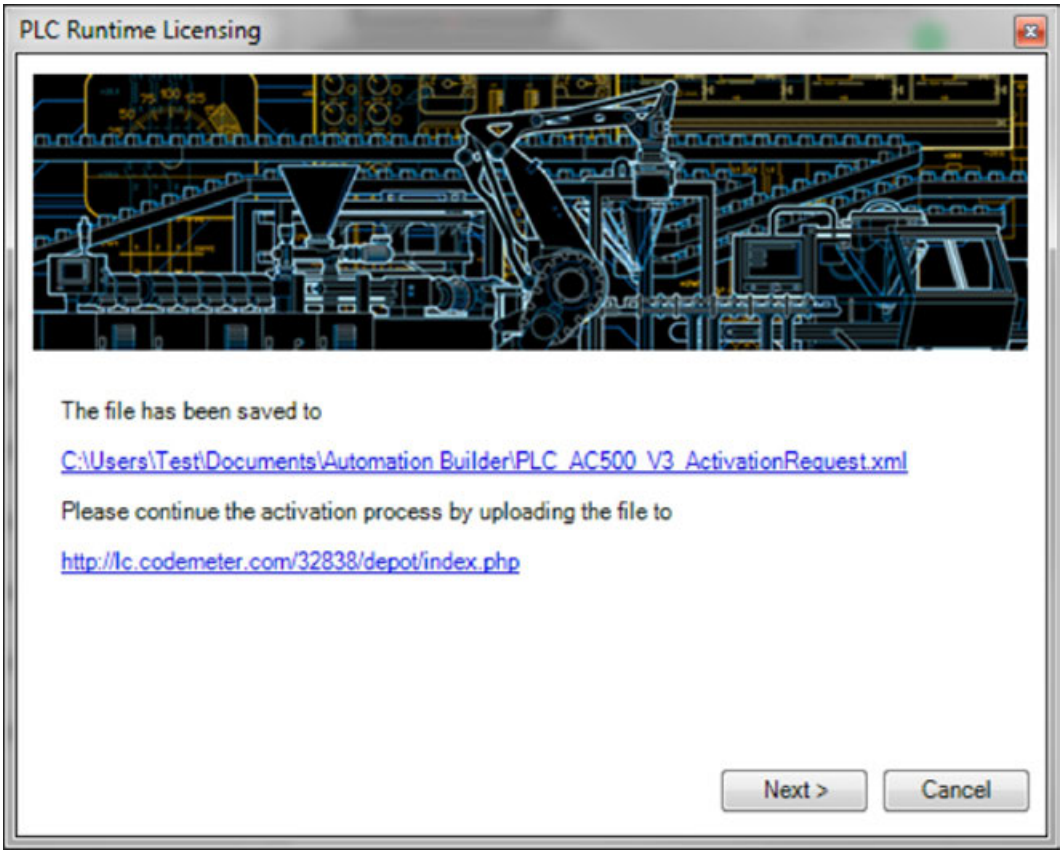
Activation without internet connection

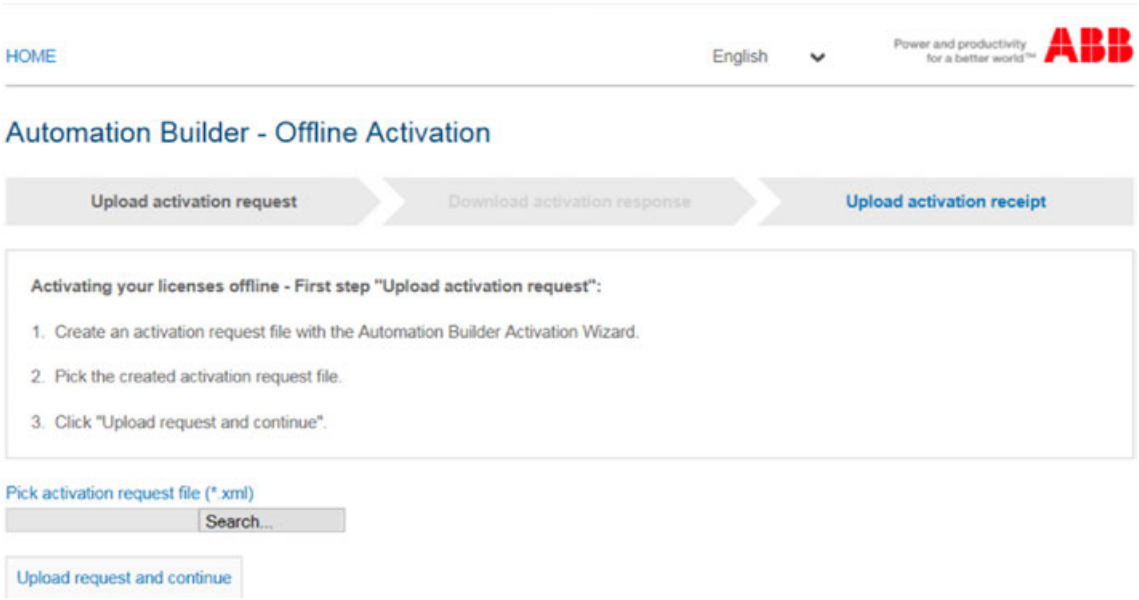
1. If an error occurs when communicating with the ABB license server, or if Automation Builder is running on a PC without internet connection, then it is possible to manually complete the ABB license server interaction by using another PC (with internet connection).



2. In the error dialog select *“Next”* and save the license activation request file to a storage location the other PC can access, e.g. a file share.

3. In the dialog the web address of the ABB license server is displayed (*http://lc.codemeter.com/32838/depot/index.php*). From the PC with internet connection, upload the license activation request file.





4. After the upload, download and save the license activation file from the ABB license server. Transfer this file to the PC without Internet connection.
5. Select *"Next"* to continue the license activation process. Click *"Cancel"* to continue the license activation process at a later time (see Offline activation ↳ *Further information on page 3668*).

6. Select *"Browse"* and select the license activation file (*.WibuCmRaU) from the defined storage location.

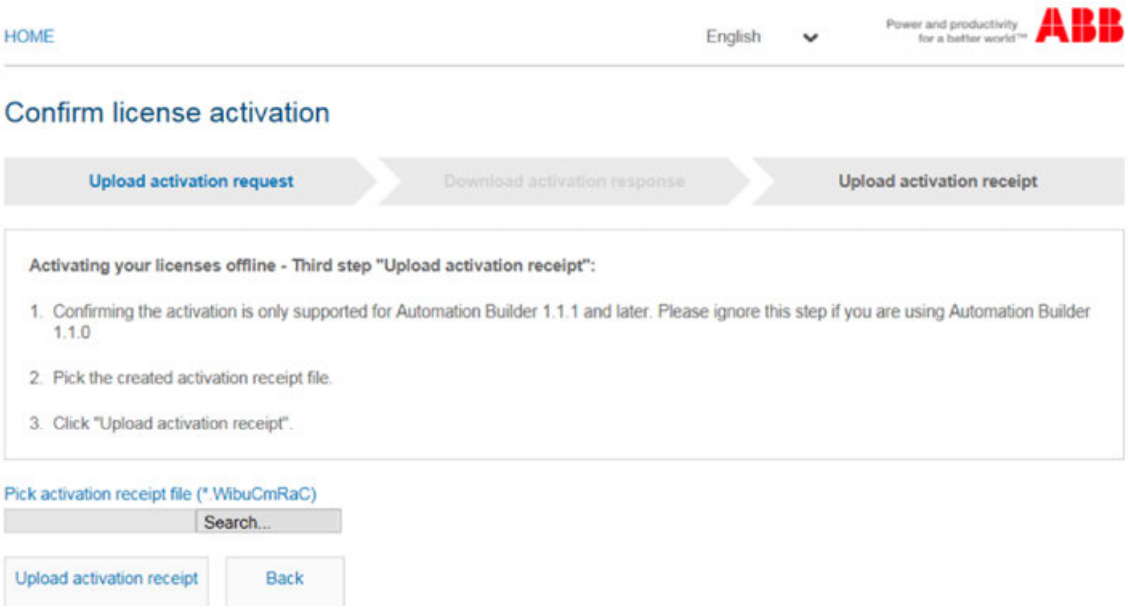
   ⇨ The license is validated by the ABB license server and afterwards activated on the PLC device.

   If the license shall be used on another PLC device, the installed license can be returned ☞ *Chapter 1.6.6.2.2.2.4 "Returning a license" on page 3671*.

7. To complete the licensing process, a license receipt file must be uploaded to the ABB license server.

   Save the license receipt file and upload it manually from a PC with internet connection to *http://lc.codemeter.com/32838/depot/index.php*.

   ⇨ A license confirmation is returned.



Offline activation

If the runtime licensing process was closed between saving the license activation request file and obtaining the license activation file from the ABB license server, perform an offline activation:

1. Right-click on the PLC node and select *"PLC runtime licensing"* from the *"Runtime Licensing"* menu.

   ⇨ A wizard starts. Follow the instructions.

2. Select the option *"Complete offline licensing process"*.

3. Select *"Browse"* and select the license activation file (*.WibuCmRaU) from the defined storage location.

   ⇨ The license is activated on the PLC device.

   If the license shall be used on another PLC device, the installed license can be returned ☞ *Chapter 1.6.6.2.2.2.4 "Returning a license" on page 3671*.

4. To complete the licensing process, a license receipt file must be uploaded to the ABB license server.

   Save the license receipt file and upload it manually from a PC with internet connection to *http://lc.codemeter.com/32838/depot/index.php*.

   ⇨ A license confirmation is returned.

Activating a demo license

It is possible to try out device features or library features by using a Demo license on the PLC. With this, you can use the features for a limited time period.

☑ PC and PLC device are connected. In case of no connection, perform the activation via memory card ↯ *Chapter 1.6.6.2.2.2.3 "Licensing via memory card" on page 3669*.

☑ There is a connection to the Internet. In case of no connection, perform the activation on another PC with Internet connection ↯ *Chapter 1.6.6.2.2.2.1.1 "Activation without internet connection" on page 3666*.

1. Right-click on the PLC node and select *"PLC runtime licensing"* from the *"Runtime Licensing"* menu.

   ⇨ A wizard is started. Follow the instructions.

2. Select the option *"Create a demo license"* and click *"Next"* to finish the licensing procedure.

   ⇨ The demo license is validated by the ABB license server and afterwards activated on the PLC device.

Licensing via memory card

When you have no connection between your PC and the PLC device the licensing procedure can be done via a memory card.

On the PC: Create a license request

☑ There is a connection to the internet.

☑ The memory card can be used with AC500 V3 products.

> **NOTICE!**
> If a SDCard.ini file is stored on the memory card, the file will be overwritten.

1. Place the memory card in the PC.
2. Right-click on the PLC node and select *"Prepare PLC license SD memory card"* from the *"Runtime Licensing"* menu.
3. From the filesystem select the root folder of the memory card.

   ⇨ A success message is displayed when the creation of the memory card files is completed.

   The license request files are stored to the selected folder.

On the PLC: Transfer the license data

1. Insert the memory card into the PLC device and reboot the PLC.

   ⇨ When the license request file is successfully created by the PLC, "done" is shown on the display of the PLC.

2. Remove the memory card from the PLC.

On the PC: Enter the license activation key

For this action, internet connection is required.

1. Place the memory card into the PC.
2. Open the PLC project in Automation Builder. Ensure the PLC is logged out.
3. Right-click on the PLC node and select *"PLC runtime licensing"* from the *"Runtime Licensing"* menu.
   ⇨ A wizard is started. Follow the instructions.
4. Enter the license activation key.
5. From the filesystem, select the root folder of the memory card.
   ⇨ The previously created license request files are sent to the ABB license server. A license activation is created on the memory card.
6. Remove the memory card from the PC.

On the PLC: Complete license activation for the PLC

1. Insert the memory card into the PLC device and reboot the PLC.
   ⇨ *done* is displayed on the PLC if license activation was successful.
2. Remove the memory card from the PLC

Trying to activate a runtime license, that has already been activated (e.g. via an online conncetion), will result in following error in the sdcard.rdy file:

;Result of license file import
ImportLicense=13;Internal Error (1)

On the PC: Complete license activation on the license server

For this action, internet connection is required.

To complete the licensing process, the license receipt file must be uploaded to the ABB license server.

1. Place the memory card into the PC.
2. Upload the license receipt file manually from a PC with internet connection to *http://lc.codemeter.com/32838/depot/index.php*.

   The license receipt on the memory card is located in the subfolder license

   ⇨ A license confirmation is returned.

Returning a license

NOTICE!
After returning a AC500 runtime license, the PLC system moves into 'Stop' mode after 24 h.

Returning a license without memory card

A license which has been installed on a PLC device can be returned and installed on another PLC device.

☑ PC and PLC device are connected. In case of no connection, perform the activation via memory card ↳ *"Returning a license via memory card" on page 3671*.

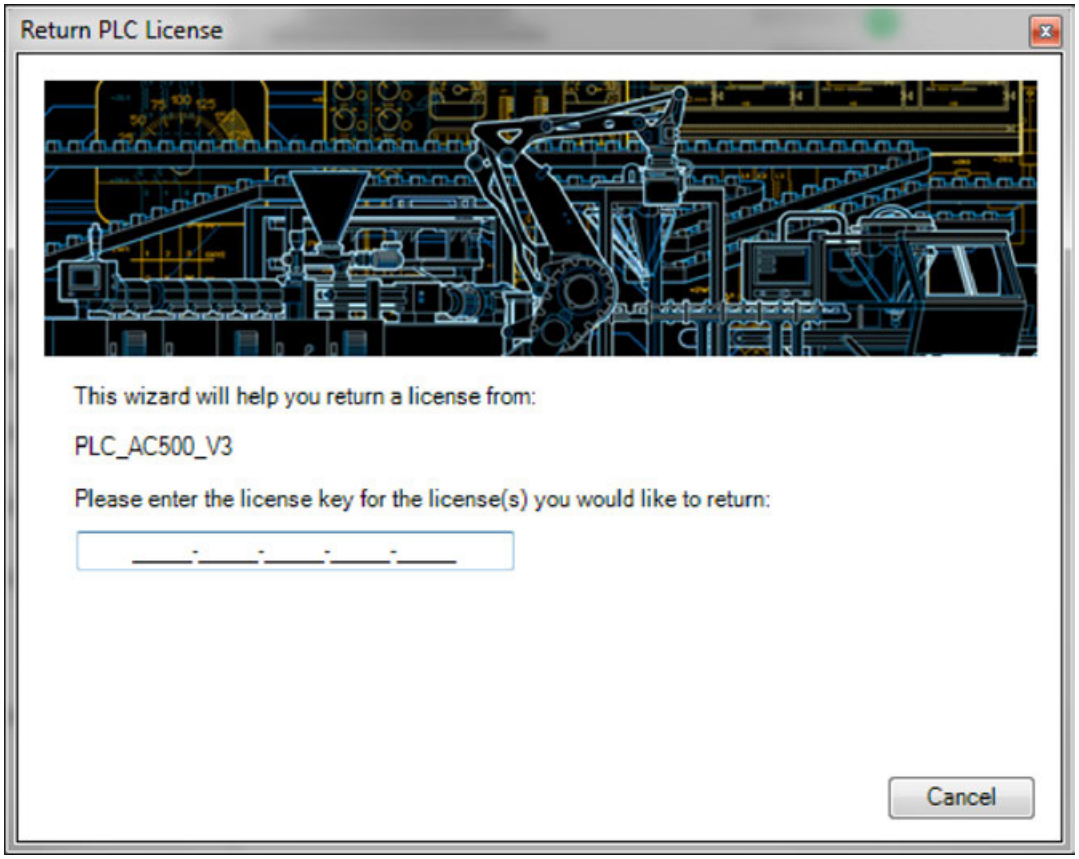
1. Right-click on the PLC node and select *"Return active license"* from the *"Runtime Licensing"* menu.
   ⇨ A wizard is started. Follow the instructions.
2. Enter the license activation key and click *"Return license"*.
   ⇨ The results of the return process will be displayed in the dialog.
   The license from the PLC device is removed and can be used now for another PLC device.

Returning a license via memory card

When the PLC is not connected to the PC (PLC logged out) it is possible to return a license via memory card.

1. Insert the memory card in the PC and execute *"Runtime Licensing ➔ Prepare PLC license SD card "* on the PLC node.
2. Place the memory card into the PLC.
3. Perform *"power cycle"* after a successful update reboot the PLC and connect to the PLC.
   ⇨ The License is removed from the PLC.
4. Place the memory card into the PC.
5. Right-click on the PLC node and select *"Return active license"* from the *"Runtime Licensing"* menu.
   ⇨ A wizard is started. Follow the instructions.

6. Enter the license activation key and click *“Return license”*.



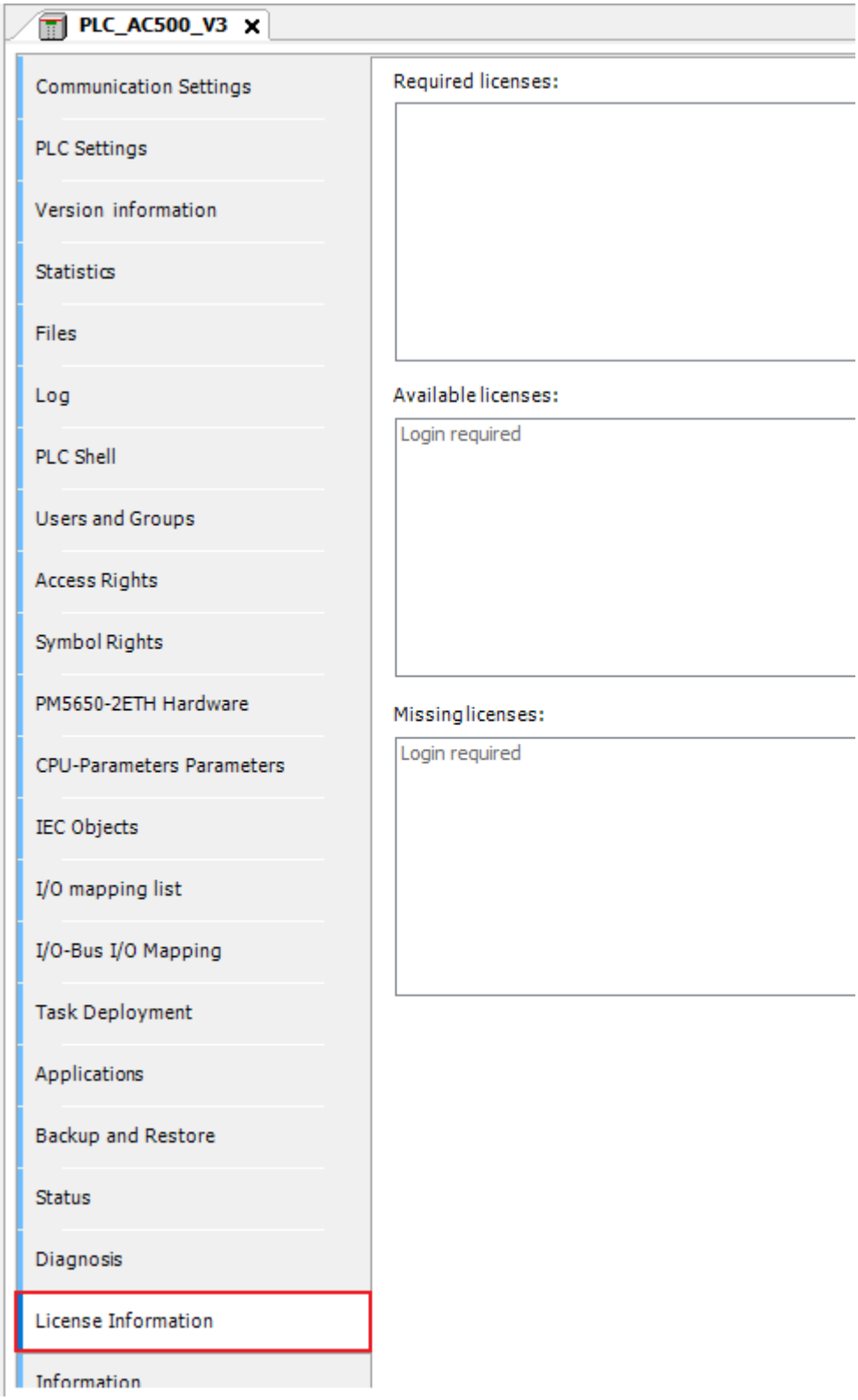
7. Click *“Browse”* and select the root folder of the memory card.
   ⇨ Returning of the license is started.
8. Place the memory card in the PLC device and reboot the PLC.
   ⇨ The license from the PLC device is removed and can be used now for another PLC device.
9. To complete the licensing process, a license receipt file must be uploaded to the ABB license server.
   Save the license receipt file and upload it manually from a PC with internet connection to *http://lc.codemeter.com/32838/depot/index.php*.
   ⇨ A license confirmation is returned.

View license information

To view the license information of AC500 V3 products:

1. In the Automation Builder device tree double-click on the PLC node.
   ⇨ The PLC tab is opened.

2. In the PLC tab select *“License Information”*.



⇨ The project is scanned for required licenses.

If you are logged into a PLC, then the licenses available on the PLC are displayed.

Missing required licenses are highlighted.

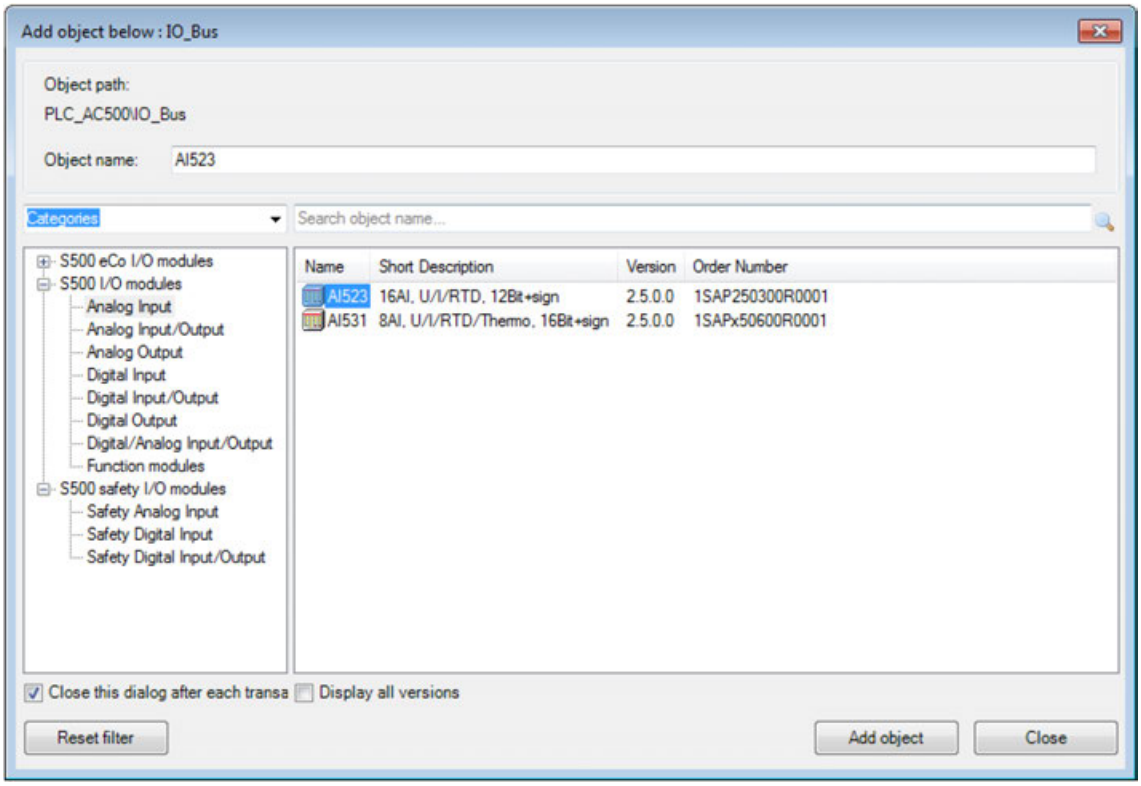
Connection of devices

All installed devices that are available in Automation Builder are listed in the *Chapter 1.6.6.2.1 “Device repository” on page 3662*.

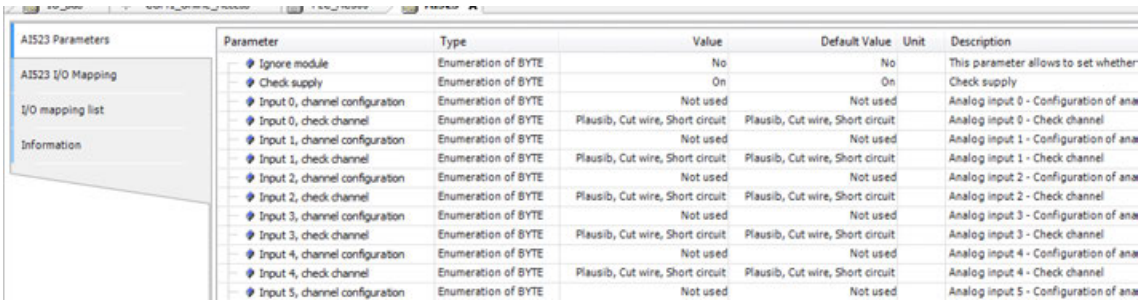
Configuring devices

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

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



2. Select the desired object and click *[Add object]*.
3. Double-click the new object in the device tree to configure the device settings. Depending on the selected item different configuration tabs are available.

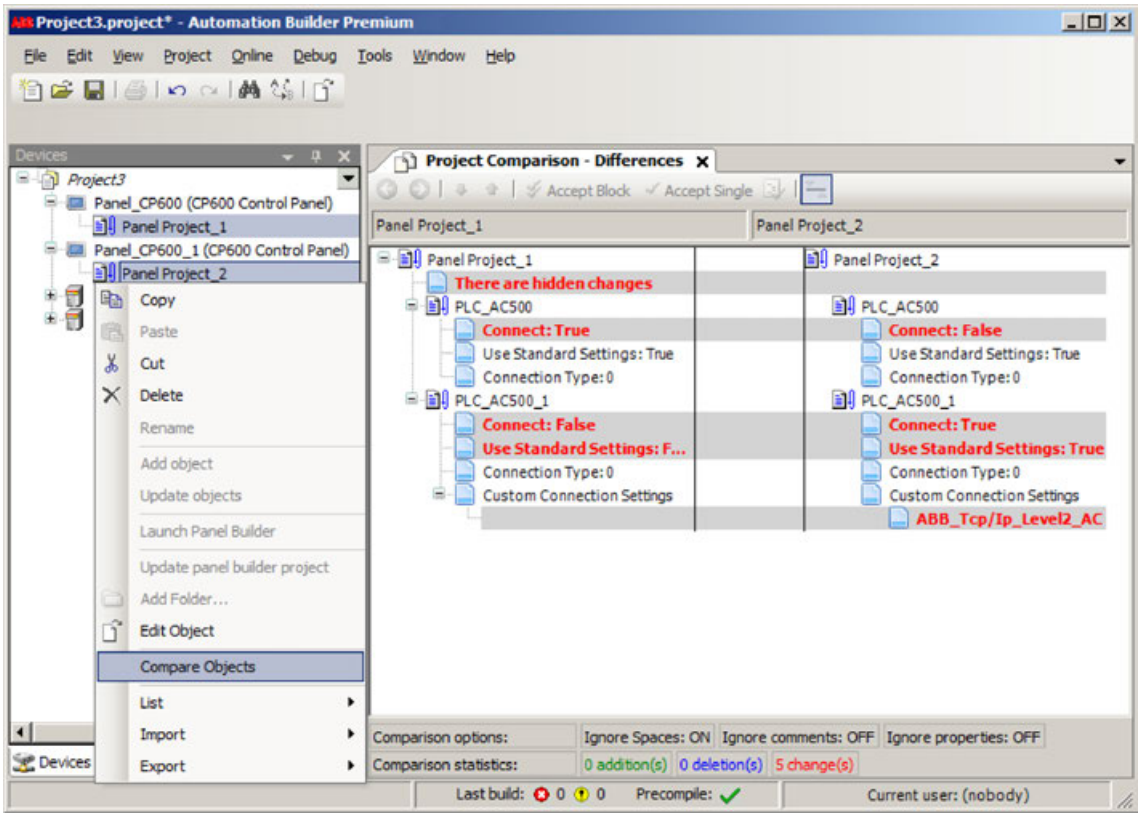


Update of AC500 devices

Perform a firmware update to update AC500 V3 devices. *Chapter 1.6.6.1.4.2 “ AC500 V3 firmware installation and update” on page 3653*

Comparing objects

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



IP settings

Configuration of the IP settings with the LED display

The IP settings for the PLC can be set directly on the processor module via keypad and LED display.

See *Chapter 1.6.5.1.6.5.3 “Configuration” on page 3493*

Configuration of the IP settings with the IP configuration tool

The IP address for AC500 devices can be set or changed in Automation Builder using

- the IP configuration tool which is described in the following.
- the 'Communication Settings'. *Chapter 1.6.6.2.2.4.3 “Configuration of communication via Ethernet (TCP/IP)” on page 3688*

As an alternative the IP address can be changed at the hardware device itself. *Chapter 1.6.5.1.6.5 “Description of the function keys” on page 3491*

The IP configuration tool:

The IP configuration tool can be used

- to set or change the IP address of devices.
  Chapter 1.6.6.2.2.4.2.2.2 “Changing the IP address” on page 3680
- to scan the network for available hardware devices.
  Chapter 1.6.6.2.2.4.2.2.1 “Network scan” on page 3678

- to update the firmware of devices.
  This functionality is only supported if the IP configuration tool is used stand-alone.
  ↳ *Chapter 1.6.6.2.2.4.2.2.3 “Firmware update” on page 3681*
- to activate certain functionality on hardware devices.
  This feature is only available on AC500 V3 devices.
  ↳ *Chapter 1.6.6.2.2.4.2.2.4 “Blink functionality” on page 3685*

The IP configuration tool is part of Automation Builder and can be called via *“Tools ➔ IP-Configuration”*.

Further the IP configuration tool can be used stand-alone without an Automation Builder application running. The stand-alone variant requires a separate installation via the Installation Manager ↳ *Chapter 1.6.6.2.2.4.2.1 “Stand-alone installation” on page 3676*.

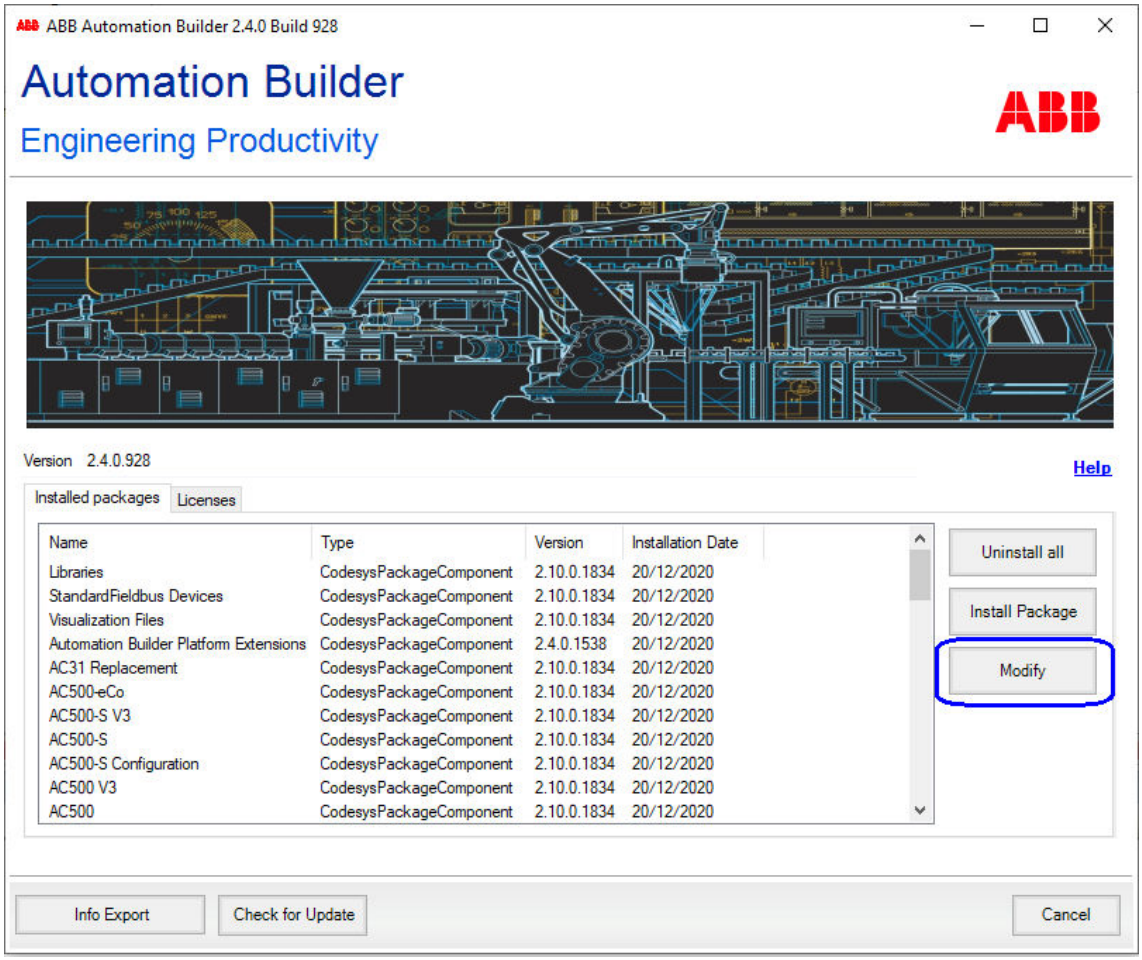
After the installation, the IP configuration tool is started via .exe file / desktop icon.

Some functionality is only supported if the IP configuration tool is used stand-alone, e.g. for firmware updates for communication interface devices.

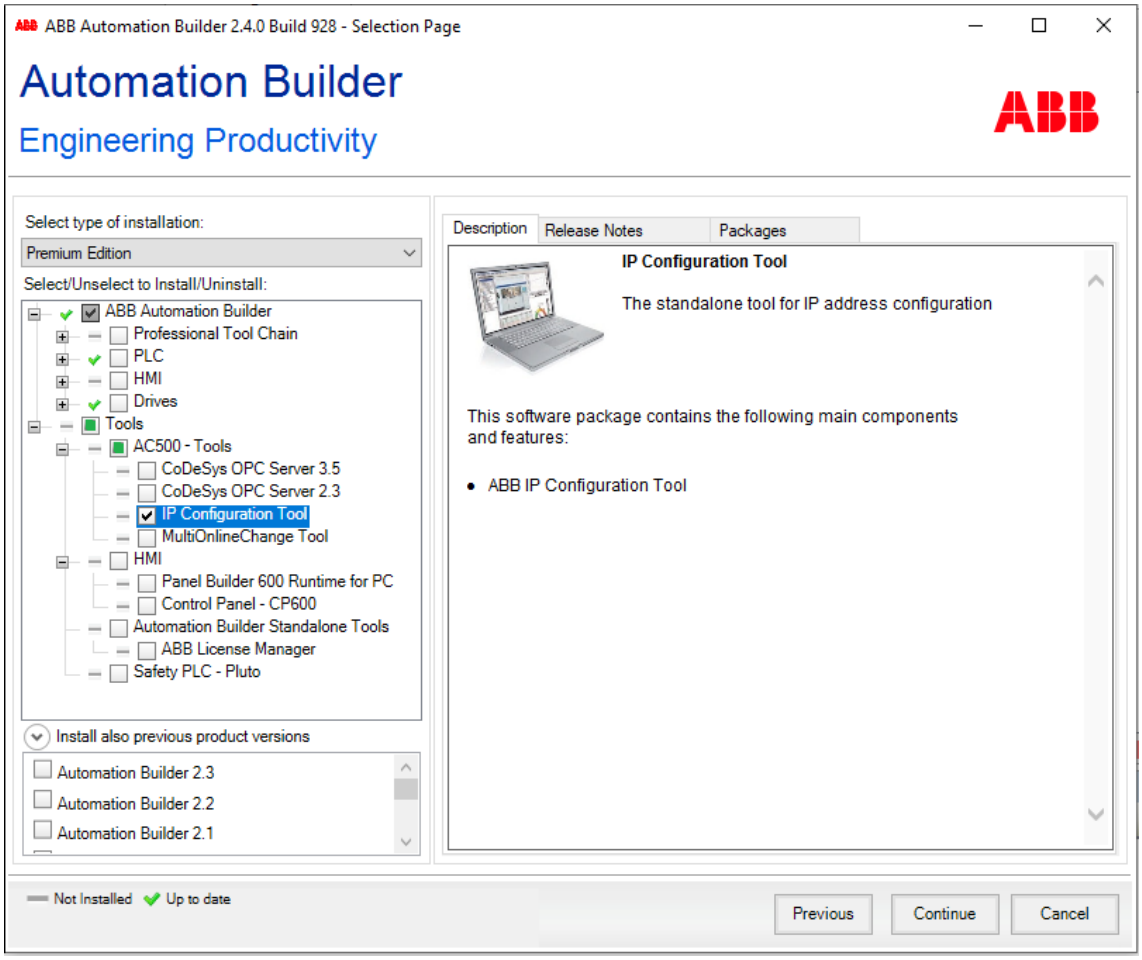
Stand-alone installation

The IP configuration tool is part of Automation Builder and can be called via “Tools ➔ IP-Configuration”. A separate installation is only required if the IP configuration tool shall be used stand-alone.

1. Open the Installation Manager in Automation Builder: *"Tools ➔ Installation Manager"*.
2. Close all other instances of Automation Builder as only one instance of the program can be executed at a time.



3. Click *"Modify"* and select the *"IP Configuration Tool"* from the structure tree.



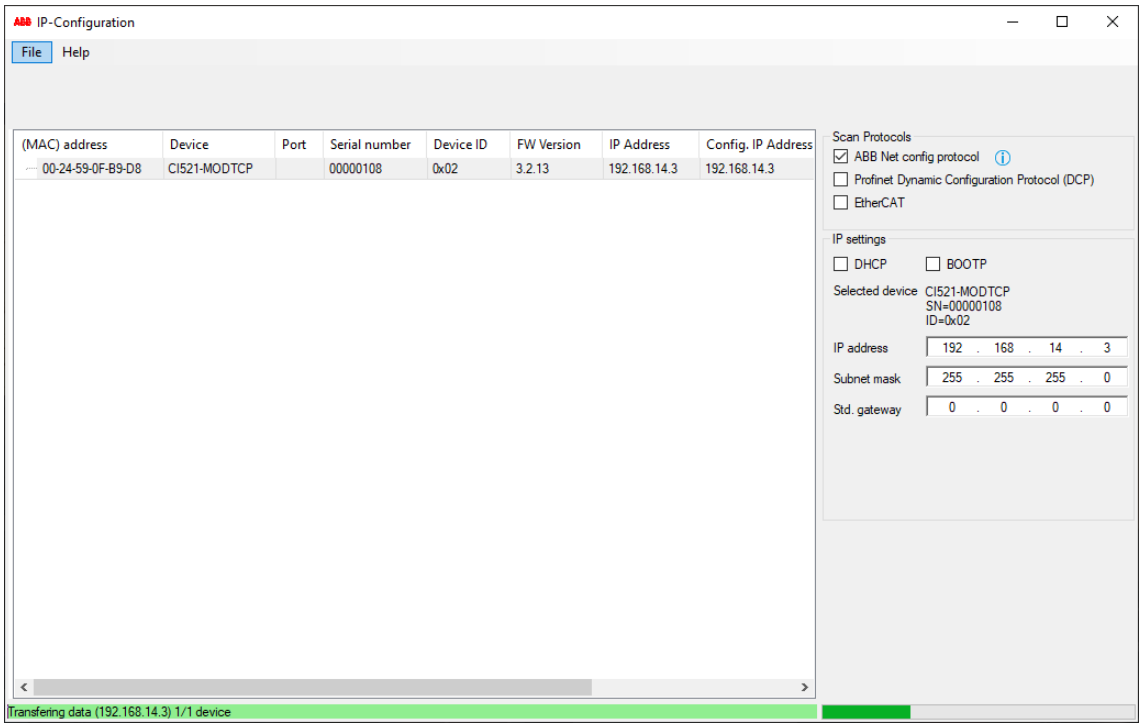
4. Click *"Continue"* to start the installation.
   - ⇨ After a successful installation the IP configuration tool is available as stand-alone tool (.exe).
   - ⇨ To start the IP configuration tool, click the new created desktop icon.

Using the tool functions

Network scan

With a network scan all devices that have been found in the network by the scan process are listed, i.e. ABB devices such as AC500 processor modules, AC500 communication interface modules or ABB Drives.

1. Start the IP configuration tool in Automation Builder (*"Tools ➔ IP-Configuration"*) or start it stand-alone (.exe).
2. The *"IP-Configuration"* dialog opens. Define the device type for the network scan by selecting the desired option under *"Scan Protocol"*:
   - *"ABB Net config protocol"*:
     Use this option for AC500 devices such as processor modules, CI5xx-Modbus devices or ABB Drives. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection.
   - *"Profinet Dynamic Configuration Protocol (DCP)"*:
     Use this option for PROFINET communication interface modules. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection (not via CM579).
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 3683
   - *"EtherCAT"*:
     Use this option for EtherCAT communication interface modules. The Ethernet cable must be connected directly to the first EtherCAT slave device of the EtherCAT fieldbus. Ensure that no EtherCAT master device is available on the bus when a scan is performed.
     "Emergency" option: Enable this option to check on failures in the EtherCAT assembly during the scan process, i.e. a frame loss or interchanged ports. Errors are displayed.
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 3683
3. Click *[Scan]* to start the scan process.



4. All devices that have been found in the network are listed including hardware and connection details. The following details can be changed under *"IP settings"*:

   ⇨ ● *"IP Address"*:
   Current IP address of the device.
   ● *"Conf. IP Address"*:
   Configured IP address of the device. A changed IP address will update this column.
   ● *"FW Version"*:
   Current installed firmware version of the device. This field is visible not until a first network scan. If this field is still empty after a network scan, check on connection errors.
   Chapter 1.6.6.2.2.4.2.3.1 "Trouble-shooting for firmware update" on page 3686

The IP address of some devices, e.g. EtherCAT devices cannot be changed.

Changing the IP address

1. In order to change the IP address of devices perform a network scan.

   Chapter 1.6.6.2.2.4.2.2.1 "Network scan" on page 3678

2. Select a device from the list and select the appropriate protocol under *"Scan protocol"*.

   "DHCP" or *"BOOTP"* option: If required, DHCP or BOOTP can be used to receive the IP address for the device from the server.

   "IP address", *"subnet mask"*, *"Std. gateway"*: Use these fields to change the IP address settings including the settings for the subnet mask and the standard gateway. Ensure that the combination of connection settings is correct.

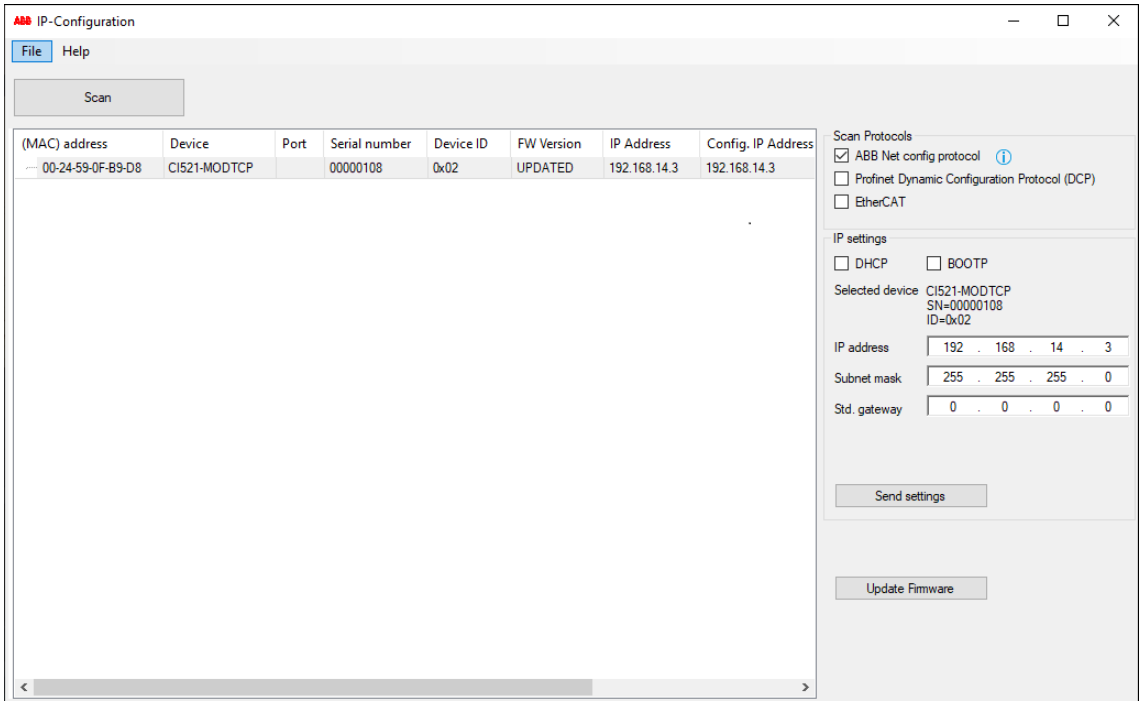
   "Check subnet configuration" on page 3686

Note for CI52x-Modbus devices

Consider the behavior of CI52x-Modbus devices if the last number of the IP address is set to "0".

"Check last number of IP address" on page 3687

3. Change the settings for the IP configuration and click *[Send settings]* to transmit the data to the device.



⇨

Note for PROFINET devices

The device name of PROFINET devices can be edited. If changing the name, ensure the following rules apply:

- *Labels must be separated by "."*
- *Total length: 1 to 240*
- *Label length: 1 to 63*
- *Labels can consist of characters [a-z] and numbers [0-9]*
- *Labels are not allowed to start with "-"*
- *Labels are not allowed to end with "-"*

4. In order to keep all IP changes after a power cycle, the settings can be stored permanently. Confirm the prompted message during the scan process.

Firmware update

The firmware of AC500 communication interface modules can be updated with the IP configuration tool.

For this, the IP configuration tool must be used as stand-alone variant.

↳ *Chapter 1.6.6.2.2.4.2.1 “Stand-alone installation” on page 3676*

It is not possible to perform a firmware update out of Automation Builder.

- *For PROFINET communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.3.3.*
- *For EtherCAT communication interface modules a firmware update is only supported for devices with firmware version ≥ 2.1.4.*
- *For Modbus communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.2.13.*

Requirements: **Before the firmware update**

- Ensure a fast and stable network connection
- Close all unused applications on the executing PC
- Stop the communication between AC500 PLC and the communication interface module that shall be updated

During the firmware update

- Do not close the IP configuration tool
- Do not open Automation Builder software or any other application
- Do not switch-off the communication interface module that shall be updated
- Do not disconnect the Ethernet connection of a communication interface module or the executing PC

The firmware update will stop the operation of the affected device(s). Hence, the device(s) will become unresponsive for 1 - 2 minutes.

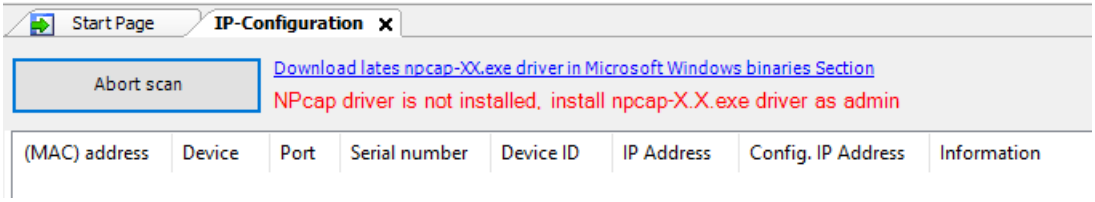
Procedure:

1. Start the IP configuration tool stand-alone (.exe).
2. Perform a network scan.

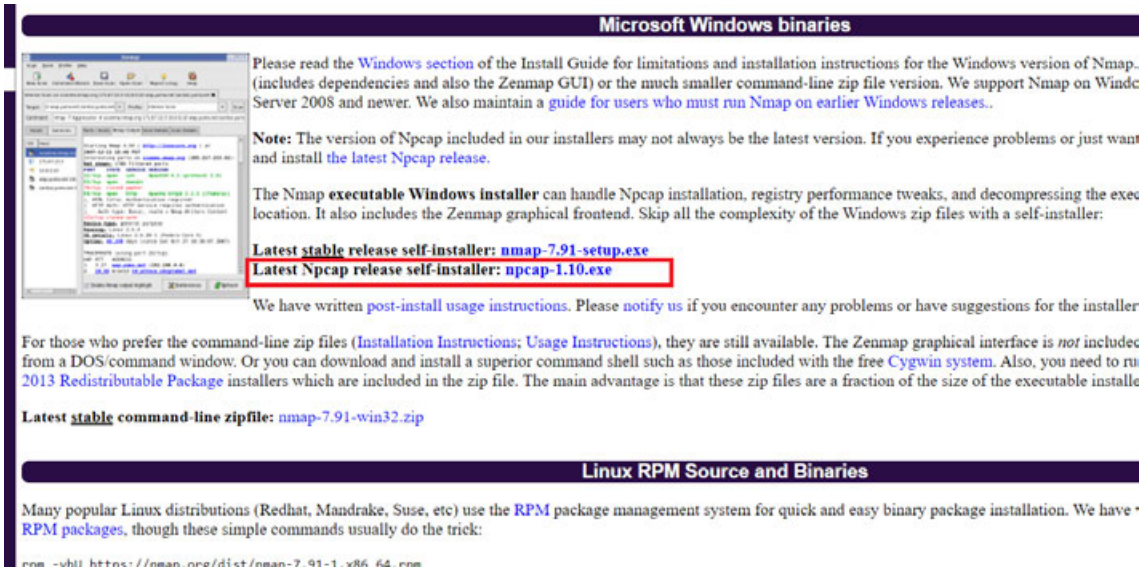
   Chapter 1.6.6.2.2.4.2.2.1 “Network scan” on page 3678
3. Select the devices that shall be updated from the list and click *[Scan]* to trigger the scan process.

   A multiple selection of several devices is possible via control key, however, ensure to select only devices of the same protocol at a time. Otherwise the firmware update fails.

4. This step is only required for devices that require an installed NPcap driver. In this case an appropriate message including a download link is prompted in the IP-Configuration dialog:



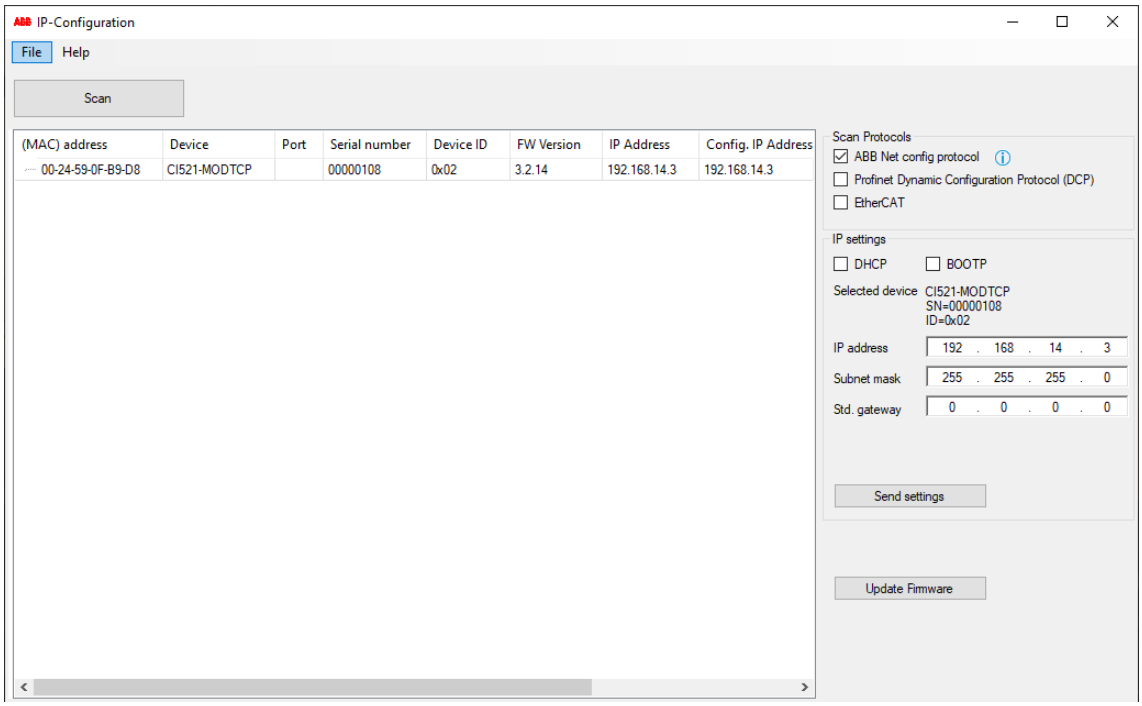
⇨ Click on the displayed link *https://nmap.org/download.html* and download the latest version of the *npcap-X.X.exe* file.



⇨ After the download, execute the file as administrator and restart the scan process.

⇨ The devices that have been scanned are listed.

5. Click *[Update Firmware]* to start the firmware update for the selected devices.



6. For CI50x, CI51x and CI52x devices a signature check is started. Select the appropriate firmware update file (*.bin) for the device(s). Example: C:\AC500\AC500_CI52x_Firmware_V3.2.8.bin.

   After a successful signature check the firmware update file (*.bin) and the respective signature file (*.bin.sig) are transferred to the device. This can last up to 3 minutes.

   If the signature check fails, check the availability of the *.bin file and the *bin.sig file.

   ↳ *"Signature check" on page 3687*

7. A status check followed by a device reboot followed by a second status check is performed automatically.

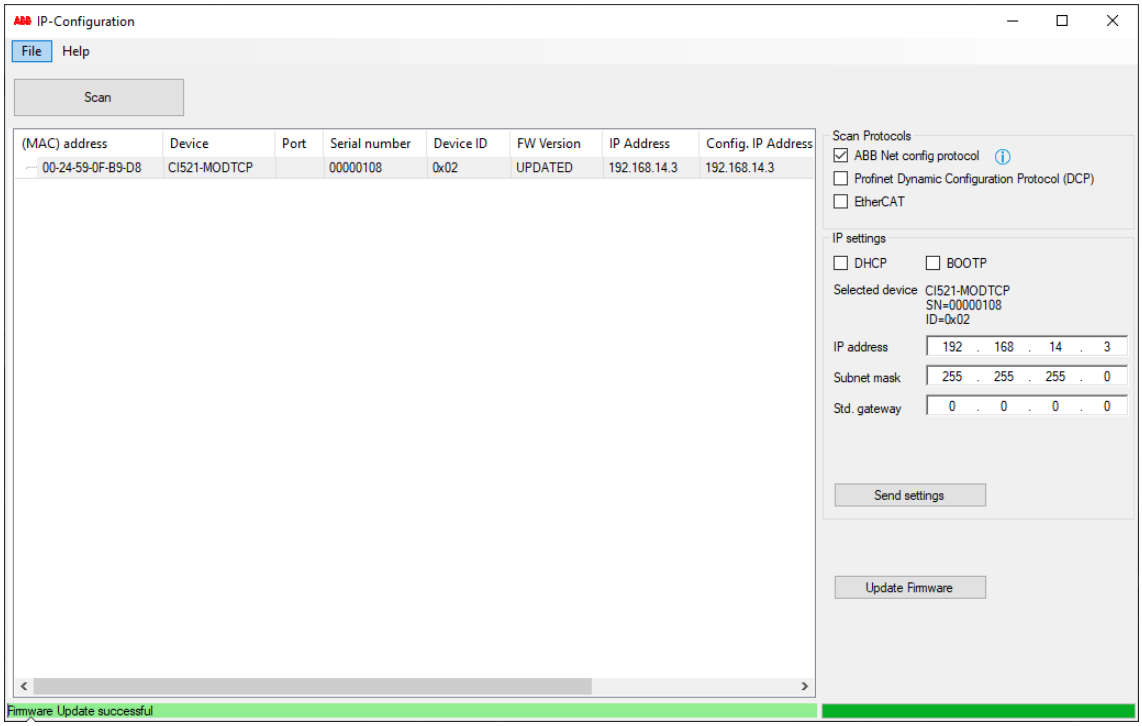
   > *After the firmware update all outputs of the updated devices are set to '0'.*

8. After a successful firmware update the update status or the new firmware version is displayed in the *"FW Version"* field.

   If this field is empty, there possibly is a connection error between the device and the executing PC.

   ↳ *"Error: Can't connect to device" on page 3688*

   Exception: For EtherCAT devices an empty *"FW Version"* field does not indicate a connection error.



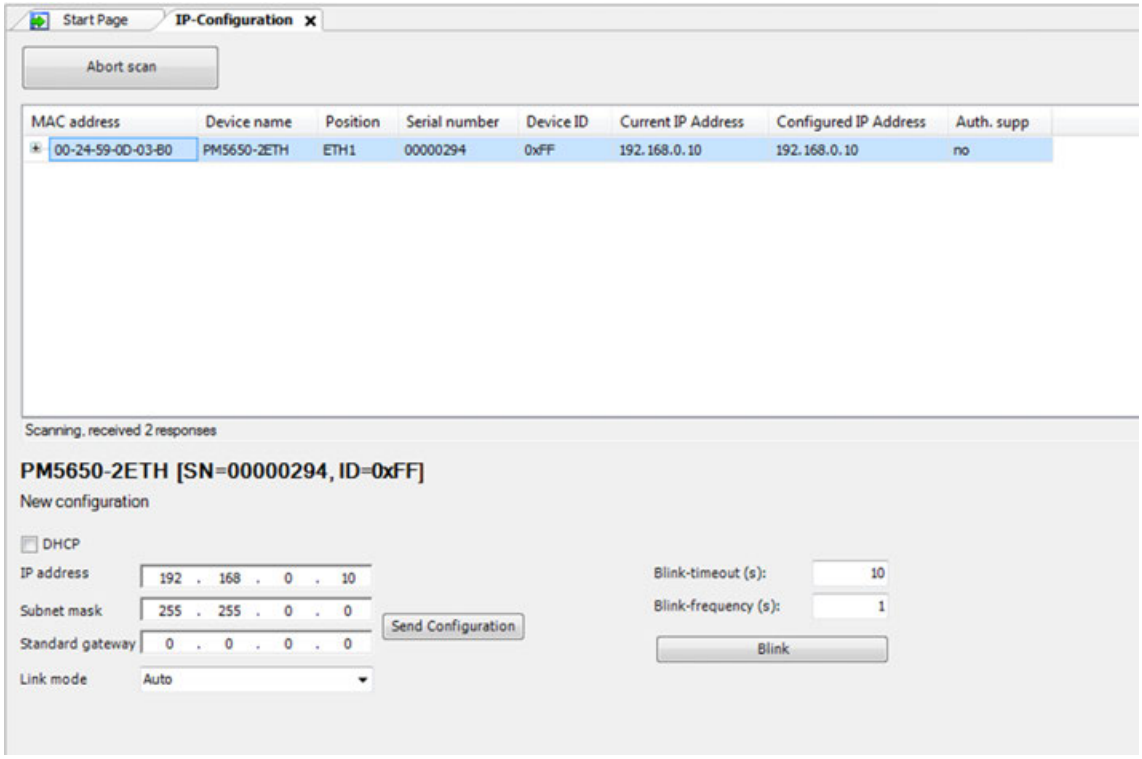
⇨ If the firmware update fails

- check the requirements for the update procedure.
  ↳ *"Requirements:" on page 3682*
- check the hints for trouble-shooting.
  ↳ *Chapter 1.6.6.2.2.4.2.3.1 "Trouble-shooting for firmware update" on page 3686*
- perform a network scan and repeat the update. If the error still persists power cycle the device and try the update again.

Blink functionality

This function activates flashing of the backlight of an AC500 LED display.

1. From the menu, select *"Tools ➔ IP-Configuration"*.
2. Click *[Scan]* to trigger the scan process for devices in the network.
   ⇨ A progress bar shows the progress. The IP settings of a selected device is displayed below the list and can be edited.
3. Adjust your desired time and click *[Blink]* to activate flashing.



Trouble-shooting for IP configuration tool

Firewall exceptions:

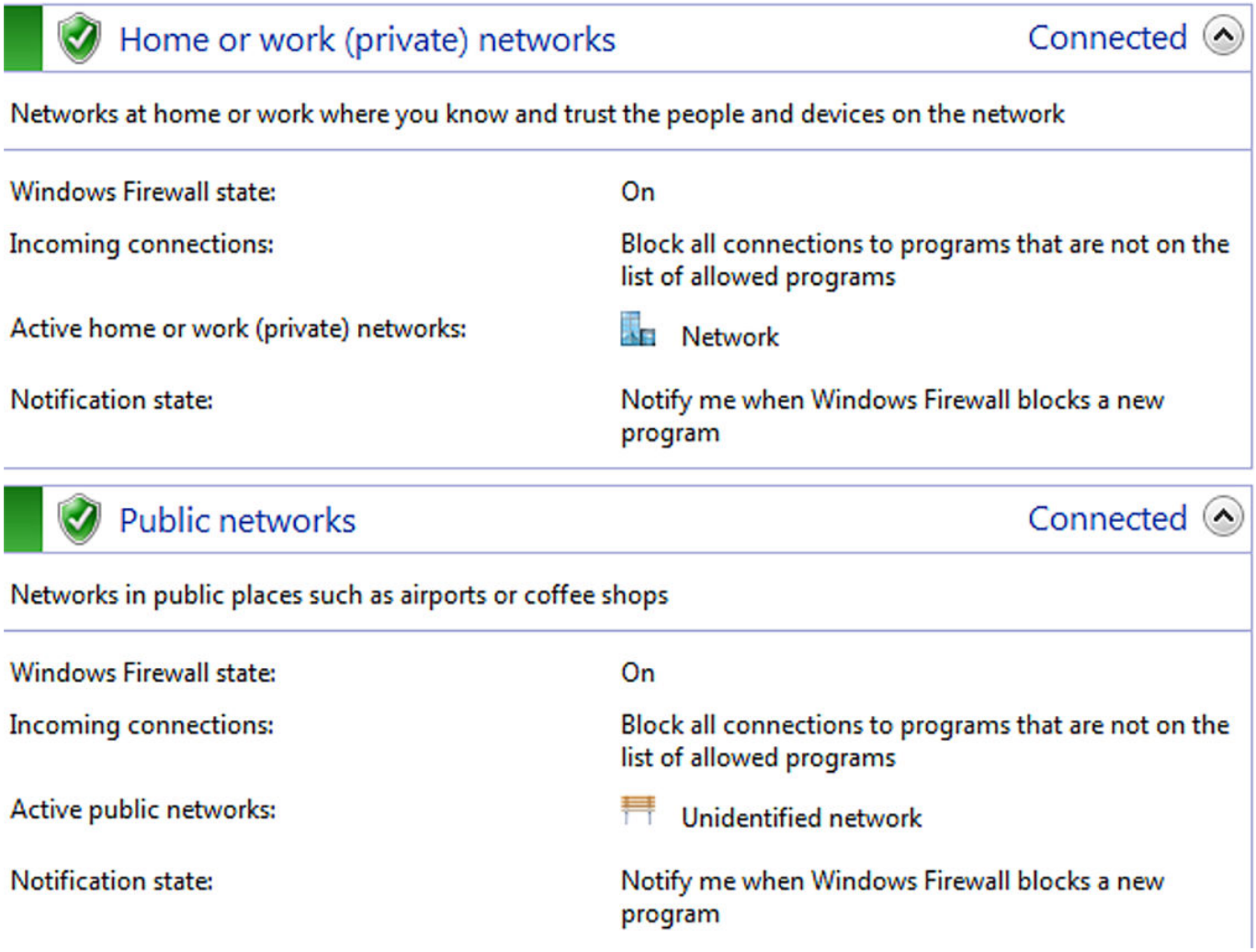
On a standard Windows 7 installation without third party firewall or security tools installed the IP configuration tool should work properly.

The Automation Builder setup installs rules or exceptions for the built-in Windows firewall to allow IPConfig to receive the responses for the IPConfig scan.

To check the Windows firewall is set correctly check the firewall settings.

Windows 7/ Windows 10:

On the network that is used for communication with the PLC, set *"Incoming connections"* to "Block all connections to programs that are not on the list of allowed programs".



If a third party firewall is used these exceptions must be configured manually.

> *Either exceptions for applications can be entered: Automation Builder and IP configuration tool must be added as application.*
>
> *Or the protocol and the port number must be given (for IPConfig: UDP protocol and port number 24576).*

Trouble-shooting for firmware update

Check the requirements

Ensure that all requirements have been considered before and during the update procedure.
“Requirements:” on page 3682

Check subnet configuration

This hint is only valid for Modbus devices and PROFINET devices.

If the *“FW Version”* field is empty after the network scan or if the firmware version has not been updated after the update procedure, there possibly is a connection error between the device and the executing PC.

Ping the device from the executing PC. If no connection can be established, check whether the device and the PC are in the same subnet.

Example

PC	Device	Result
192.168.14.71 / 255.255.255.0	192.168.14.10 / 255.255.255.0	OK
192.168.**10**.71 / 255.255.**255**.0	192.168.**14**.10 / 255.255.**255**.0	ERROR
192.168.10.71 / 255.255.0.0	192.168.14.10 / 255.255.0.0	OK

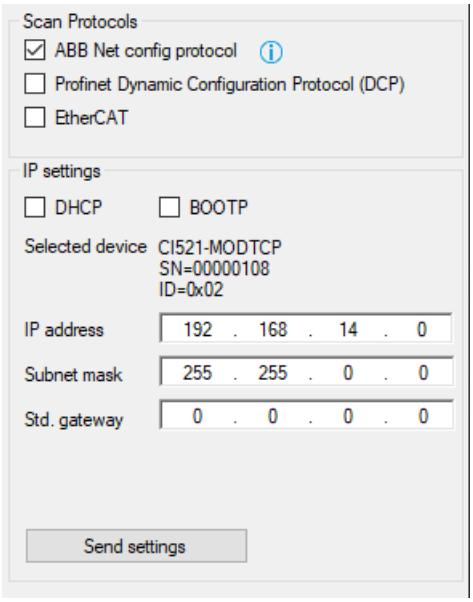
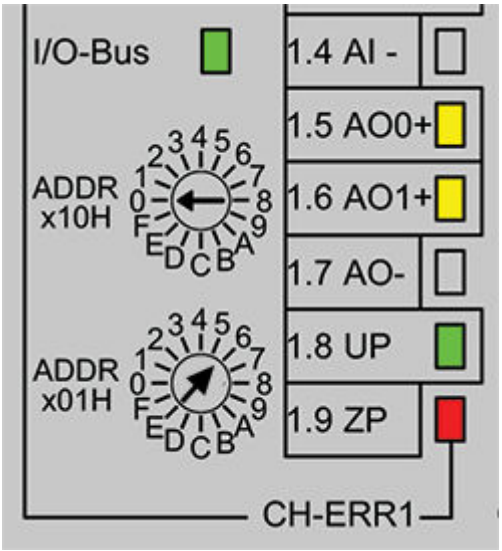
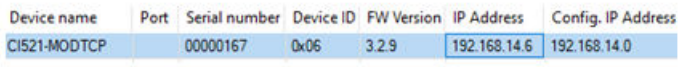
Click *[Scan]* again to restart the network scan. If the connection is successful a newer firmware version is displayed in the *"FW Version"* column.

Check last number of IP address

This hint is only valid for CI52x-Modbus devices.

Check the last number of the IP address. If it is set to "0", the IP address setting for this last number will be used from the rotary switches on the hardware device.

Example:

Automation Builder	AC500 communication interface module (rotary switch)
IP address: 192.168.14.0	IP address: 6
	
As a result, in the field *"IP Address"* the last number is set to "6": 	

Signature check

During the firmware update of CI50x, CI51x and CI52x devices a signature check is started.

The update procedure expects a firmware update file (**.bin*) and a signature file (**.bin.sig*) in the same directory. Without a signature file the signature check will fail.

Example:

Firmware update file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin***

Signature file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin.sig***

Error: Package timeout

A timeout error may occure due to an instable network.

Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

Error: Unable to read device status

A read error may occure due to errors in the firmware update protocol.

After the firmware update the IP configuration tool reads out the status of the updated device in order to check if the update was successful.

Error: IP is not unique

If an IP address is obtained by more than one device an error occures. A firmware update is not possible.

Error: Error State

Internal device error during the firmware update.

Solution:

Step 1: Scan again and repeat the firmware update.

Step 2: If this does not work, power cycle the device, scan again and repeat the firmware update.

Error: Can't connect to device

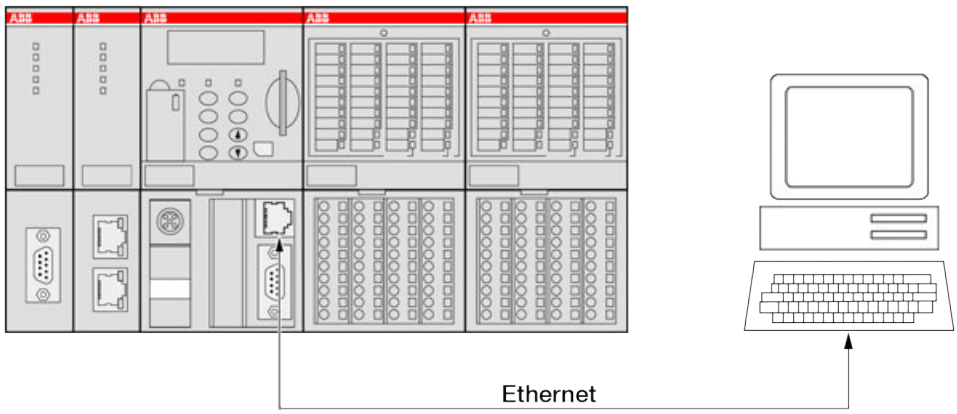
The TCP communication is not sufficient. Increase the connection quality.

Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

Configuration of communication via Ethernet (TCP/IP)

Programming via Ethernet is only possible on a PC with Ethernet board and installed network. Programming can be done via the internal (onboard) Ethernet communication module.

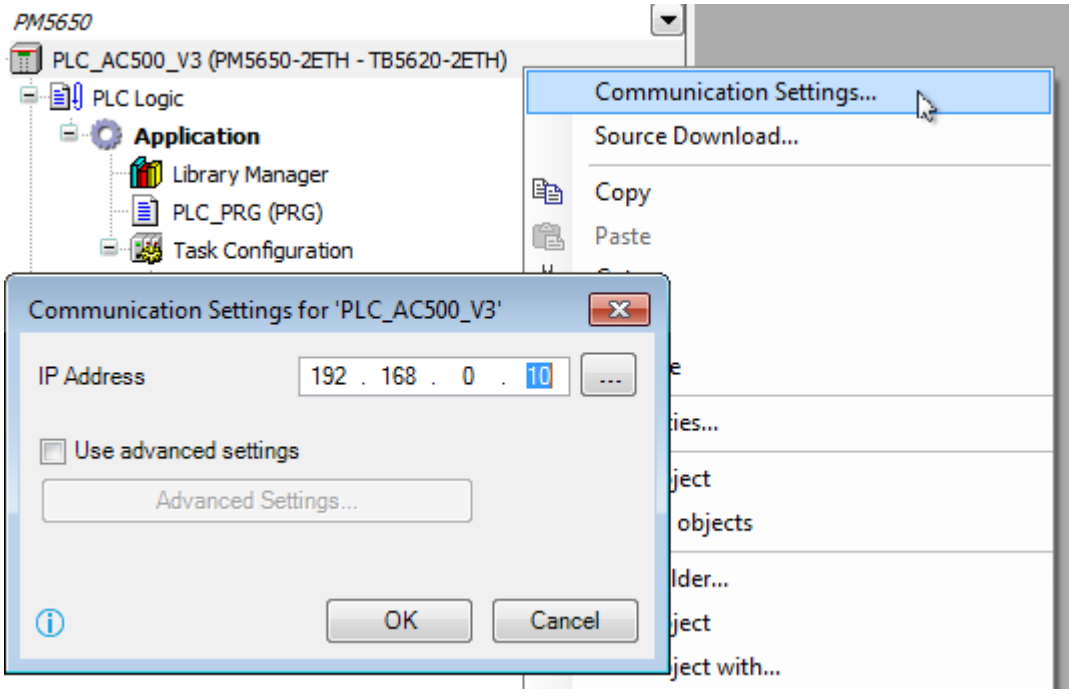
An application note describes the configuration of an AC500 V3 PLC for *EtherNet/IP communication* ↳ *Chapter 1.4.2.4 "EtherNet/IP Configurator" on page 1220*.



Enter a known PLC IP address

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.

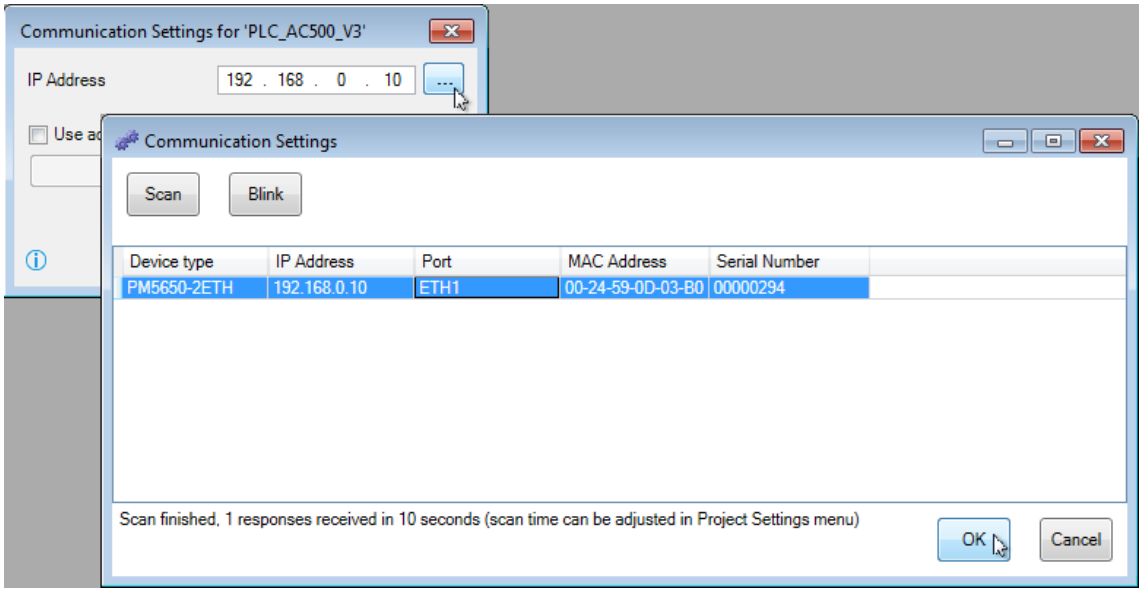


2. Enter your PLC *IP Address* and click *[OK]*.

Enter PLC IP address by scanning devices

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.

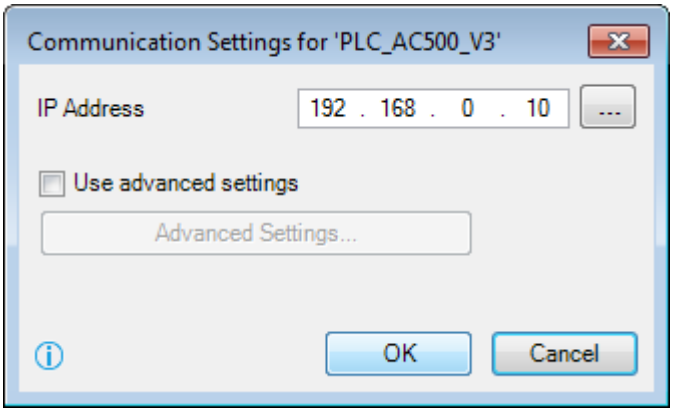


2. Click *[...]*.

   ⇨ Dialog box *Communication Settings <...>* appears.

3. Click *[Scan]*, select your desired PLC and click *[OK]*.
   ⇨ Entry is transferred to the dialog box *Communication Settings* <...>.

   Click *[OK]*.

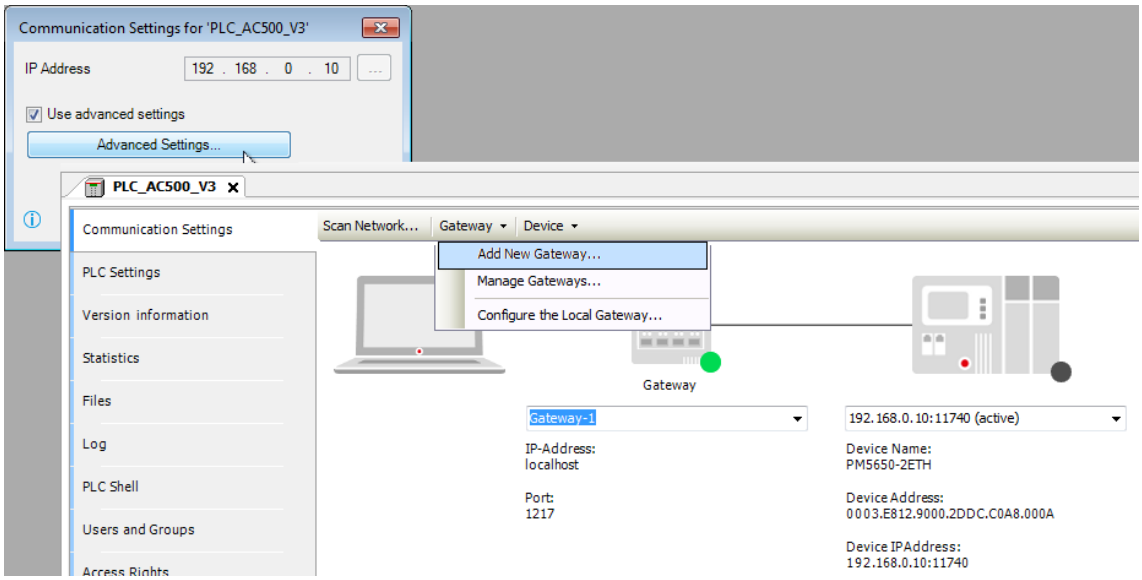


4. Click to log in the *"PLC_AC500_V3"* project.

Enter PLC IP address by *[Advanced Settings...]*

If a remote gateway instead of a local one has to be used it can be configured in the *[Advanced Settings...]*.

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.
   ⇨ Dialog box *Communication Settings* <...> appears.



2. Enable checkbox*Use advanced settings* and click *[Advanced Settings...]*.
   ⇨ Tab *"Communication Settings"* opens.
3. Check gateway or change if required.
   ⇨ Successful connection is indicated by green dot on the gateway icon.

4.

Manual entry of the IP address.

Check IP adress or change if required.

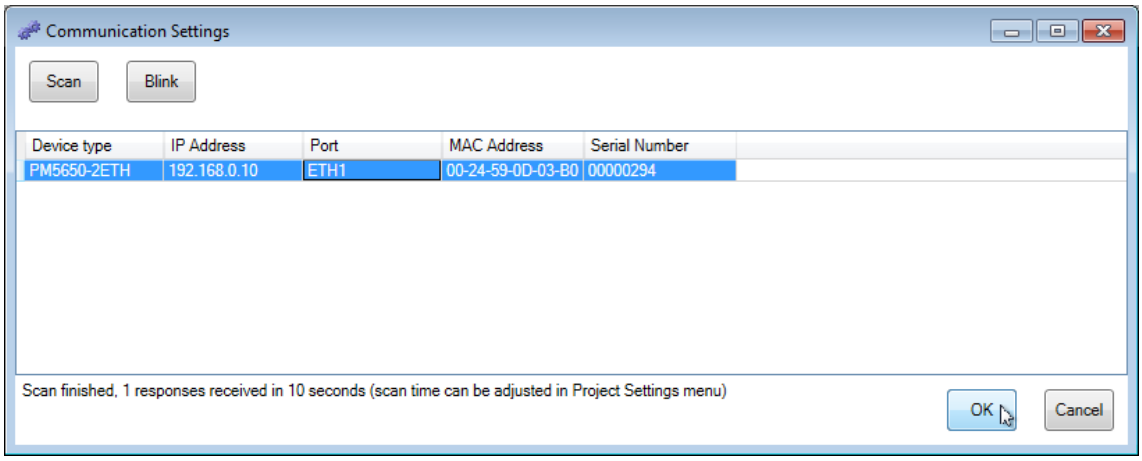
5. Press ENTER to confirm changed IP address.

   ⇨ Successful communication is indicated by green dot on the PLC icon.

6. Or instead of the last two steps:

Set the IP address via a scan.

Click *[Scan Network]*, select your desired PLC and click *[OK]*.



   ⇨ Successful connection is indicated by green dot on the gateway icon.

7. Click to log in the *"PLC_AC500_V3"* project.

1.6.6.2.3 Processor modules

Configure a processor module in the device tree

1. Add a processor module to your project. ↳ *Chapter 1.6.6.1.1.1 “Creating a new project ” on page 3632*
2. Double-click the PLC node in the device tree.
   ⇨ This will open a new window with tabs for the device configuration:
   - *“Communication Settings”* ↳ *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840*
   - *“PLC Settings”* ↳ *Chapter 1.4.1.20.2.8.9 “Tab 'PLC Settings'” on page 850*
   - *“Version information”* ↳ *Chapter 1.6.6.1.4.1 “Version information” on page 3652*
   - *“Statistics”* ↳ *Chapter 1.7.2.4.2 “Statistics” on page 4053*
   - *“Files”* ↳ *Chapter 1.4.1.20.2.8.7 “Tab 'Files'” on page 848*
   - *“Log”* ↳ *Chapter 1.7.2.4.3 “Log” on page 4053*
   - *“PLC Shell”* ↳ *Chapter 1.6.6.4.4 “PLC shell commands” on page 3950*
   - *“Users and Groups”* ↳ *Chapter 1.4.1.20.2.8.13 “Tab 'Users and Groups'” on page 860*
   - *“Access Rights ”* ↳ *Chapter 1.4.1.20.2.8.14 “Tab 'Access Rights'” on page 863*
   - *“Symbol Rights”* ↳ *Chapter 1.4.1.20.2.8.15 “Tab 'Symbol Rights'” on page 868*
   - *“PM5xxx Hardware”* ↳ *Chapter 1.6.6.2.3.2 “Changing the processor module type” on page 3694*
   - *“CPU-Parameters Parameters”* ↳ *Chapter 1.4.1.20.2.8.3 “Tab 'Parameters'” on page 844*
   - *“IEC Objects”* ↳ *Chapter 1.4.1.20.2.8.12 “Tab '<device name> IEC Objects'” on page 859*
   - *“I/O mapping list”* ↳ *Chapter 1.6.6.2.13.8 “I/O mapping list” on page 3777*
   - *“I/O-Bus I/O Mapping”* ↳ *Chapter 1.7.2.5 “Live values in views with I/O components” on page 4056*
   - *“Task Deployment”* ↳ *Chapter 1.4.1.20.2.8.17 “Tab 'Task deployment'” on page 869*
   - *“Applications”* ↳ *Chapter 1.4.1.20.2.8.4 “Tab 'Applications'” on page 845*
   - *“Backup and Restore”* ↳ *Chapter 1.4.1.20.2.8.5 “Tab 'Backup and Restore'” on page 846*
   - *“Status”* ↳ *Chapter 1.7.2.4.5 “Status” on page 4055*
   - *“Diagnosis”* ↳ *Chapter 1.7.1.3.4 “Device diagnosis” on page 4018*
   - *“Diagnosis History”* ↳ *Chapter 1.7.1.3.5 “Diagnosis history” on page 4019*
   - *“License Information”* ↳ *Chapter 1.6.6.2.2.2.5 “View license information” on page 3672*
   - *“Information”* General information about the device (name, vendor, version etc.)
3. Select the *“CPU-Parameters Parameters”* tab to configure the parameters for the processor module. ↳ *Table on page 3761*
4. Use the *“PM5xxx Hardware”* tab for later on changes of *“Terminal Base Type”* or *“Processor Module Type”* ↳ *Chapter 1.6.6.2.3.2 “Changing the processor module type” on page 3694*.
5. Select the *“I/O mapping list”* tab to create mapping variables with better usability support compared to the tree structured view. ↳ *Chapter 1.6.6.2.13.8 “I/O mapping list” on page 3777*
6. Select the *“Backup and Restore”* tab to create a backup or restore the project. ↳ *Chapter 1.4.1.20.2.8.5 “Tab 'Backup and Restore'” on page 846*
7. Select the *“Diagnosis”* tab to know what errors have occurred in the project. ↳ *Chapter 1.7.1.3.4 “Device diagnosis” on page 4018*

Parameter	Default	Value	Description
Error LED	On	On	The error LED lights up for errors of all classes, no fail-safe function activated.
		Off by E4	Warnings (E4) are not indicated by the error LED, no fail-safe function activated.
		Off by E3	Warnings (E4) and minor errors (E3) are not indicated by the error LED, no fail-safe function activated.
		POU control	Control LED ERR with POU PmErrLedSet
Check battery	On	On	The presence of the battery and the battery status are checked. If no battery is available or the battery is empty, a warning (E4) is generated and the ERR LED lights up.
		Off	The presence of the battery is not checked. No warning (E4) is generated. The LCD display "Batt" (triangle) can not be acknowledged! This also applies if a battery is installed but empty.
Stop on error class	Diagnosis of at least error class 2	Diagnosis of at least error class 2	In case of a fatal or severe error (E1-E2), the user program is stopped.
		Diagnosis of at least error class 3	In case of a fatal, severe or minor error (E1-E3), the user program is stopped.
		Diagnosis of at least error class 4	In case of a fatal, severe or minor error (E1-E3) or a warning (E4) the user program is stopped.
Diagnosis - Add PLC name to node name	Off	Off	Diagnosis - Add PLC name to node name.
		On	
PLC behavior after voltage dip	Halt	Halt	Behavior of the PLC after short voltage dip: reboot or halt.
		Reboot	
Diagnosis history	On	On	Enable the diagnosis history.
		Off	Disable the diagnosis history.
Max. Diagnosis history entries	1000	1000	Max. number of entries kept by the diagnosis history.
Missed cycle behavior	Next	Next	Skip the current cycle and start task in time on next cycle.
		ASAP	Start the task immediately.
Communication Schema	Default	Default	Balanced priority for communication via communication modules (CMs) and onboard Ethernet communication.
		Communication modules	Priority and high performance for communication module (CM) based communication via sync tasks. Lower priority for onboard Ethernet and local I/O bus.
		Onboard Ethernet	Priority for onboard Ethernet communication (e.g. via Modbus TCP). Lower priority for communication via communication modules (CMs).

Parameter	Default	Value	Description
		Real time onboard Ethernet	Very high priority for onboard Ethernet communication (e.g. EtherCAT PROFINET Ethernet). Low priority for communication via communication modules (CMs).
Automated reboot after E2 error	Off	Off	Not automated reboot after E2 error.
		On	Automated reboot after E2 error.

Changing the processor module type

In a project, you can change the target system by changing the type of processor module or terminal base type. If possible, the device configuration of fieldbusses and interfaces is kept and switched over to the device configuration of the new module.

Target change options:

- between platforms: from V2 platform to V3 platform (and vice versa)
- between module types: from AC500 (standard) to AC500-eCo (and vice versa)
- a combination of changed platform and changed module type

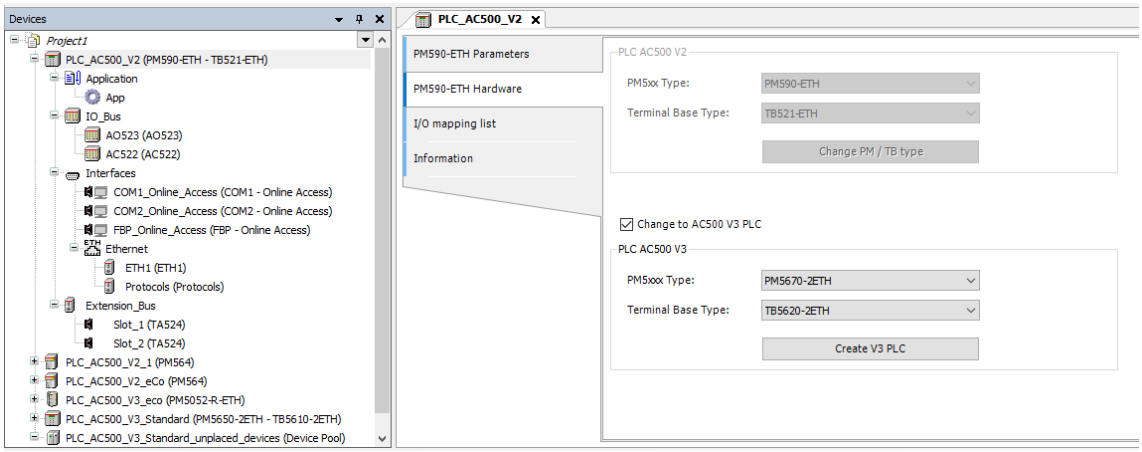
Target change from a V2 processor module to a V3 processor module

Target change options:

- AC500 V2 processor module ➜ AC500 V3 processor module
- AC500 V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500 V3 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node and open the *“PM5<...> Hardware”* tab.
3. Enable *“Change to AC500 V3 PLC”* and select the desired V3 processor module from the *“PM5xx Type”* drop-down list.



4. Click *[Create V3 PLC]*.
   ⇨ The new V3 processor module is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

> *In case of a target change from AC500-eCo V2 to AC500-eCo V3, the I/O bus and Ethernet configuration is kept.*

Target change from a V3 processor module to another V3 processor module

Target change options:

- AC500 V3 processor module ➜ AC500 V3 processor module
- AC500 V3 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V3 processor module ➜ AC500 V3 processor module
- AC500-eCo V3 processor module ➜ AC500-eCo V3 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V3 <...>* node and open the *"PM5<...> Hardware"* tab.
3. Select the desired V3 processor module from the *"PM5xx Type"* drop-down list.

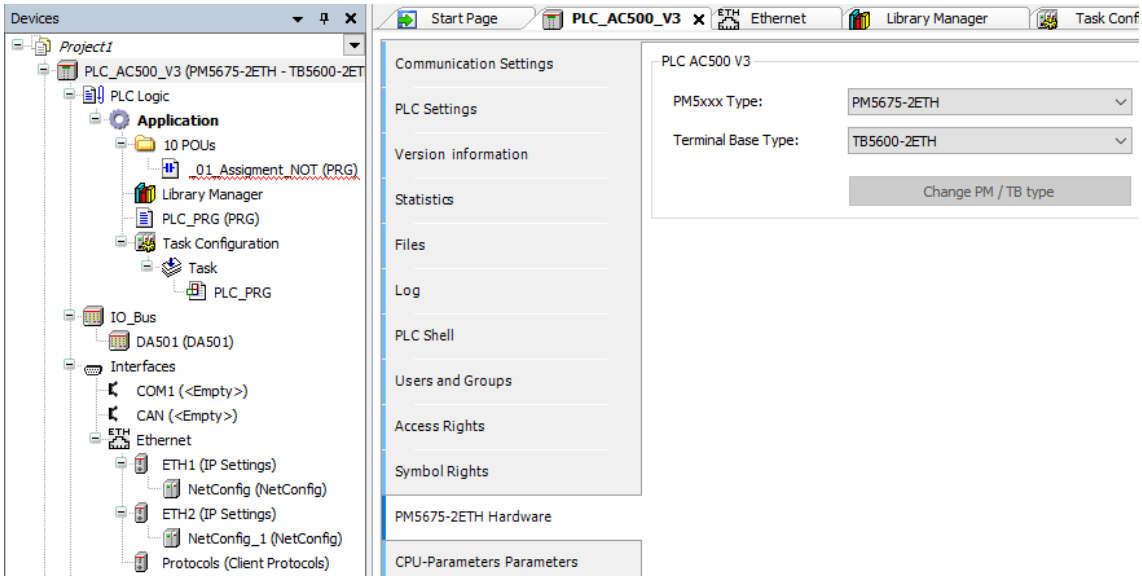
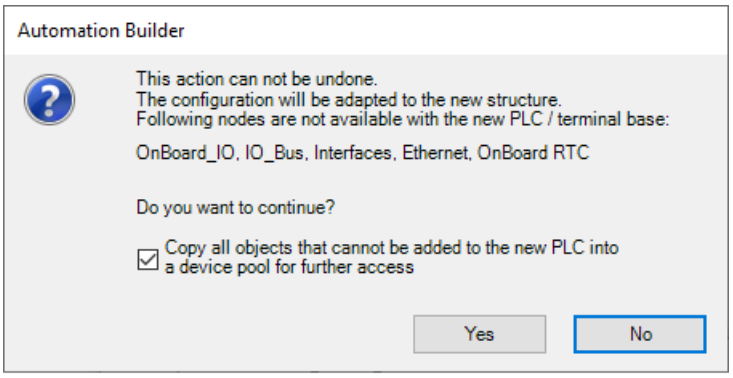


Fig. 310: Change_Hardware_V3

4. Ensure the correct *"Terminal Base Type"* is selected and click *[Change PM / TB type]*.

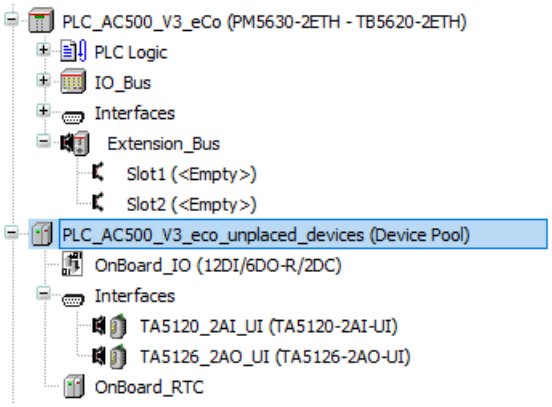
   ⇨ If possible, the device configurations from the previous processor module will be kept and switched over to the new processor module.

   The device configurations that cannot be kept are listed in a prompted information dialog.



   By default, all device configurations which cannot be switched over will be copied to a "device pool" section in the navigation tree (option "Copy all objects that cannot be added to the new PLC into a device pool for further access"). If required, this backuped configuration can be used in another project or in another processor module configuration.

   If the checkbox is deactivated all device configurations that cannot be switched will be lost after the execution of the target change.



Target change from AC500-eCo V3 to AC500 V3

The configuration of the onboard I/Os, the option board slots and the onboard RTC cannot be changed-over to the new module.

Target change from AC500 V3 to AC500-eCo V3

The configuration of COM1, CAN and the I/O bus cannot be changed-over to the new module. Depending on the selected target, also the I/O bus configuration and ETH2 configuration cannot be switched.

ETH1 configuration is kept even if the configured protocols are not allowed for the selected AC500-eCo V3 PLC. In this case error messages are displayed in the messages window.

Libraries which are not used anymore are not deleted with the target change. Libraries of option boards are kept in the Library Manager even if no longer available at the target module.

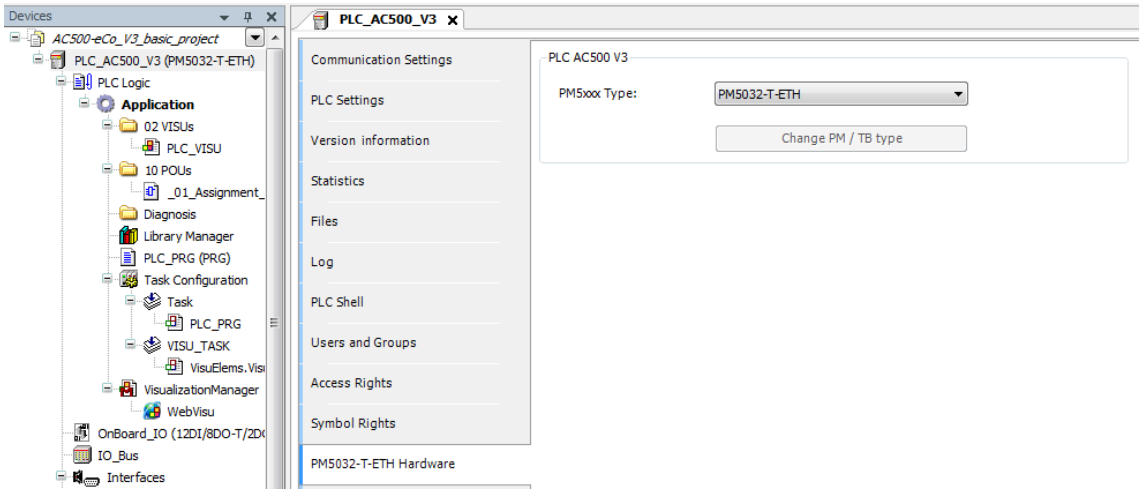
Changing the processor module type for AC500-eCo V3 CPU

It is not possible to change from an AC500 V3 processor module to an AC500-eCo V3 processor module!

Changing an AC500-eCo V3 processor module to another AC500-eCo V3 processor module is possible and the same limitation as listed before are applying, only the available or possible feature from the new processor module will be kept from the old processor module.

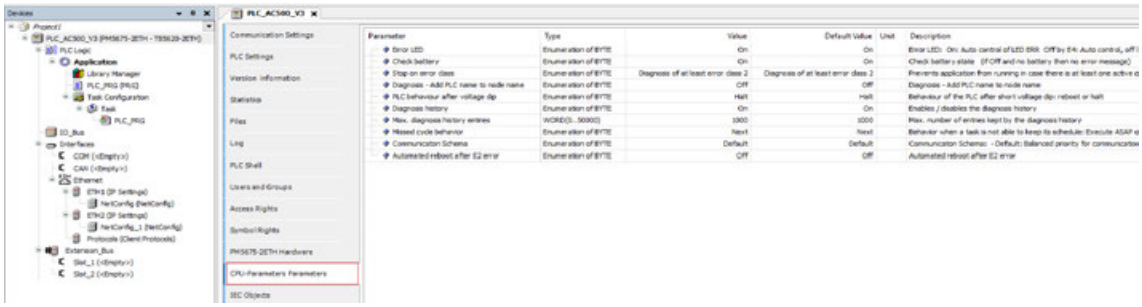
☑ Close CODESYS.

1. Double-click the *PLC_AC500_V3 <...>* node.



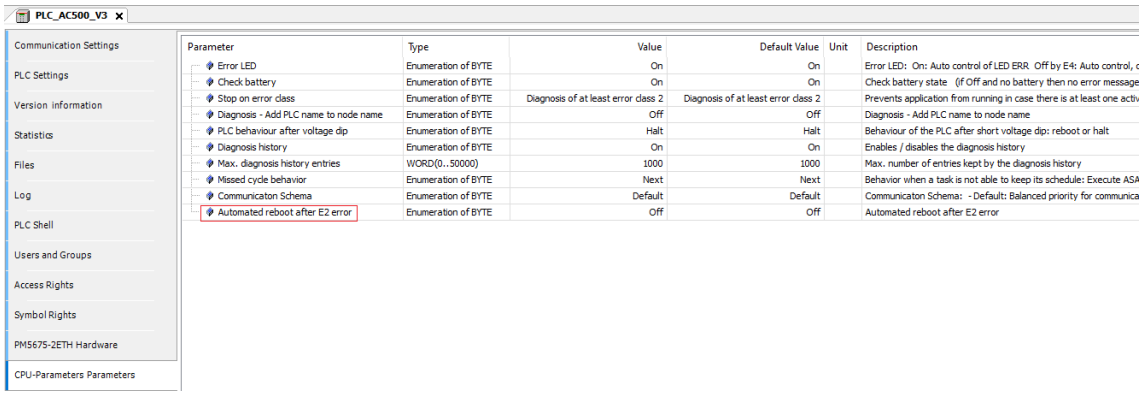
2. Open the *“PM50xx Hardware”* tab and select the new *“PM50xx Type”* from the drop-down list.

Parameters of the processor module



Automated reboot after E2 error

The parameter *“Automated reboot after E2 error”* allows to set the behavior of the CPU in case of severe errors (class E2).

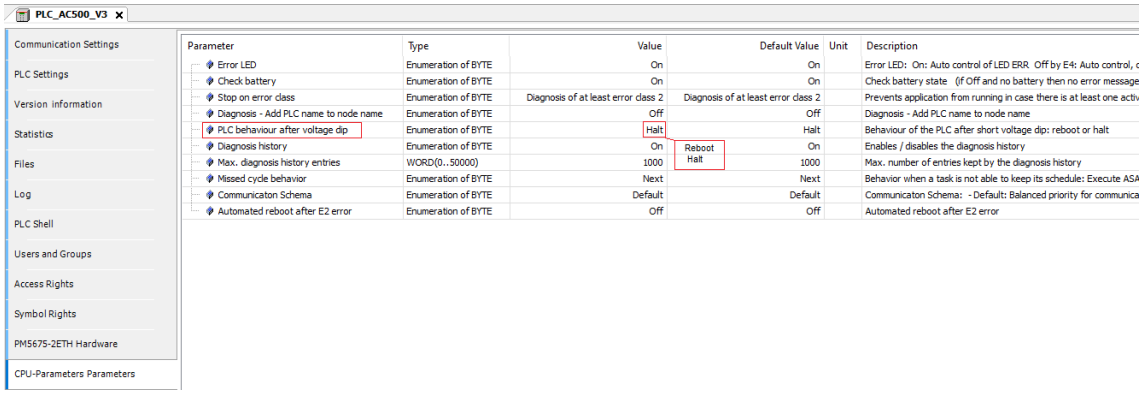


If the default setting *“Off”* is used, no automated reebot after an E2 error is performed.

If the setting *“On”* is used, an automated reebot after an E2 error is performed.

PLC behaviour after voltage dip

The parameter *“PLC behaviour after voltage dip”* allows to set the behavior of the CPU in case of short voltage dips.



If the default setting *“Halt”* is used, the CPU is changed to STOP mode if a short voltage dip >10 ms occurs. A new powercycle is required.

If the setting *“Reboot”* is used, CPU will reboot after power supply has recovered to nominal value.

Floating point values

A calculation with floating points can lead to the following values:

0 (zero)

If a calculation results in an underrun, the value is set to 0 (result near 0, but not presentable). Depending on the sign bit, it can be a positive zero or a negative zero. The operator "=" of -0 and 0 returns TRUE.

Infinity

If a calculation results in an overrun, the value is set to Infinity (the result is not presentable). Depending on the sign bit, it can be a positive infinity (Infinity) or negative infinity (-Infinity).

If Infinity is converted into another data type it results in the maximum value of the other data type (e.g.. conversion into DWORD with REAL_TO_DWORD: 16#FFFFFFFF, into DINT with REAL_TO_DINT: 16#7FFFFFFF).

If -Infinity is converted into another data type it results in the maximum value of the other data type (e.g.. conversion into DWORD with REAL_TO_DWORD: 16#00000000, into DINT with REAL_TO_DINT: 16#80000000).

Except for:

```
TRUE := REAL_TO_BOOL(Infinity);
'#Inf' := REAL_TO_STRING(Infinity);
'-#Inf' := REAL_TO_STRING(-Infinity);
```

Examples:

Infinity	-Infinity
Infinity := 1.0 / 0.0	-Infinity := -1.0 / 0.0
Infinity := Infinity + Infinity	-Infinity := -Infinity -Infinity
Infinity := Infinity + 1.0	-Infinity := -Infinity + 1.0
Infinity := LREAL_TO_REAL(Infinity)	-Infinity := LREAL_TO_REAL(-Infinity)

NaN

If a calculation results in an undefined value the result is set to NaN (Not a Number). The result of each calculation with NaN is NaN. The operators "<", "<=", ">" and ">=" return FALSE if either or both operands are NaN.

Operator "=" returns FALSE if one operand is NaN.

Operator "<>" returns TRUE if one operand is NaN.

If NaN is converted into another data type the result is 0.

Except for:

```
TRUE := REAL_TO_BOOL(NaN);
'#NaN' := REAL_TO_STRING(NaN);
```

Examples:

NaN := SQRT(-2.0)

NaN := 0.0 / 0.0

NaN := Infinity -Infinity

NaN := 0.0 * Infinity

NaN := Inifnity / Infinity

The result of an operation can be checked with the following program parts:

Check for NaN (REAL):	Check for NaN (LREAL):

`rX: REAL;`
`IF (rX <> rX) THEN`
(* rX is a NaN *)
`...;`
`END_IF;`
	`lrX: LREAL;`
	`IF (lrX <> lrX) THEN`
	(* lrX is a NaN *)
	`...;`
	`END_IF;`

Check for Infinity (REAL):	Check for Infinity (LREAL):
Infinity is represented with sign bit 0, exponent of all 1s and a fraction of all 0s. -Infinity is represented with sign bit 1, exponent of all 1s and a fraction of all 0s.	

rX: REAL;
prX: POINTER TO REAL;
pdwX: POINTER TO DWORD;
prX := ADR(rX);
pdwX := prX;
IF (pdwX^ = 16#7F800000) THEN
(* rX is Infinity *)
...;
END_IF;
IF (pdwX^ = 16#FF800000) THEN
(* rX is -Infinity *)
...;
END_IF;
	lrX: LREAL;
	plrX: POINTER TO LREAL;
	plwX: POINTER TO LWORD;
	plrX := ADR(lrX);
	plwX := plrX;
	IF (plwX^ = 16#7FF0000000000000) THEN
	(* lrX is Infinity *)
	...;
	END_IF;
	IF (plwX^ = 16#FFF0000000000000) THEN
	(* lrX is -Infinity *)
	...;
	END_IF;

1.6.6.2.4 AC500-eCo V3 onboard I/Os

According to the used AC500-eCo V3 processor module, the onboard I/Os are different and the functionality of the I/Os are adapted to the processor module type.

Onboard I/O combination	PM5012-x-ETH	PM5032-x-ETH	PM5052-x-ETH	PM5072-T-2ETH
6 DI, digital input 24 V DC / 4 DO, digital output transistor 24 V DC / 0.5 A	X			
6 DI, digital input 24 V DC / 4 DO, digital output relay 240 V AC / 2 A	X			
12 DI, digital input 24 V DC / 8 DO, digital output transistor 24 V DC / 0.5 A / 2 DC, digital in/out configurable 24 V DC, 24 V DC / 0.5 A		X	X	X
12 DI, digital input 24 V DC / 6 DO, digital output relay 240 V AC / 2A / 2 DC, digital in/out configurable 24 V DC, 24 V DC / 0.5 A		X	X	

1.6.6.2.5 Configure the onboard I/O channel

The onboard I/Os support the following channels functions according to the processor module type:

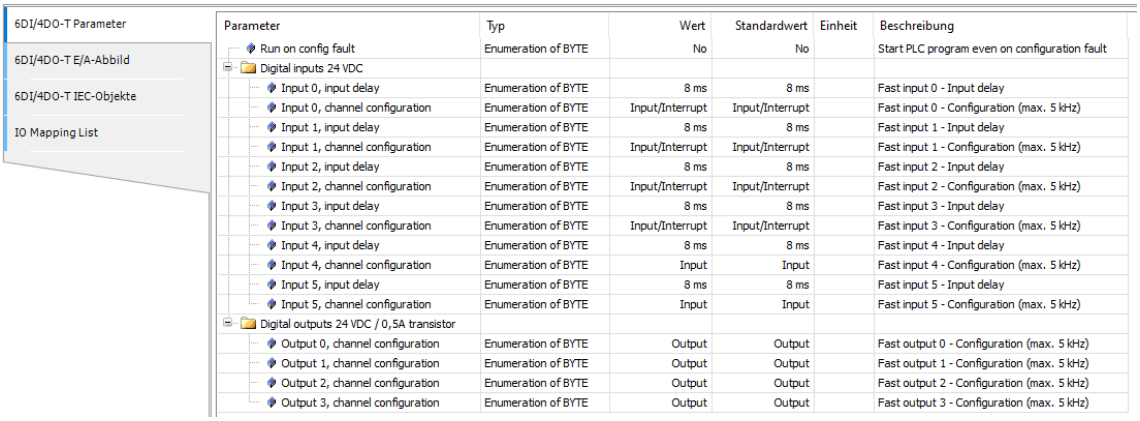
Onboard I/O type	Channel function	PM5012-T-ETH	PM5012-R-ETH	Channel name when available
Digital input channel total thereof as	Digital input	6	6	DI0 … DI5
Fast input x, max. 5 kHz	Digital input	6	6	DI0 … DI5
	Interrupt input	4	4	DI0 … DI3
	Fast counter	2	2	DI4 … DI5
Digital output channel total thereof as	Digital output	4	4	DO0 … DO3
Fast output x, max. 5 kHz	Digital output	4	4	DO0 … DO3
	Limit switch	4	-	DO0 … DO3
	PWM output	4	-	DO0 … DO3

Onboard I/O type	Channel function	PM5032-T-ETH PM5052-T-ETH PM5072-T-2ETH	PM5032-R-ETH PM5052-R-ETH	Channel name when available
Digital input channel total thereof as	Digital input	12	12	DI0 … DI11
Fast input x, max. 5 kHz	Digital input	4	4	DI0 … DI3
	Interrupt input	4	4	DI0 … DI3
	Touch/Reset	4, together with dedicated encoder	4, together with dedicated encoder	DI0 … DI3
Fast input x, max. 100 kHz	Digital input	4	4	DI4 … DI7
	Interrupt input	2, with A/B tracks	2, with A/B tracks	DI4 … DI7
	Fast counter	4	4	DI4 … DI7
Standard input	Digital input	4	4	DI8 … DI11
Digital output channel total thereof as	Digital output	8	6	DO0 … DO7 DO0 … DO5
Fast output x, max. 5 kHz	Digital output	4	-	DO0 … DO3
	Limit switch	4	-	DO0 … DO3
	PWM output	4	-	DO0 … DO3
Fast output x, max. 100 kHz	Digital output	4	-	DO4 … DO7
	Limit switch	4	-	DO4 … DO7
	PWM output	4	-	DO4 … DO7
	PTO output	2, pair of output	-	DO4 … DO7
Digital in/output configurable channel total thereof as	Digital in/output	2	2	DC12 … DC13
Standard dig. channel	Digital In/output	2	2	DC12 … DC13

Onboard I/O type	Channel function	PM5032-T-ETH PM5052-T-ETH PM5072-T-2ETH	PM5032-R-ETH PM5052-R-ETH	Channel name when available
Fast output, max. 100 kHz	Limit switch	-	2	DC12 … DC13
	PWM output	-	2	DC12 … DC13
	PTO output	-	2	DC12 … DC13

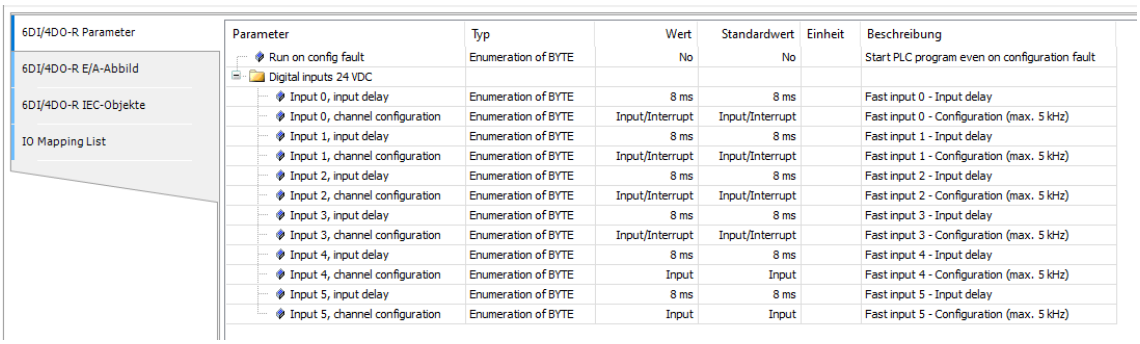
PM5012-x-ETH Basic CPU

For all CPU versions the configuration of the input channels is the same. The configuration of the output channels is only available on CPU version with transistor output channels:



Parameter	Typ	Wert	Standardwert	Einheit	Beschreibung
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Digital inputs 24 VDC					
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 0 - Input delay
Input 0, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 0 - Configuration (max. 5 kHz)
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 1 - Input delay
Input 1, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 1 - Configuration (max. 5 kHz)
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 2 - Input delay
Input 2, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 2 - Configuration (max. 5 kHz)
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 3 - Input delay
Input 3, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 3 - Configuration (max. 5 kHz)
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 4 - Input delay
Input 4, channel configuration	Enumeration of BYTE	Input	Input		Fast input 4 - Configuration (max. 5 kHz)
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 5 - Input delay
Input 5, channel configuration	Enumeration of BYTE	Input	Input		Fast input 5 - Configuration (max. 5 kHz)
Digital outputs 24 VDC / 0,5A transistor					
Output 0, channel configuration	Enumeration of BYTE	Output	Output		Fast output 0 - Configuration (max. 5 kHz)
Output 1, channel configuration	Enumeration of BYTE	Output	Output		Fast output 1 - Configuration (max. 5 kHz)
Output 2, channel configuration	Enumeration of BYTE	Output	Output		Fast output 2 - Configuration (max. 5 kHz)
Output 3, channel configuration	Enumeration of BYTE	Output	Output		Fast output 3 - Configuration (max. 5 kHz)

Version with relay outputs, same configuration for the input channels, no configuration for the output channels relay:



Parameter	Typ	Wert	Standardwert	Einheit	Beschreibung
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Digital inputs 24 VDC					
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 0 - Input delay
Input 0, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 0 - Configuration (max. 5 kHz)
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 1 - Input delay
Input 1, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 1 - Configuration (max. 5 kHz)
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 2 - Input delay
Input 2, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 2 - Configuration (max. 5 kHz)
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 3 - Input delay
Input 3, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 3 - Configuration (max. 5 kHz)
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 4 - Input delay
Input 4, channel configuration	Enumeration of BYTE	Input	Input		Fast input 4 - Configuration (max. 5 kHz)
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 5 - Input delay
Input 5, channel configuration	Enumeration of BYTE	Input	Input		Fast input 5 - Configuration (max. 5 kHz)

The following parameter can be configured:

Onboard I/O type	Parameter	Channel name	Default value	Value	Description
Digital inputs	Input X, input delay	Channel 0..5	8 ms	No delay	Configures input with no delay
				1 ms	Configures 1 ms input delay
				8 ms	Configures 8 ms input delay
				32 ms	Configures 32 ms input delay

Onboard I/O type	Parameter	Channel name	Default value	Value	Description
	Input X, channel configuration	Channel 0..3	Input/Interrupt	Input/Interrupt	Configures the channel as normal digital or interrupt input
	The configuration /function of the following channels is realized using function blocks in the program	Channel 4..5	Input	Input	Configures the channel as normal digital input
				Encoder 0 track-A or B	Configures the pair of channels as encoder input track A or B. When that value is configured then both channels are reserved for that functionality
				Forward counter	Configures the channel as forward counter
The configuration of output channel is only available on the CPU with transistor outputs					
Digital outputs	Output X, channel configuration	Channel 0..3	Output	Output	Configures the channel as digital output
				Limit switch	Configures the channel as limit switch output
				PWM	Configures the channel as PWM output

PM5032-x-ETH, PM5052-x-ETH Standard CPU

For all CPU versions the configuration of the input channels is the same. The configuration of the output channels is only available on CPU version with transistor output channels, the digital configurable In/Output channels are present on both version (transistor or relay output) but with different features configurable:

12DI/8DO-T/2DC Parameter
12DI/8DO-T/2DC E/A-Abbild
12DI/8DO-T/2DC IEC-Objekte
IO Mapping List

Parameter	Typ	Wert	Standardwert	Einheit	Beschreibung
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Digital inputs 24 VDC					
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 0 - Input delay
Input 0, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 0 - Configuration (max. 5 kHz)
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 1 - Input delay
Input 1, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 1 - Configuration (max. 5 kHz)
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 2 - Input delay
Input 2, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 2 - Configuration (max. 5 kHz)
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 3 - Input delay
Input 3, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 3 - Configuration (max. 5 kHz)
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 4 - Input delay
Input 4, channel configuration	Enumeration of BYTE	Input	Input		Fast input 4 - Configuration (max. 100 kHz)
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 5 - Input delay
Input 5, channel configuration	Enumeration of BYTE	Input	Input		Fast input 5 - Configuration (max. 100 kHz)
Input 6, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 6 - Input delay
Input 6, channel configuration	Enumeration of BYTE	Input	Input		Fast input 6 - Configuration (max. 100 kHz)
Input 7, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 7 - Input delay
Input 7, channel configuration	Enumeration of BYTE	Input	Input		Fast input 7 - Configuration (max. 100 kHz)
Input 8, input delay	Enumeration of BYTE	8 ms	8 ms		Standard input 8 - Input delay
Input 9, input delay	Enumeration of BYTE	8 ms	8 ms		Standard input 9 - Input delay
Input 10, input delay	Enumeration of BYTE	8 ms	8 ms		Standard input 10 - Input delay
Input 11, input delay	Enumeration of BYTE	8 ms	8 ms		Standard input 11 - Input delay
Digital outputs 24 VDC / 0,5A transistor					
Output 0, channel configuration	Enumeration of BYTE	Output	Output		Fast output 0 - Configuration (max. 5 kHz)
Output 1, channel configuration	Enumeration of BYTE	Output	Output		Fast output 1 - Configuration (max. 5 kHz)
Output 2, channel configuration	Enumeration of BYTE	Output	Output		Fast output 2 - Configuration (max. 5 kHz)
Output 3, channel configuration	Enumeration of BYTE	Output	Output		Fast output 3 - Configuration (max. 5 kHz)
Output 4, channel configuration	Enumeration of BYTE	Output	Output		Fast output 4 - Configuration (max. 100 kHz)
Output 5, channel configuration	Enumeration of BYTE	Output	Output		Fast output 5 - Configuration (max. 100 kHz)
Output 6, channel configuration	Enumeration of BYTE	Output	Output		Fast output 6 - Configuration (max. 100 kHz)
Output 7, channel configuration	Enumeration of BYTE	Output	Output		Fast output 7 - Configuration (max. 100 kHz)
Digital configurable In/outputs 24 VDC / 0,5A transistor					
Input / Output DC12, channel configuration	Enumeration of BYTE	Input/Output	Input/Output		Digital configurable channel 12 - Configuration
Input / Output DC13, channel configuration	Enumeration of BYTE	Input/Output	Input/Output		Digital configurable channel 13 - Configuration

For the CPU with relay outputs, the digital configurable Input/Output channels have specific functionalities:

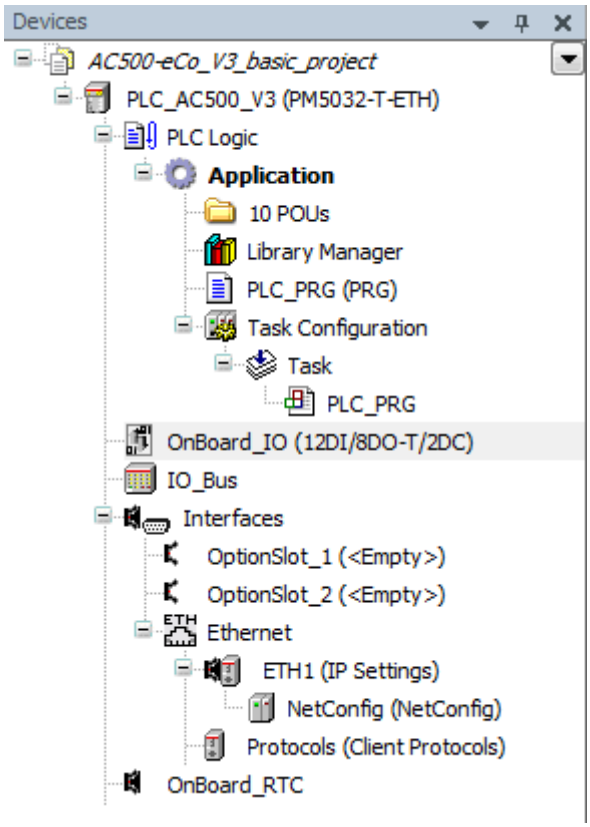
Onboard I/O type	Parameter	Channel name	Default value	Value	Description
Digital inputs	Input X, input delay	Channel 0..11	8 ms	No delay	Configures input with no delay
				1 ms	Configures 1 ms input delay
				8 ms	Configures 8 ms input delay
				32 ms	Configures 32 ms input delay
Fast inputs max. 5 kHz	Input X, channel configuration	Channel 0..1	Input/Interrupt	Input/Interrupt	Configures the channel as normal digital or interrupt input
				Touch/Reset 0	Configures the pair of adjacent channels as Touch/Reset inputs together with encoder 0
		Channel 2..3	Input/Interrupt	Input/Interrupt	Configures the channel as normal digital or interrupt input
				Touch/Reset 1	Configures the pair of adjacent channels as Touch/Reset inputs together with encoder 1
Fast inputs max. 100/200 kHz	The function of the following channels is realized using function blocks in the program	Channel 4..5	Input	Input	Configures the channel as normal digital input
Max. frequency 200 kHz	When that value is configured then both channels are reserved for that functionality			Encoder 0 track-A or B	Configures the pair of adjacent channels as encoder 0 input track A or B.
Max. frequency 100 kHz				Forward counter	Configures the channel as forward counter
Fast inputs max. 100/200 kHz		Channel 6..7	Input	Input	Configures the channel as normal digital input

Onboard I/O type	Parameter	Channel name	Default value	Value	Description
Max. frequency 200 kHz	When that value is configured then both channels are reserved for that functionality			Encoder 1 track-A or B	Configures the pair of adjacent channels as encoder 1 input track A or B.
				Touch/Reset	Configures the pair of adjacent channels as Touch/Reset inputs together with encoder 0
Max. frequency 100 kHz				Forward counter	Configures the channel as forward counter
The following configuration of output channel is only available on the CPU with transistor					
Fast outputs, max. 5 kHz	Output X, channel configuration	Channel 0..3	Output	Output	Configures the channel as digital output
	The configuration / function of the following channels is realized using function blocks in the program			Limit switch	Configures the channel as limit switch output
				PWM	Configures the channel as PWM output
Fast outputs max. 100/200 kHz	Output X, channel configuration	Channel 4..7	Output	Output	Configures the channel as digital output
	The function of the following channels is realized using function blocks in the program			Limit switch	Configures the channel as limit switch output
Max. frequency 100 kHz				PWM	Configures the channel as PWM output
Max. frequency 100/200 kHz	Depending on the OBIOMotionPTO or OBIOMotionPWM function block used			PTO	Configures the pair of adjacent channels as PTO output
Digital configurable input/outputs	Output X, channel configuration	Channel DC12..DC13	Input/Output	Input/Output	Configures the channel as digital input/output
The following configuration of output channel is only available on the CPU with relay outputs					

Onboard I/O type	Parameter	Channel name	Default value	Value	Description
Digital configurable input/outputs	Output X, channel configuration	Channel DC12..DC13	Input/Output	Input/Output	Configures the channel as digital input/output
	The function of the following channels is realized using function blocks in the program			Limit switch	Configures the channel as limit switch output
Max. frequency 100 kHz				PWM	Configures the channel as PWM output
Max. frequency 100/200 kHz	Depending on the OBIOMotionPTO or OBIOMotionPWM function block used			PTO	Configures the pair of channels as PTO output

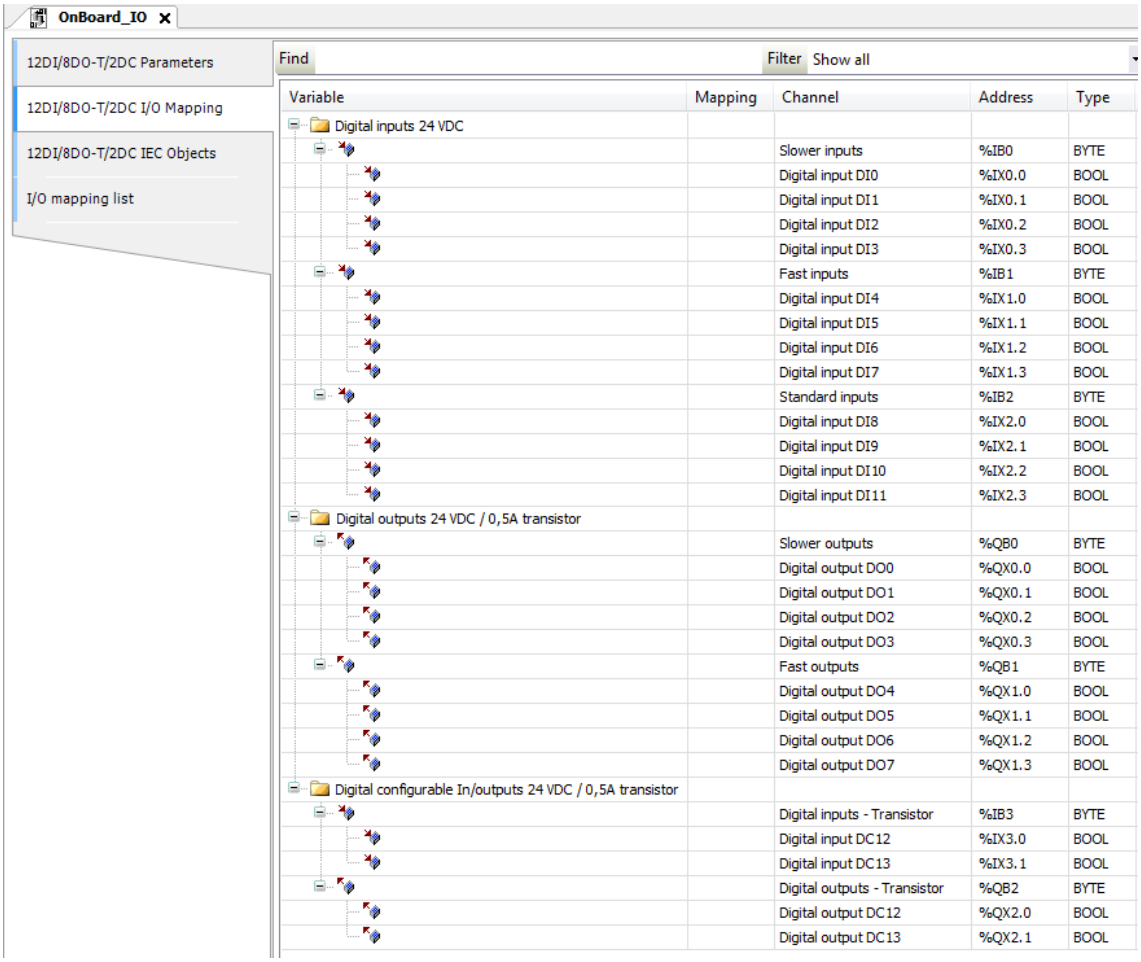
1.6.6.2.6 Mapping of the I/O channels

Onboard I/O variable mapping



1. Double-click *“OnBoard_IO”* in the device tree.

⇨ A tab opens in the editor view.

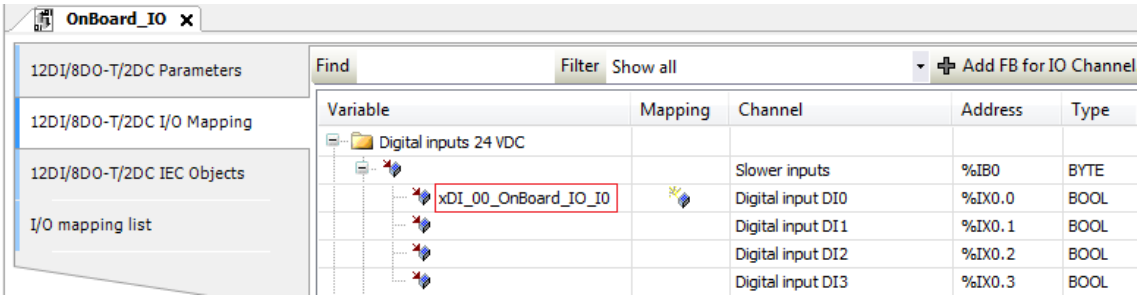


2. Select *“12DI/8DO-T/2DC I/O Mapping”*.

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

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

Handle the digital input variables

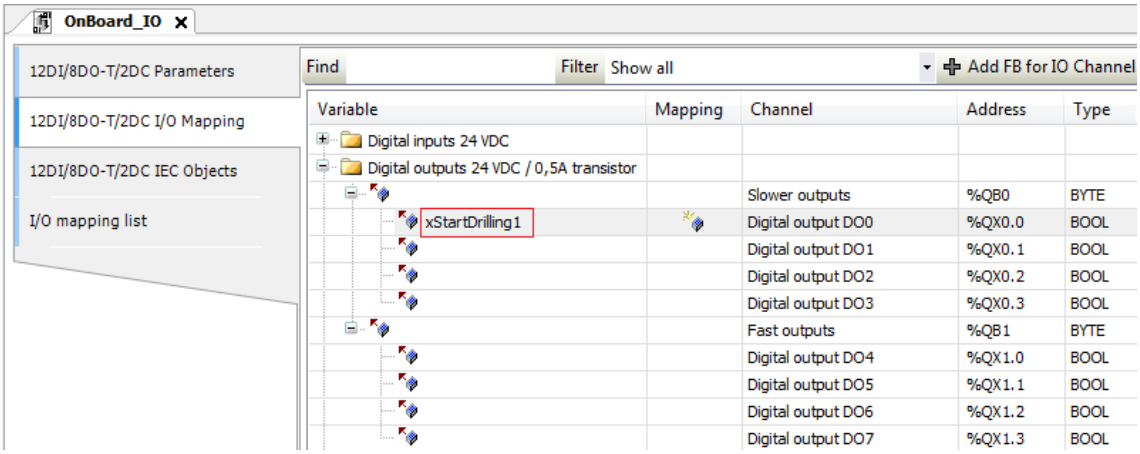


1. Open the list of the digital inputs.

2. Fill in the variable names:

Channel	Type	Variable
Digital input DI0	BOOL	xDI_00_OnBoard_IO_I0

Handle the digital output variables



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

Channel	Type	Variable
Digital output DO0	BOOL	xStartDrilling1

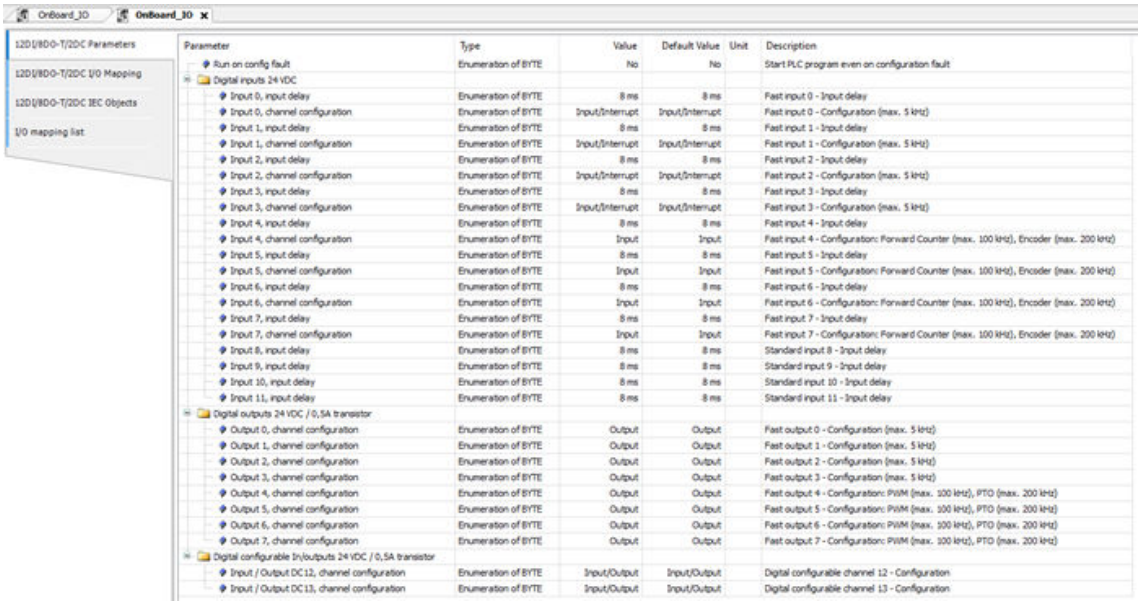
1.6.6.2.7 Configuration of the onboard I/Os of AC500-eCo V3 PLC

Digital inputs from the onboard I/Os

Depending on the processor module used, several configurations are possible for the onboard I/Os mostly different per group of channels.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
Digital inputs	PM5012-x-ETH	I0…I3	Input/Interrupt	Not needed	Input delay can be sometimes configured according to channels type
		I4…I5	Input		
	PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH	I0…I3	Input/Interrupt	Not needed	Input delay can be sometimes configured according to channels type, for the PLC with relay outputs,
		I4…I7	Input		

Functionality to be realized	Processor module type	Digital input channels	Configura-tion of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
		I8...I11	- (Always inputs)		the digital configurable channels have some other configu-rable features.
		C12…C13	Input/Output		



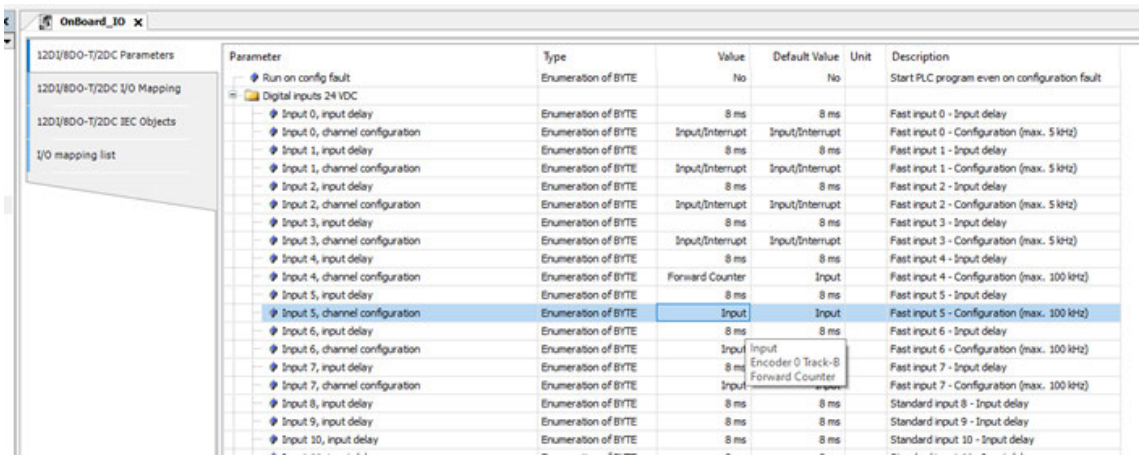
Fast counters in the onboard I/Os

General details on fast counters see (System Technology) ↳ *Chapter 1.6.5.1.13.2 “Fast counter in AC500-eCo V3 (Onboard I/O in PM50xx)” on page 3576*

Details on the configuration see ↳ *Chapter 1.6.6.2.5 “Configure the onboard I/O channel” on page 3700*

Depending on the configuration for the input channels of the onboard I/O from the processor module different functionality are possible which must be used together with the dedicated function block of the user program.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
Forward counter	PM5012-x-ETH	I0…I3	Not relevant for the functionality	OBIOForwardCounter	Up to 2 forward counters with up to 5 kHz can be used, the other inputs can be used for other purpose
		I4...I5	Forward counter		
	PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH	I0…I3	Not relevant for the functionality	OBIOForwardCounter	Up to 4 forward counters with up to 100 kHz can be used, the other inputs can be used for other purpose
		I4…I7	Forward counter		
		I8...I11	Not relevant for the functionality		



A/B Encoder in the onboard I/Os

Depending on the configuration for the input channels of the onboard I/O from the processor module different functionality are possible which must be used together with the dedicated function block of the user program.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
A/B encoder 5 kHz with touch/reset inputs	PM5012-x-ETH	I0	Touch/Reset0	OBIOEncoderCounter	The functionality uses the 4 digital inputs, the other can be used for other purpose
		I1	Touch/Reset0		
		I2…I3	Not relevant for the functionality		
		I4	Encoder 0 Track A		
		I5	Encoder 0 Track B		
Up to 2x A/B encoders 200kHz possible with touch/reset standard inputs	PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH	I0	Touch/Reset0	OBIOEncoderCounter	The functionality uses up to the 4 digital fast inputs 200 kHz and the 4x 5 kHz, the other inputs can be used for other purpose. Select encoder x track A for an input (I4 or I7) automatically selects the adjacent input for B track
		I1	Touch/Reset0		
		I2	Touch/Reset1		
		I3	Touch/Reset1		
		I4	Encoder 0 Track A		
		I5	Encoder 0 Track B		
		I6	Encoder 1 Track A		
		I7	Encoder 1 Track B		
		I8...I11	Not relevant for the functionality		
One A/B encoder 200 kHz with touch/reset	PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH	I0	Not relevant for the functionality	OBIOEncoderCounter	The functionality uses the 4 digital fast inputs 200 kHz, the other inputs can be used for other purpose. Select encoder x track A for the input (I4) automatically selects the adjacent input for B track
		I1	Not relevant for the functionality		
		I2	Not relevant for the functionality		
		I3	Not relevant for the functionality		
		I4	Encoder 0 Track A		
		I5	Encoder 0 Track B		
		I6	Touch/Reset		
		I7	Touch/Reset		
		I8...I11	Not relevant for the functionality		

Example with one encoder

Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Digital inputs 24 VDC					
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 0 - Input delay
Input 0, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 0 - Configuration (max. 5 kHz)
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 1 - Input delay
Input 1, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 1 - Configuration (max. 5 kHz)
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 2 - Input delay
Input 2, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 2 - Configuration (max. 5 kHz)
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 3 - Input delay
Input 3, channel configuration	Enumeration of BYTE	Input/Interrupt	Input/Interrupt		Fast input 3 - Configuration (max. 5 kHz)
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 4 - Input delay
Input 4, channel configuration	Enumeration of BYTE	Encoder 0 Track-A	Input		Fast input 4 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 5 - Input delay
Input 5, channel configuration	Enumeration of BYTE	Encoder 0 Track-B	Input		Fast input 5 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)
Input 6, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 6 - Input delay
Input 6, channel configuration	Enumeration of BYTE	Reset 0	Input		Fast input 6 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)
Input 7, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 7 - Input delay
Input 7, channel configuration	Enumeration of BYTE	Touch 0	Input		Fast input 7 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)

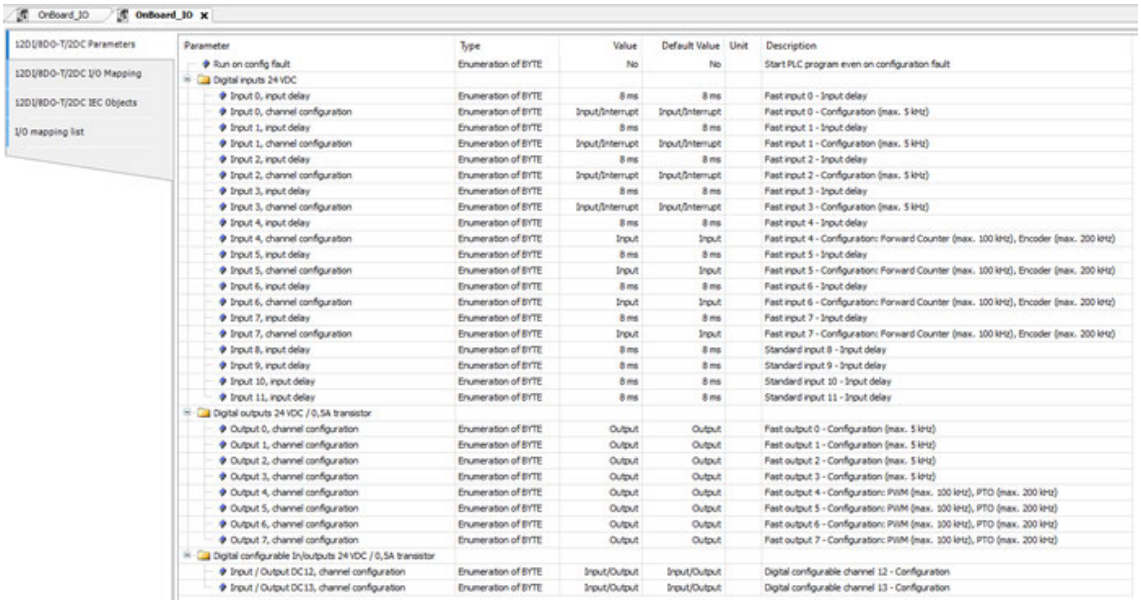
Example with two encoders

Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Digital inputs 24 VDC					
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 0 - Input delay
Input 0, channel configuration	Enumeration of BYTE	Reset 0	Input/Interrupt		Fast input 0 - Configuration (max. 5 kHz)
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 1 - Input delay
Input 1, channel configuration	Enumeration of BYTE	Touch 0	Input/Interrupt		Fast input 1 - Configuration (max. 5 kHz)
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 2 - Input delay
Input 2, channel configuration	Enumeration of BYTE	Reset 1	Input/Interrupt		Fast input 2 - Configuration (max. 5 kHz)
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 3 - Input delay
Input 3, channel configuration	Enumeration of BYTE	Touch 1	Input/Interrupt		Fast input 3 - Configuration (max. 5 kHz)
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 4 - Input delay
Input 4, channel configuration	Enumeration of BYTE	Encoder 0 Track-A	Input		Fast input 4 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 5 - Input delay
Input 5, channel configuration	Enumeration of BYTE	Encoder 0 Track-B	Input		Fast input 5 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)
Input 6, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 6 - Input delay
Input 6, channel configuration	Enumeration of BYTE	Encoder 1 Track-A	Input		Fast input 6 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)
Input 7, input delay	Enumeration of BYTE	8 ms	8 ms		Fast input 7 - Input delay
Input 7, channel configuration	Enumeration of BYTE	Encoder 1 Track-B	Input		Fast input 7 - Configuration: Forward Counter (max. 100 kHz), Encoder (max. 200 kHz)

Configuration of interrupt inputs

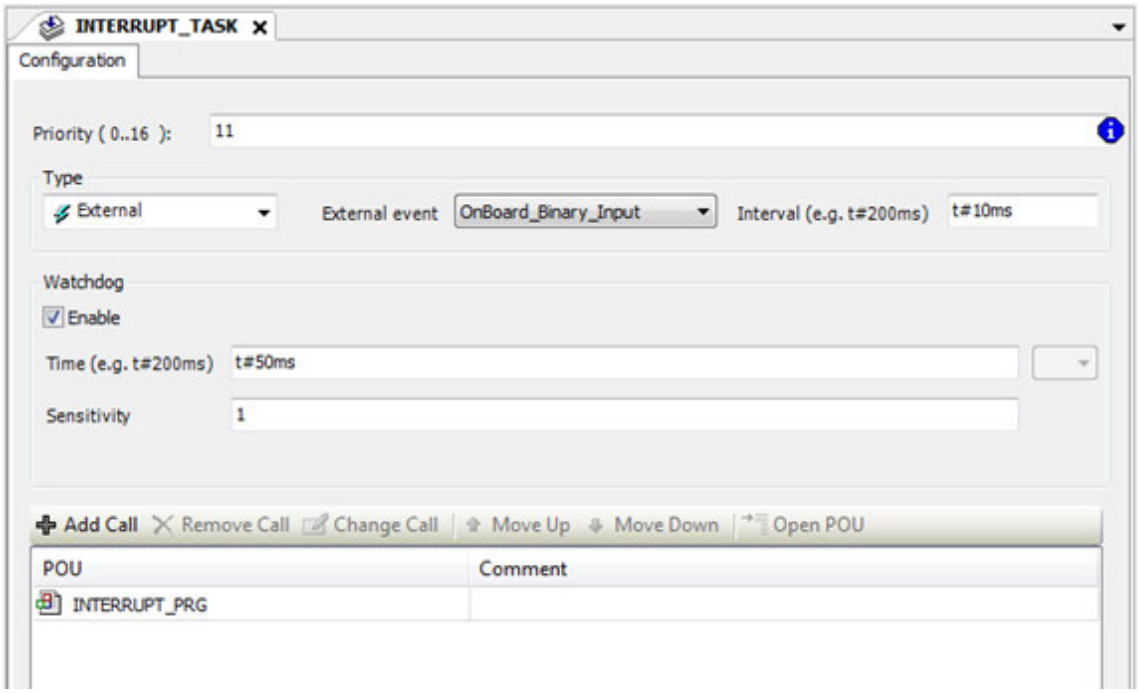
Depending on the configuration for the input channels of the onboard I/O from the processor module different functionality are possible which must be used together with the dedicated function block of the user program.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
Interrupt inputs	PM5012-x-ETH	I0…I3	Input/Interrupt	OBIO InterruptInfo OBIOInterruptPara	Up to 4 interrupt input channels can be used, the other inputs can be used for other purpose
		I4...I5	Not relevant for the functionality		
	PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH	I0…I3	Input/Interrupt	OBIO InterruptInfo OBIOInterruptPara	Up to 4 interrupt input channels can be used, the other inputs can be used for other purpose
		I4…I11	Not relevant for the functionality		



Creating an interrupt task

After configuring the parameter, the user needs to create a new task with the *“Type”* set to *“External”* and the *“External event”* set to *“OnBoard_Binary_Input”*.

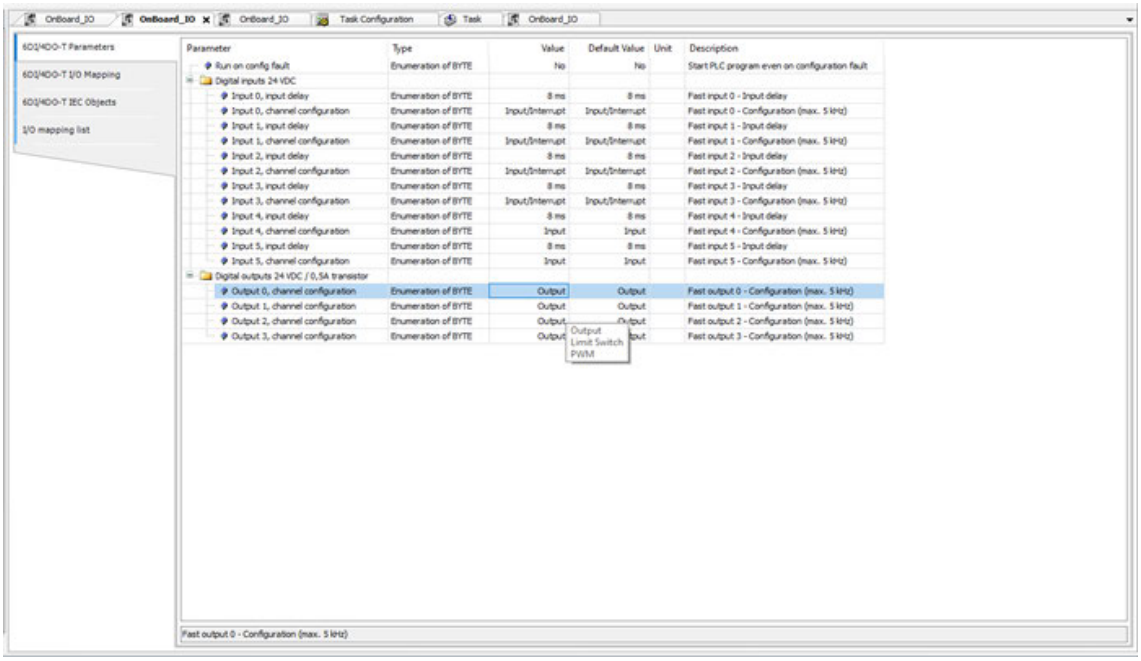


Please see the chapter how to use the function in the system technology…

Configuration of digital outputs

According to the processor module type, the digital outputs have several functionalities. To use them as digital output and as default configuration the following configuration is needed:

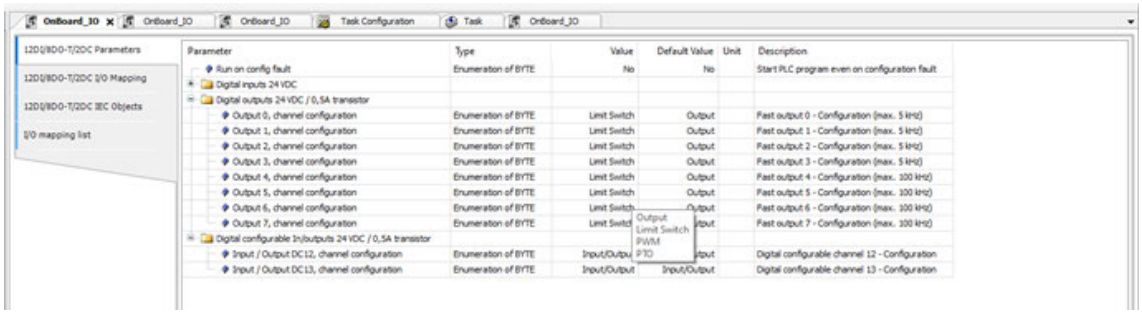
Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
Digital Outputs	PM5012-x-ETH	O0…O3	Output	Not needed	-
	PM5032-x-ETH, PM5052-x-ETH, PM5072-T-2ETH	O0…O3	Output	Not needed	No other configuration needed
		O4…O7	Output		
		C12…C13	Input/Output		



Configuration of outputs as limit switch

The AC500-eCo V33 processor modules provide according to the output variants transistor or relay some output which can be used as limit switch. For the process modules with relay outputs, only the digital configurable channels provide this functionality.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
Limit switch	PM5012-x-ETH	O0…O3	Limit Switch	OBIO-LimitSwitch	Up to 4 limit switches
	PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH	O0…O3	Limit Switch	OBIO-LimitSwitch	Up to 8 limit switches
		O4…O7	Limit Switch		
		C12…C13	Not relevant for the functionality		
	PM5032-R-ETH, PM5052-R-ETH	O0…O2	Not relevant for the functionality	OBIO-LimitSwitch	Relay outputs without other functions
		O3…O5	Not relevant for the functionality		
		C12…C13	Limit Switch		Up to 8 limit switches



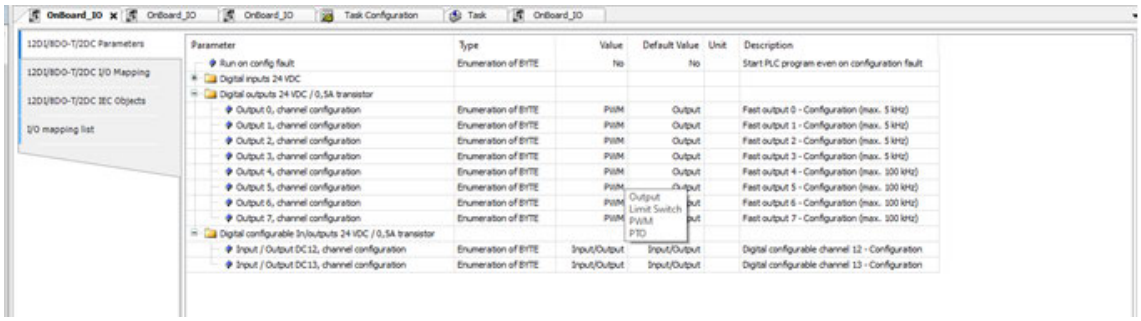
Operating the limit switch output with user program

The OBIOLimitSwitch function block of the library must be used to operate the outputs with help of user program.

Configuration of PWM outputs (Pulse Width Modulation)

The AC500-eCo V3 processor modules provide up to 8 PWM output channels with a maximum frequency of 20 KHz. The parameter of PWM output channel of onboard I/O must be configured before it can be used. User should take these steps to configure the PWM output function.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
PWM outputs	PM5012-x-ETH	O0…O3	PWM	OBIOPwm	Up to 4 PWM 100 Hz
	PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH	O0…O3	PWM	OBIOPwm	Up to 4 PWM with 100 Hz and 4 PWM 30 kHz
		O4…O7	PWM		
		C12…C13	Not relevant for the functionality		
	PM5032-R-ETH, PM5052-R-ETH	O0…O2	Not relevant for the functionality	OBIOPwm	Relay outputs without other functions
		O3…O5	Not relevant for the functionality		
		C12…C13	PWM		Up to 2 PWM 30 kHz only on these channels



Operating the PWM output with user program

The OBIOPwm function block of the library must be used to operate the PWM outputs with help of user program.

Configuration of PTO outputs (HW fast outputs for Pulse Train Output)

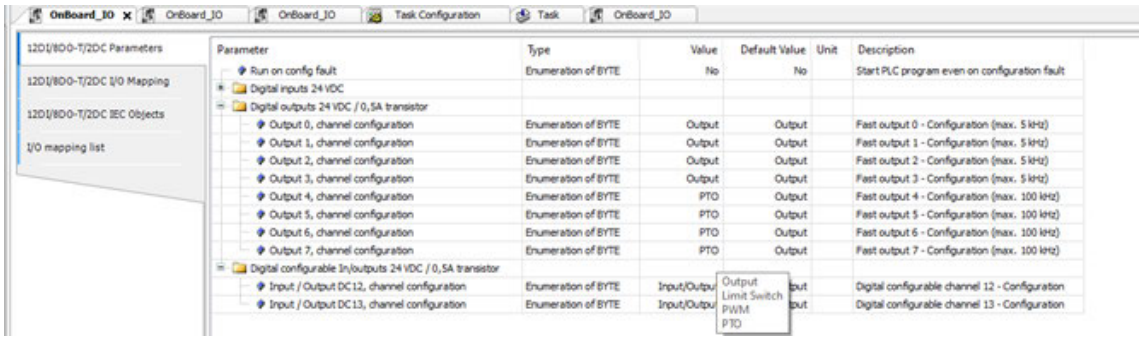
TheAC500-eCo V3 processor modules provide up to 2 PTO hardware dedicated output channels with a maximum frequency of 200 kHz. The parameter of PTO output channel of onboard I/O must be configured before it can be used. User should take these steps to configure the PTO output function.

The PTO outputs can be used with 2 different modes either Pulse / Direction or Cc/Ccw mode.

Please refer to the chapter…

The PTO channels are always requiring 2 consecutive output channels for the function.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
PTO outputs	PM5012-x-ETH	O0…O3	Not possible	-	No PTO available
	PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH	O0…O3	Not relevant for the functionality	OBIOPulseTrainOutput OBIOMotionPTO	Up to 2 PTO 200 Hz with Pulse/Direction or Cc/Ccw mode
		O4…O7	PTO -> automatically for O5 also		
		C12…C13	PTO -> automatically for O7 also		
	PM5032-R-ETH, PM5052-R-ETH	O0…O2	Not relevant for the functionality	OBIOPulseTrainOutput OBIOMotionPTO	Up to 2 PTO 200 Hz with Pulse/Direction or Cc/Ccw mode
		O3…O5	Not relevant for the functionality		
		C12…C13	PTO -> automatically for C13 also		



Operating the PTO hardware output with user program

The OBIOPulseTrainOutput function block of the library can be used to operate the PTO outputs with help of user program.This FB allows to control the output in PTO mode. The OBIOMotionPTO function block is a dedicated Motion control block to realize point-to-point movement or velocity control of a motion axis. See the dedicated chapter of the system Info…

Configuration of SW PTO (PWM) outputs (HW fast outputs and standard outputs with software dedicated function block)

The AC500-eCo V3 processor modules could also provide up to 4 PTO (PWM) software output channels with a maximum frequency of 100 kHz. To use that mode, the parameter of output channel of Onboard I/O must be configured before it can be used. User should take these steps to configure this special PTO output function. The PTO outputs channels can be only used as Pulse / Direction mode. Please refer to the chapter…

The PTO channel is using a digital fast output configured as PWM output to generate the Pulse output and a standard digital output to indicate the direction. A dedicated PTO motion block will then control the channel to realize the functionality.

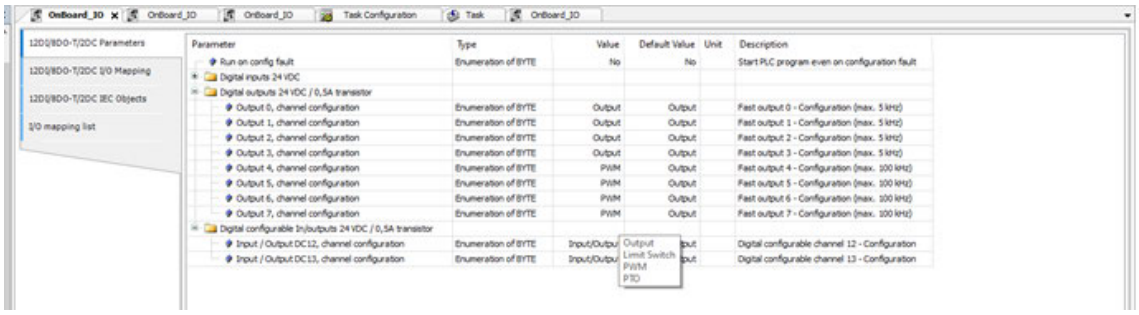
Up to 4 PTO can be then provided each using 2 digital outputs.

A mixed configuration of one HW PTO channel (e.g. Output 04..05) up to 200 kHz and Pulse/Direction or Cc/Ccw mode together with up to 2 other software PTO Channels (e.g. O6, O7 + dedicated output) up to 100 kHz and only Pulse/Direction mode is then possible.

To achieve such a software PTO mode the following channel configuration must be done.

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
PTO outputs (HW fast outputs PWM and software PTO)	PM5012-x-ETH	O0…O3	Not possible	-	No PTO available
	PM5032-T-ETH, PM5052-T-ETH, PM5072-T-2ETH	O0	Output	OBIOMotionPWM	The 4 software PTO channels will use the fast outputs O4…O7 PWM to generate the Pulse signal of each SW PTO and the
		O1	Output		
		O2	Output		
		O3	Output		
		O4	PWM		
		O5	PWM		
		O6	PWM		

Functionality to be realized	Processor module type	Digital input channels	Configuration of the channel to be selected in Automation Builder	Dedicated function block to be used in the user program	Comments
		O7	PWM		outputs O0…O3 will generate the direction signals
		C12…C13	Not relevant for the functionality		
	PM5032-R-ETH, PM5052-R-ETH	O0…O2	Not relevant for the functionality	OBIOMotionPWM	Relay outputs without other functions
		O3…O5	Not relevant for the functionality		
		C12…C13	PWM		Theoretically possible up to 2 software PTO 100 Hz with Pulse/Direction but need an additional digital ouput module for the direction signal



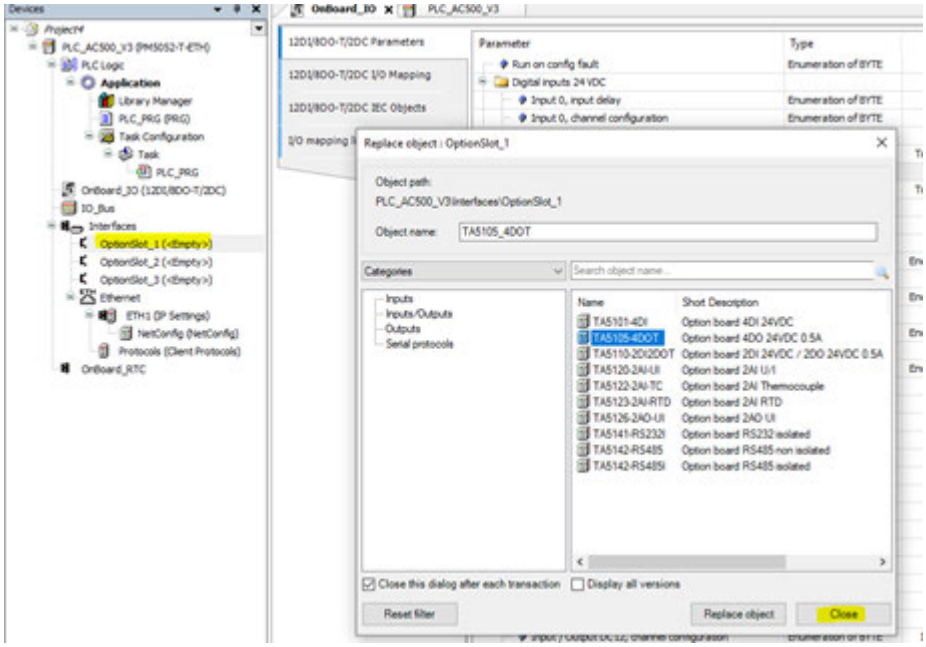
Operating the software PTO output channels with user program

The OBIOMotionPWM function block is a dedicated motion control block to realize point-to-point movement or velocity control of a motion axis. This block will then control the output channels as PTO mode Pulse/Direction only up to 100 kHz. See the dedicated chapter of the system Info…

1.6.6.2.8 Option board for processor modules PM50xx

Depending on processor module type, up to 3 option board slot are available and for each several option board modules are available.

Select the option board

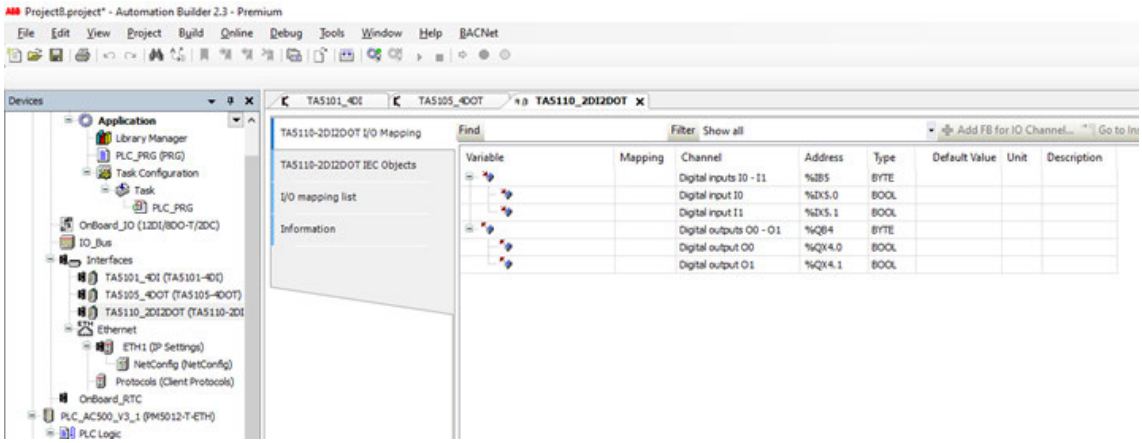


To add an option board on the processor module, select the desired OptionSlot to be configured and attach the needed option board from the list.

There is no limitation on the number of same option board used on the CPU, no dedicated slot for a specific function and no specific order to place the option board module.

Depending on the type of option board selected and attached to the CPU some further configuration of channels or function may be needed.

Attach an option board for digital I/O extension



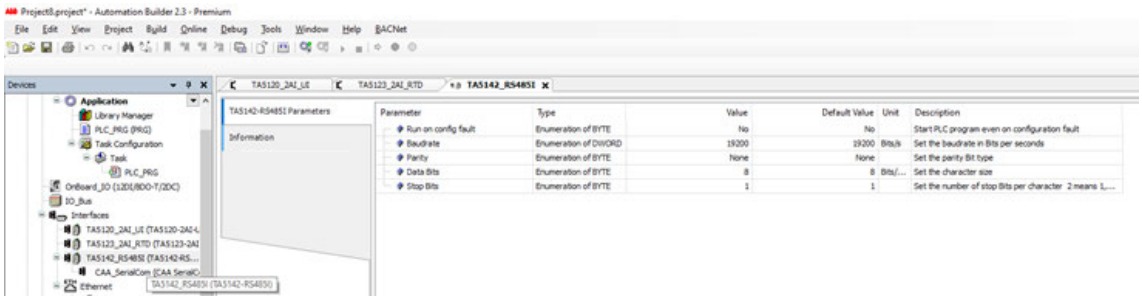
Just select and attached the module, no other channel configuration is needed. The I/O channels are directly mapped in the I/O mapping and variables can then be defined.

Attach an option board for COMx serial communication

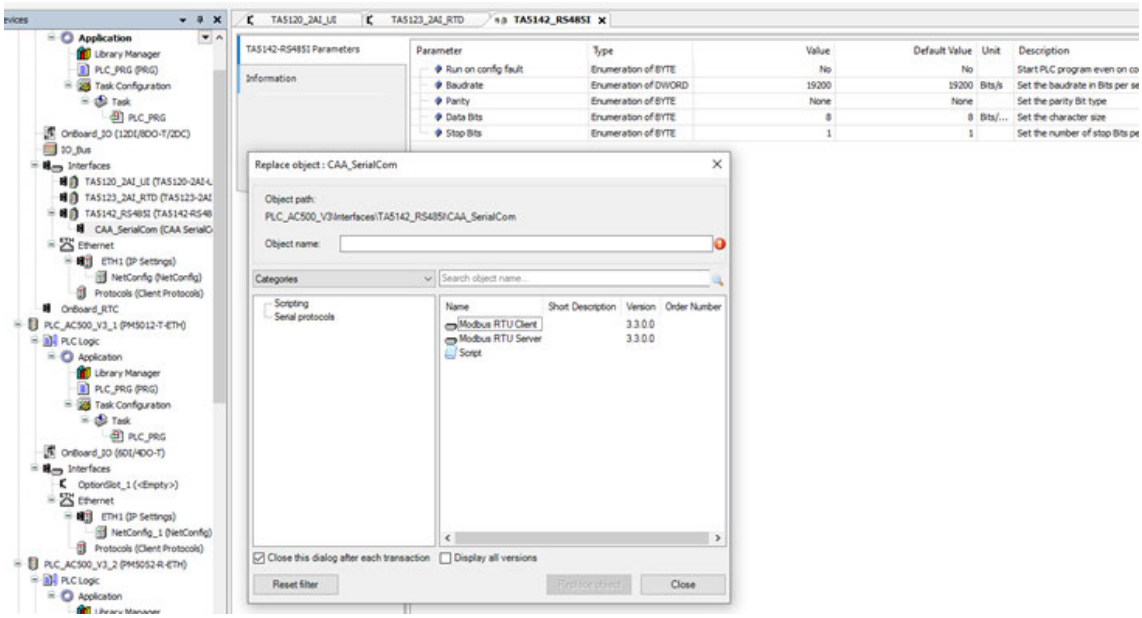
The desired serial interface option board type for the desired option board slot must be selected and added.

The option board may require some other channel configuration according to your need.

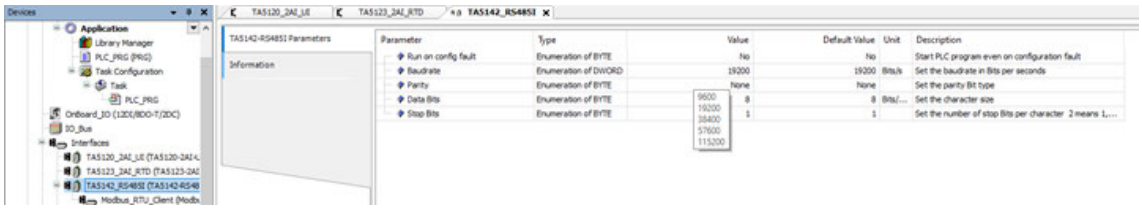
Following example shows how to add a TA5142-RS485I isolated interface and the desired protocol.



The desired protocol has also to be added according to your needs, e.g. Modbus RTU client:



The parameter for the serial interface can also be adapted like baudrate, data bit, stop bit or parity.



1.6.6.2.9 Onboard Ethernet configuration

Onboard Ethernet is provided for device types with -ETH extension.

Configuration of the IP settings with the IP configuration tool

The IP address for AC500 devices can be set or changed in Automation Builder using

- the IP configuration tool which is described in the following.
- the 'Communication Settings'. ↳ *Chapter 1.6.6.2.2.4.3 “Configuration of communication via Ethernet (TCP/IP)” on page 3688*

As an alternative the IP address can be changed at the hardware device itself. ↳ *Chapter 1.6.5.1.6.5 “Description of the function keys” on page 3491*

The IP configuration tool:

The IP configuration tool can be used

- to set or change the IP address of devices.
  ↳ *Chapter 1.6.6.2.9.1.2.2 “Changing the IP address” on page 3727*
- to scan the network for available hardware devices.
  ↳ *Chapter 1.6.6.2.9.1.2.1 “Network scan” on page 3725*
- to update the firmware of devices.
  This functionality is only supported if the IP configuration tool is used stand-alone.
  ↳ *Chapter 1.6.6.2.9.1.2.3 “Firmware update” on page 3728*
- to activate certain functionality on hardware devices.
  This feature is only available on AC500 V3 devices.
  ↳ *Chapter 1.6.6.2.9.1.2.4 “Blink functionality” on page 3732*

The IP configuration tool is part of Automation Builder and can be called via *“Tools ➔ IP-Configuration”*.

Further the IP configuration tool can be used stand-alone without an Automation Builder application running. The stand-alone variant requires a separate installation via the Installation Manager ↳ *Chapter 1.6.6.2.9.1.1 “Stand-alone installation” on page 3723*.

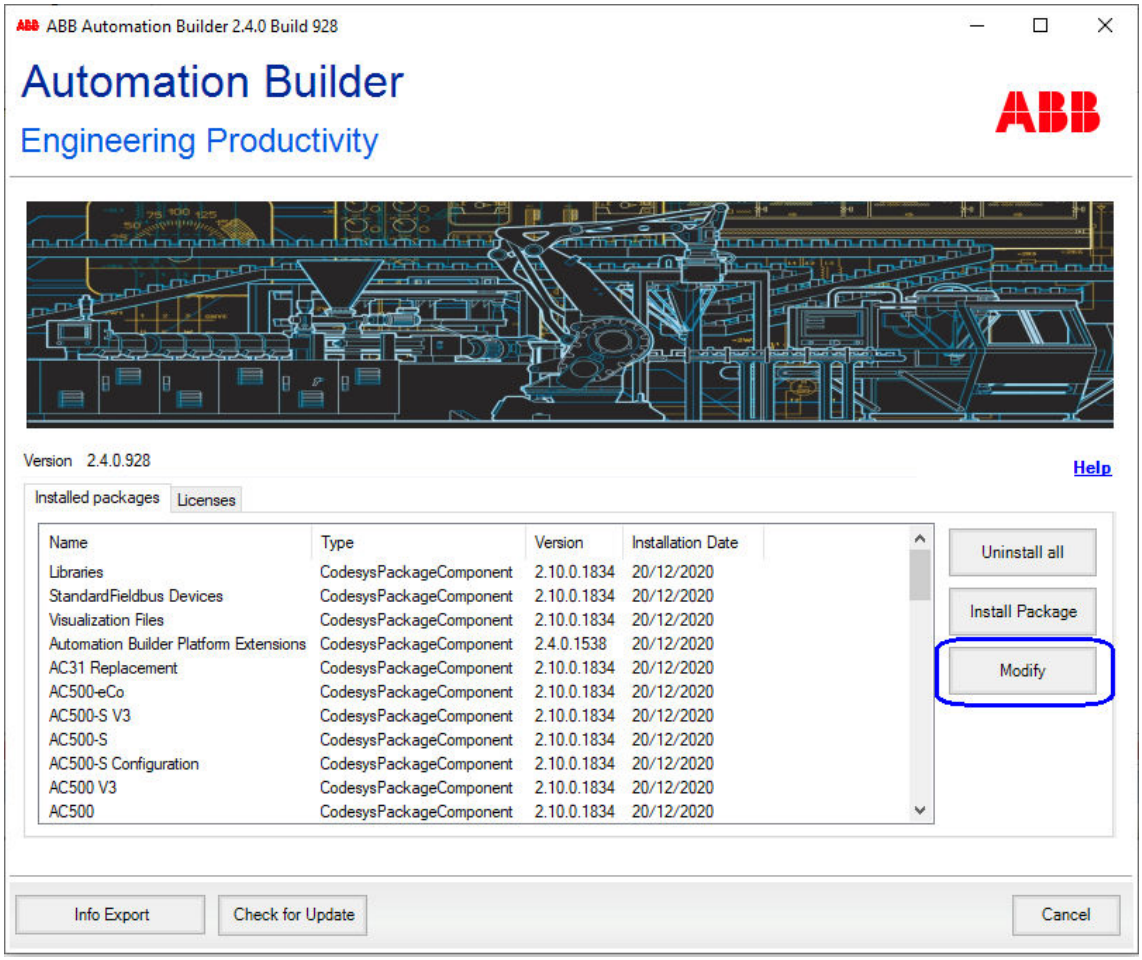
After the installation, the IP configuration tool is started via .exe file / desktop icon.

Some functionality is only supported if the IP configuration tool is used stand-alone, e.g. for firmware updates for communication interface devices.

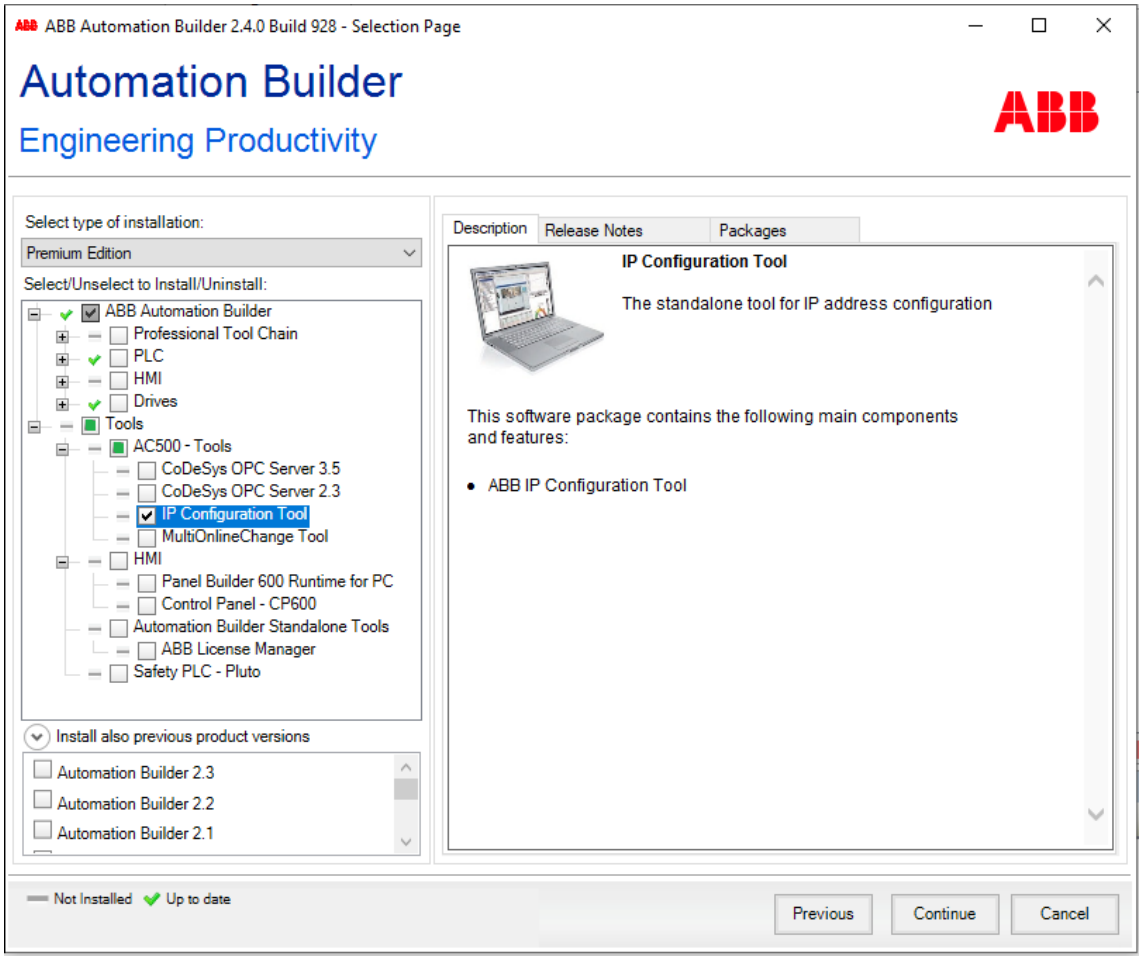
Stand-alone installation

The IP configuration tool is part of Automation Builder and can be called via “Tools ➔ IP-Configuration”. A separate installation is only required if the IP configuration tool shall be used stand-alone.

1. Open the Installation Manager in Automation Builder: *"Tools ➔ Installation Manager"*.
2. Close all other instances of Automation Builder as only one instance of the program can be executed at a time.



3. Click *"Modify"* and select the *"IP Configuration Tool"* from the structure tree.



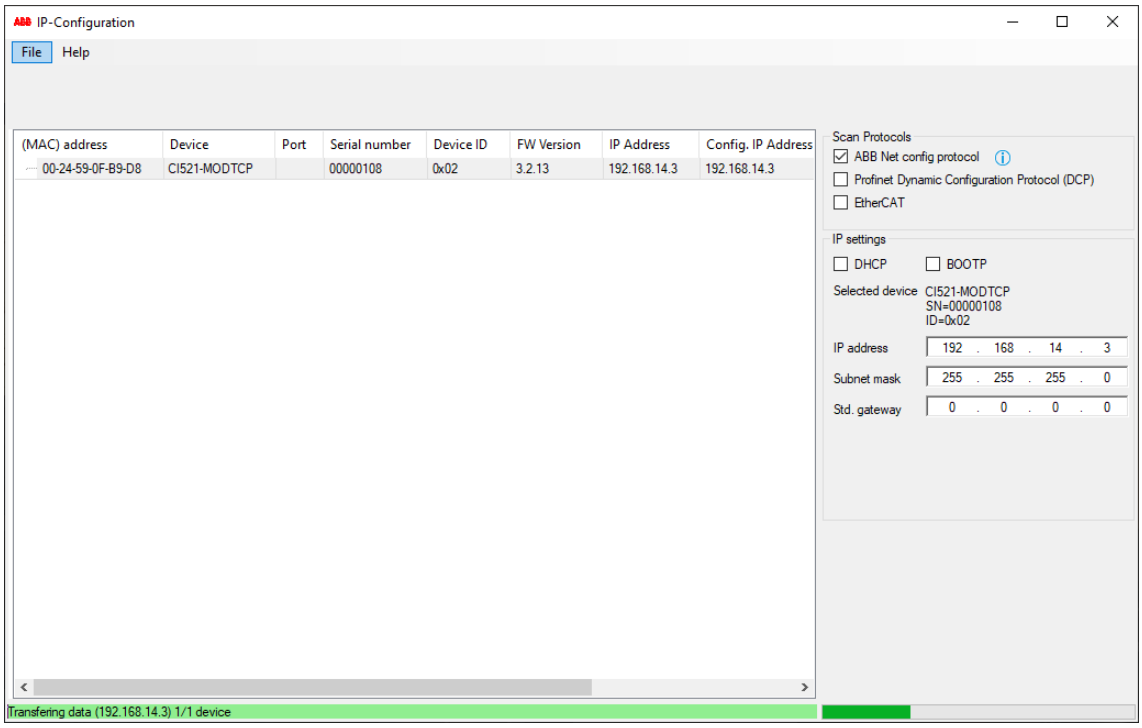
4. Click *"Continue"* to start the installation.
   - ⇨ After a successful installation the IP configuration tool is available as stand-alone tool (.exe).
   - ⇨ To start the IP configuration tool, click the new created desktop icon.

Using the tool functions

Network scan

With a network scan all devices that have been found in the network by the scan process are listed, i.e. ABB devices such as AC500 processor modules, AC500 communication interface modules or ABB Drives.

1. Start the IP configuration tool in Automation Builder (*"Tools ➔ IP-Configuration"*) or start it stand-alone (.exe).
2. The *"IP-Configuration"* dialog opens. Define the device type for the network scan by selecting the desired option under *"Scan Protocol"*:
   - *"ABB Net config protocol"*:
     Use this option for AC500 devices such as processor modules, CI5xx-Modbus devices or ABB Drives. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection.
   - *"Profinet Dynamic Configuration Protocol (DCP)"*:
     Use this option for PROFINET communication interface modules. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection (not via CM579).
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 3730
   - *"EtherCAT"*:
     Use this option for EtherCAT communication interface modules. The Ethernet cable must be connected directly to the first EtherCAT slave device of the EtherCAT fieldbus. Ensure that no EtherCAT master device is available on the bus when a scan is performed.
     "Emergency" option: Enable this option to check on failures in the EtherCAT assembly during the scan process, i.e. a frame loss or interchanged ports. Errors are displayed.
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 3730
3. Click *[Scan]* to start the scan process.



4. All devices that have been found in the network are listed including hardware and connection details. The following details can be changed under *"IP settings"*:

   ⇨ ● *"IP Address"*:
   Current IP address of the device.
   ● *"Conf. IP Address"*:
   Configured IP address of the device. A changed IP address will update this column.
   ● *"FW Version"*:
   Current installed firmware version of the device. This field is visible not until a first network scan. If this field is still empty after a network scan, check on connection errors.
   Chapter 1.6.6.2.9.1.3.1 "Trouble-shooting for firmware update" on page 3733

The IP address of some devices, e.g. EtherCAT devices cannot be changed.

Changing the IP address

1. In order to change the IP address of devices perform a network scan.

   Chapter 1.6.6.2.9.1.2.1 "Network scan" on page 3725

2. Select a device from the list and select the appropriate protocol under *"Scan protocol"*.

   "DHCP" or *"BOOTP"* option: If required, DHCP or BOOTP can be used to receive the IP address for the device from the server.

   "IP address", *"subnet mask"*, *"Std. gateway"*: Use these fields to change the IP address settings including the settings for the subnet mask and the standard gateway. Ensure that the combination of connection settings is correct.

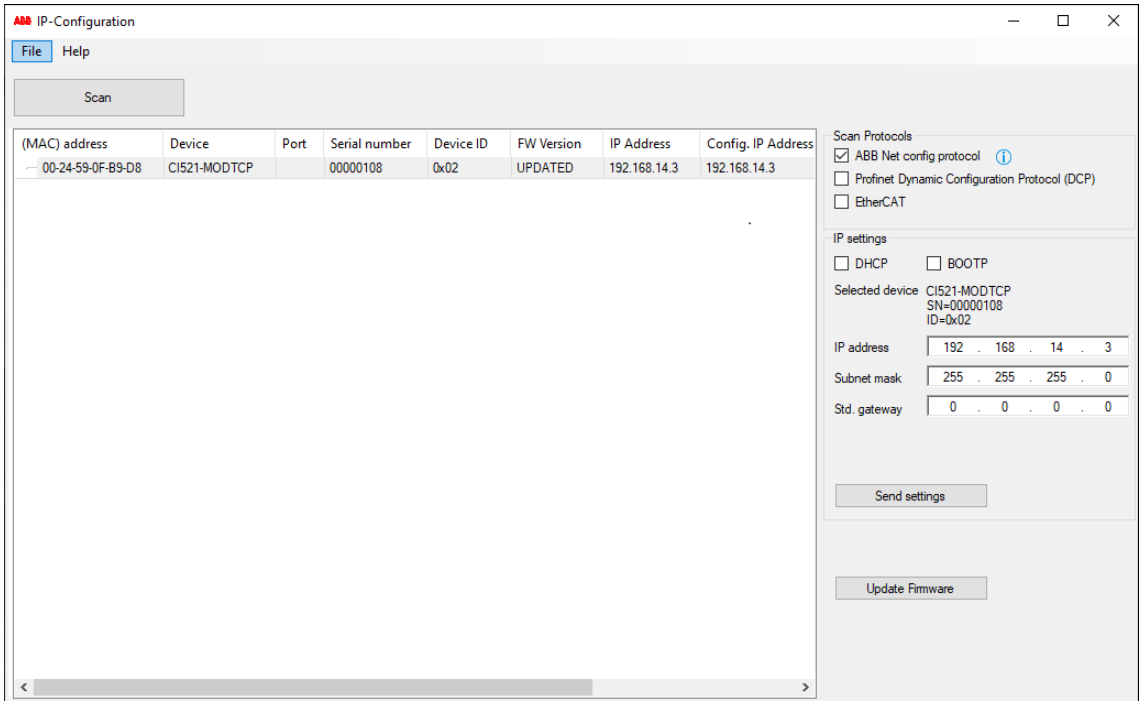
   "Check subnet configuration" on page 3733

Note for CI52x-Modbus devices

Consider the behavior of CI52x-Modbus devices if the last number of the IP address is set to "0".

"Check last number of IP address" on page 3734

3. Change the settings for the IP configuration and click *[Send settings]* to transmit the data to the device.



⇨

Note for PROFINET devices

The device name of PROFINET devices can be edited. If changing the name, ensure the following rules apply:

- *Labels must be separated by "."*
- *Total length: 1 to 240*
- *Label length: 1 to 63*
- *Labels can consist of characters [a-z] and numbers [0-9]*
- *Labels are not allowed to start with "-"*
- *Labels are not allowed to end with "-"*

4. In order to keep all IP changes after a power cycle, the settings can be stored permanently. Confirm the prompted message during the scan process.

Firmware update

The firmware of AC500 communication interface modules can be updated with the IP configuration tool.

For this, the IP configuration tool must be used as stand-alone variant.

↳ *Chapter 1.6.6.2.9.1.1 “Stand-alone installation” on page 3723*

It is not possible to perform a firmware update out of Automation Builder.

- *For PROFINET communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.3.3.*
- *For EtherCAT communication interface modules a firmware update is only supported for devices with firmware version ≥ 2.1.4.*
- *For Modbus communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.2.13.*

Requirements: **Before the firmware update**

- Ensure a fast and stable network connection
- Close all unused applications on the executing PC
- Stop the communication between AC500 PLC and the communication interface module that shall be updated

During the firmware update

- Do not close the IP configuration tool
- Do not open Automation Builder software or any other application
- Do not switch-off the communication interface module that shall be updated
- Do not disconnect the Ethernet connection of a communication interface module or the executing PC

The firmware update will stop the operation of the affected device(s). Hence, the device(s) will become unresponsive for 1 - 2 minutes.

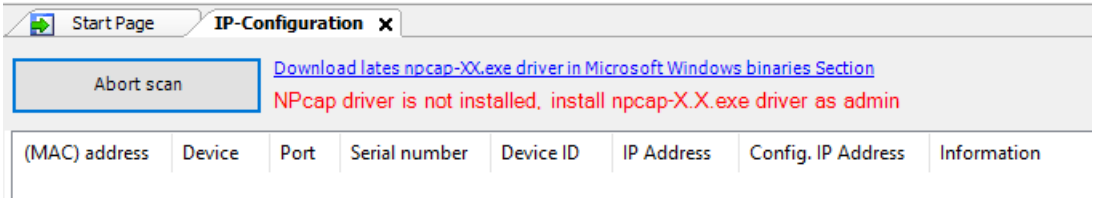
Procedure:

1. Start the IP configuration tool stand-alone (.exe).
2. Perform a network scan.

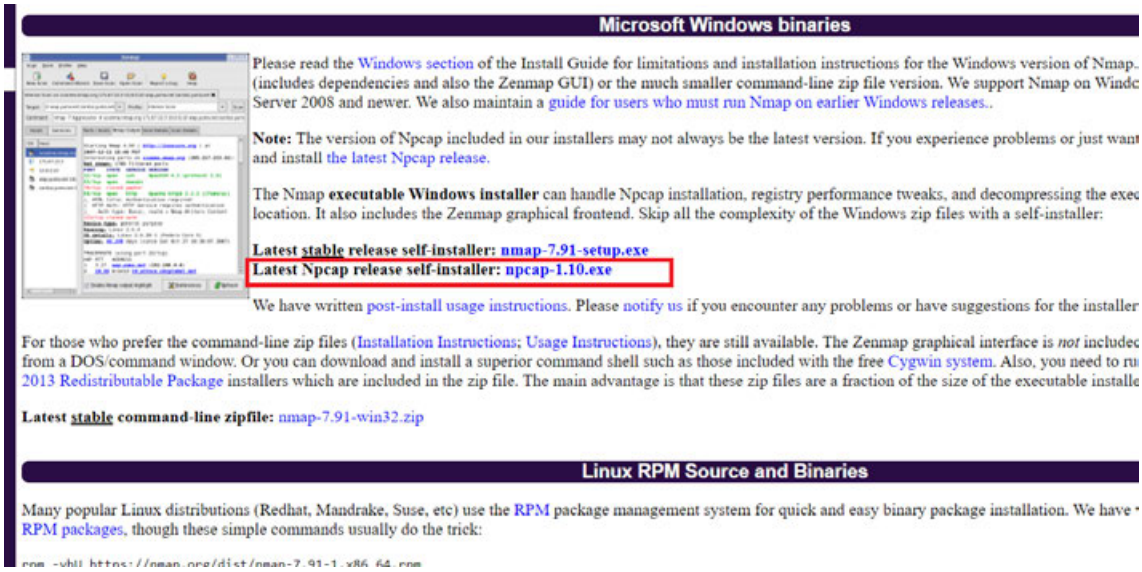
   Chapter 1.6.6.2.9.1.2.1 “Network scan” on page 3725
3. Select the devices that shall be updated from the list and click *[Scan]* to trigger the scan process.

   A multiple selection of several devices is possible via control key, however, ensure to select only devices of the same protocol at a time. Otherwise the firmware update fails.

4. This step is only required for devices that require an installed NPcap driver. In this case an appropriate message including a download link is prompted in the IP-Configuration dialog:



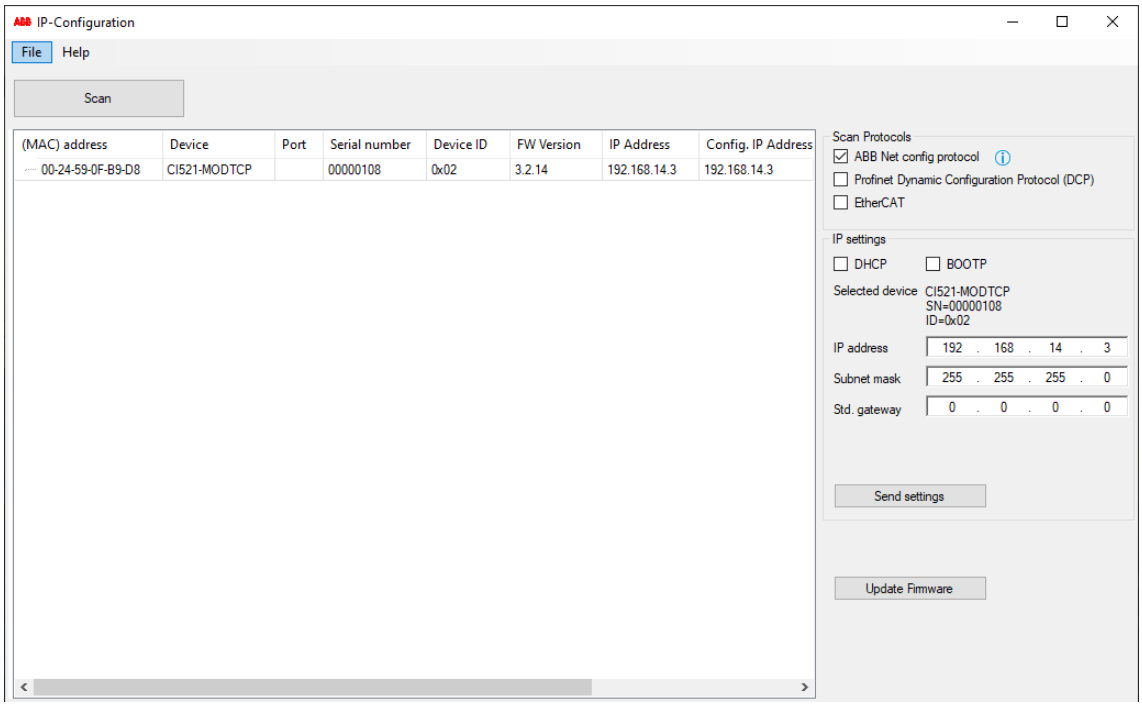
⇨ Click on the displayed link *https://nmap.org/download.html* and download the latest version of the *npcap-X.X.exe* file.



⇨ After the download, execute the file as administrator and restart the scan process.

⇨ The devices that have been scanned are listed.

5. Click *[Update Firmware]* to start the firmware update for the selected devices.



6. For CI50x, CI51x and CI52x devices a signature check is started. Select the appropriate firmware update file (*.bin) for the device(s). Example: C:\AC500\AC500_CI52x_Firmware_V3.2.8.bin.

   After a successful signature check the firmware update file (*.bin) and the respective signature file (*.bin.sig) are transferred to the device. This can last up to 3 minutes.

   If the signature check fails, check the availability of the *.bin file and the *bin.sig file.

   ↳ *"Signature check" on page 3734*

7. A status check followed by a device reboot followed by a second status check is performed automatically.

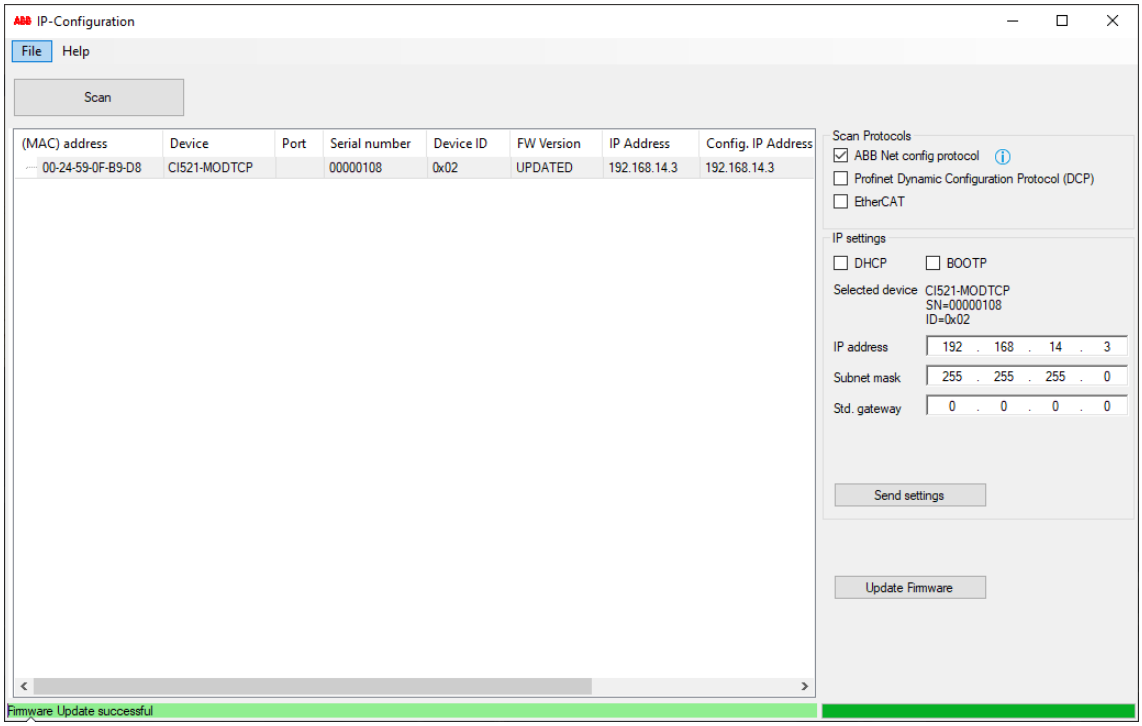
   > *After the firmware update all outputs of the updated devices are set to '0'.*

8. After a successful firmware update the update status or the new firmware version is displayed in the *"FW Version"* field.

   If this field is empty, there possibly is a connection error between the device and the executing PC.

   ↳ *"Error: Can't connect to device" on page 3735*

   Exception: For EtherCAT devices an empty *"FW Version"* field does not indicate a connection error.



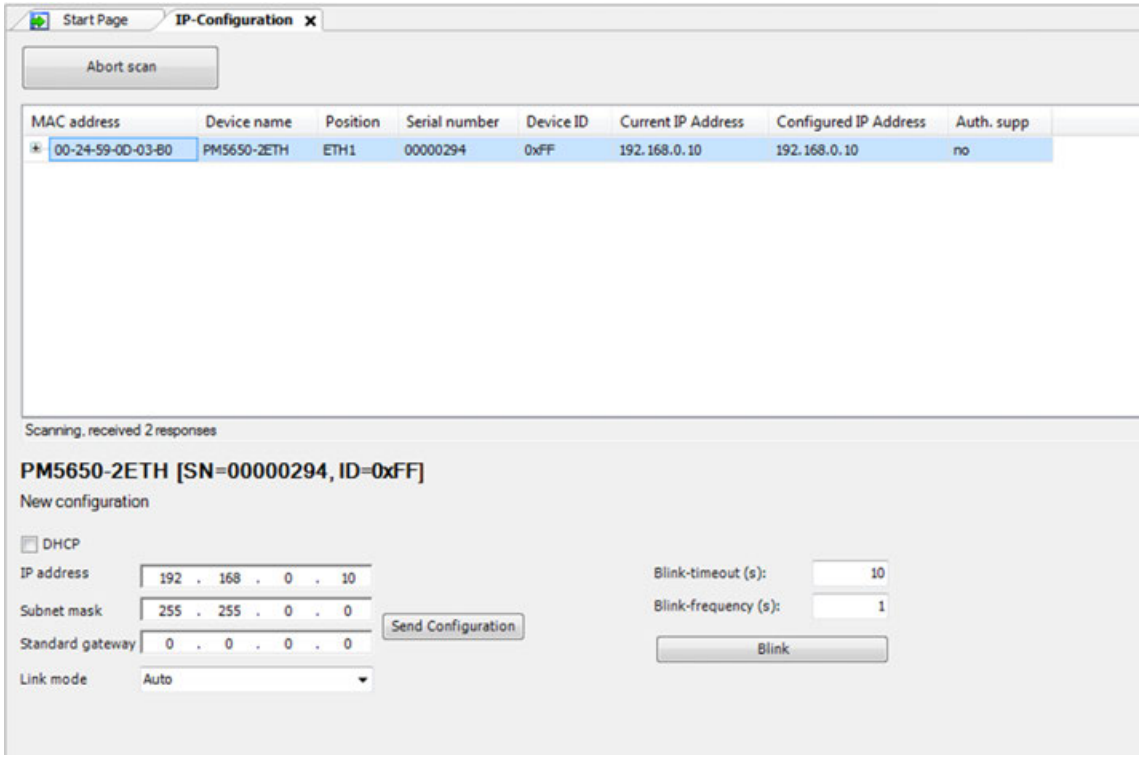
⇨ If the firmware update fails

- check the requirements for the update procedure.
  ↳ *"Requirements:" on page 3729*
- check the hints for trouble-shooting.
  ↳ *Chapter 1.6.6.2.9.1.3.1 "Trouble-shooting for firmware update" on page 3733*
- perform a network scan and repeat the update. If the error still persists power cycle the device and try the update again.

Blink functionality

This function activates flashing of the backlight of an AC500 LED display.

1. From the menu, select *"Tools ➔ IP-Configuration"*.
2. Click *[Scan]* to trigger the scan process for devices in the network.
   ⇨ A progress bar shows the progress. The IP settings of a selected device is displayed below the list and can be edited.
3. Adjust your desired time and click *[Blink]* to activate flashing.



Trouble-shooting for IP configuration tool

Firewall exceptions:

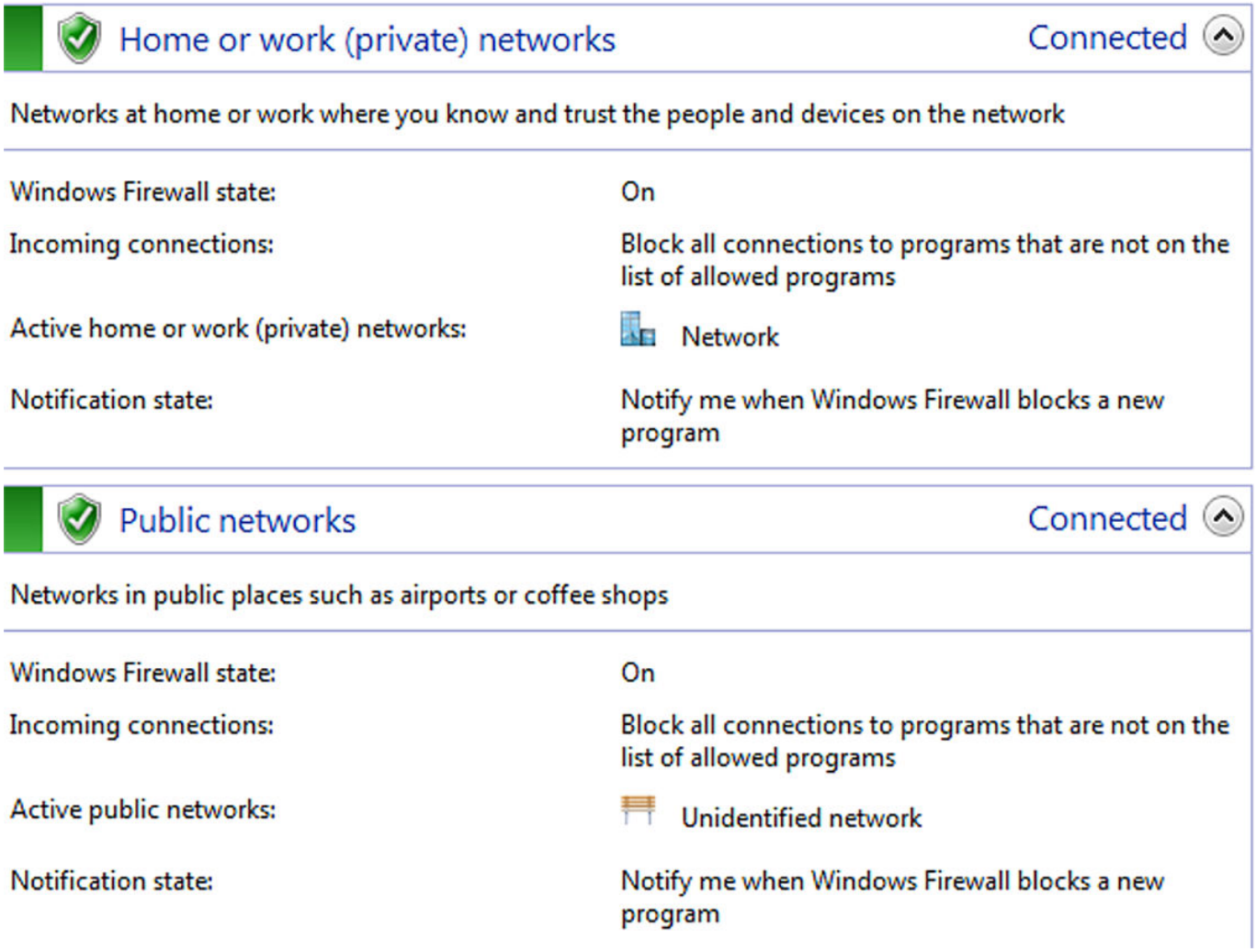
On a standard Windows 7 installation without third party firewall or security tools installed the IP configuration tool should work properly.

The Automation Builder setup installs rules or exceptions for the built-in Windows firewall to allow IPConfig to receive the responses for the IPConfig scan.

To check the Windows firewall is set correctly check the firewall settings.

Windows 7/ Windows 10:

On the network that is used for communication with the PLC, set *"Incoming connections"* to "Block all connections to programs that are not on the list of allowed programs".



If a third party firewall is used these exceptions must be configured manually.

> *Either exceptions for applications can be entered: Automation Builder and IP configuration tool must be added as application.*
>
> *Or the protocol and the port number must be given (for IPConfig: UDP protocol and port number 24576).*

Trouble-shooting for firmware update

Check the requirements

Ensure that all requirements have been considered before and during the update procedure.
↳ *"Requirements:" on page 3729*

Check subnet configuration

This hint is only valid for Modbus devices and PROFINET devices.

If the *"FW Version"* field is empty after the network scan or if the firmware version has not been updated after the update procedure, there possibly is a connection error between the device and the executing PC.

Ping the device from the executing PC. If no connection can be established, check whether the device and the PC are in the same subnet.

Example

PC	Device	Result
192.168.14.71 / 255.255.255.0	192.168.14.10 / 255.255.255.0	OK
192.168.**10**.71 / 255.255.**255**.0	192.168.**14**.10 / 255.255.**255**.0	ERROR
192.168.10.71 / 255.255.0.0	192.168.14.10 / 255.255.0.0	OK

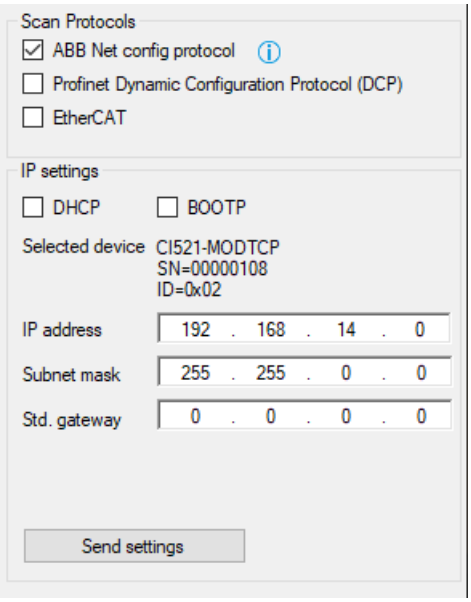
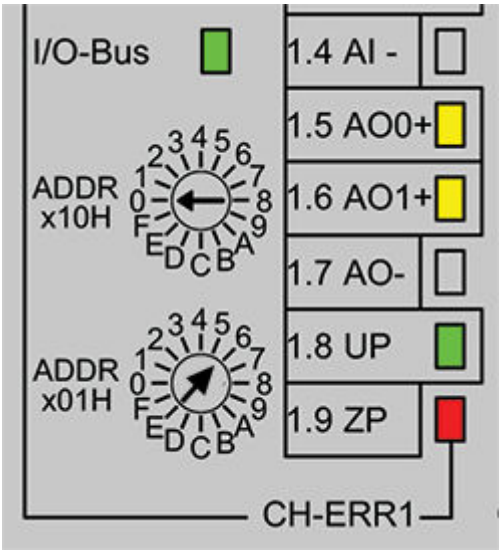
Click *[Scan]* again to restart the network scan. If the connection is successful a newer firmware version is displayed in the *"FW Version"* column.

Check last number of IP address

This hint is only valid for CI52x-Modbus devices.

Check the last number of the IP address. If it is set to "0", the IP address setting for this last number will be used from the rotary switches on the hardware device.

Example:

Automation Builder	AC500 communication interface module (rotary switch)
IP address: 192.168.14.0	IP address: 6
	
As a result, in the field *"IP Address"* the last number is set to "6": 	

Signature check

During the firmware update of CI50x, CI51x and CI52x devices a signature check is started.

The update procedure expects a firmware update file (**.bin*) and a signature file (**.bin.sig*) in the same directory. Without a signature file the signature check will fail.

Example:

Firmware update file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin***

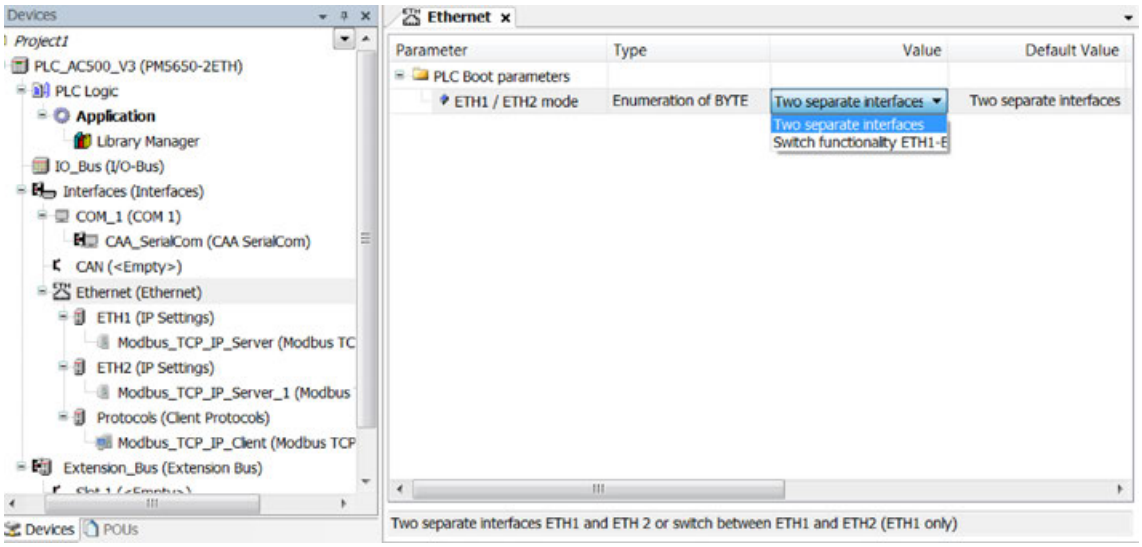
Signature file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin.sig***

Error: Package timeout	A timeout error may occure due to an instable network. Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.
Error: Unable to read device status	A read error may occure due to errors in the firmware update protocol. After the firmware update the IP configuration tool reads out the status of the updated device in order to check if the update was successful.
Error: IP is not unique	If an IP address is obtained by more than one device an error occures. A firmware update is not possible.
Error: Error State	Internal device error during the firmware update. Solution: Step 1: Scan again and repeat the firmware update. Step 2: If this does not work, power cycle the device, scan again and repeat the firmware update.
Error: Can’t connect to device	The TCP communication is not sufficient. Increase the connection quality. Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

Switch functionality of Ethernet interfaces ETH1/ETH2

As of SystemFW 3.1.0 the Ethernet interfaces ETH1/ETH2 can be configured as an Ethernet switch.

The default setting is *“Two separate interfaces”*.



The change of the PLC Boot parameter ETH1 / ETH2 mode will become active after a PLC reboot.

Create and download a Boot project before rebooting the PLC!

Parameter	Value	Description
ETH1 / ETH2 mode	Two separate interfaces	Two separate Ethernet interfaces ETH1 and ETH2
	Switch functionality ETH1-ETH2	Switch between ETH1 and ETH2

If the Switch functionality ETH1-ETH2 is active, only the Ethernet interface ETH1 is available (see Chapter 1.6.5.1.6.5.3 “Configuration” on page 3493). Any protocols configured under Ethernet interface ETH2 must be deleted. Otherwise a compile error will be created.

The setting of *ETH1 / ETH2 mode* can be checked on LED display with soft key <CFG> (see *Chapter 1.6.5.1.6.5.3 “Configuration” on page 3493*).

1.6.6.2.10 Onboard CAN configuration

AC500 V3 PLCs provide the following methods for CAN integration:

- Onboard CAN interface
- CANopen master-slave arrangement (with CM598-CN as a master device)

Table 655: Differences in supported protocols

	Onboard CAN	CM598-CN
CANopen Manager	X	
CAN 2A/2B	X	X
J1939	X	

Onboard CAN interface is not available on AC500-eCo V3!

Supported protocols

Onboard CAN interface supports the following protocols

- CANopen Manager: Connection of CI581 and CI582 without additional I/O modules
- CAN 2A/2B
- J1939

Configuration in Automation Builder is described in chapter *Chapter 1.6.6.2.11.1.1 "CM598-CAN - CANopen master communication module" on page 3737*.

Further information can be found in chapter *Chapter 1.6.6.2.16 "CAN onboard" on page 3800*

1.6.6.2.11 Communication modules

CANopen

CM598-CAN - CANopen master communication module

Configuration of the communication module

– *Click menu "Tools ➔ Options" and select "Device editor" in the Options window.*
– *Enable first checkbox Show generic device configuration views and click [OK].*

Append a CM598-CAN

1. Right-click on your desired *Slot* below node *"Extension_Bus"* and click *"Add object"*.
   ⇨ Dialog *Replace object:* appears.
2. Click *CM598_CAN* in the list and click *[Replace object]*.
3. Double-click *"CM598_CAN (CM598-CAN)"* to get the *"CM598-CAN Parameters"* in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the CM598-CAN communication module.
Bus behavior	Asynchronous (IEC bus cycle)	Asynchronous (IEC bus cycle)	Not yet supported.

Parameter	Default value	Value	Description
		Synchronous (start of bus cycle)	Not yet supported.
Node ID	1	1 - 127	Identifier of the device within CANopen.

The tab *“CAN Bus”* contains the basic settings of the CAN bus and special settings for the CAN 2.0 B protocol.

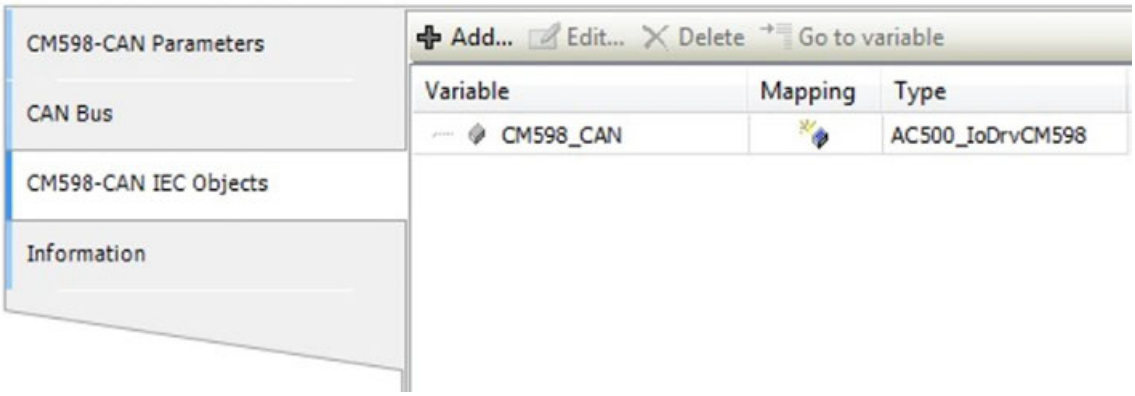
The settings at “29 Bit COB-ID” are only valid for CAN 2.0 B protocol. Ensure the option “Enable 29 Bit COB-ID” is enabled. Otherwise no CAN 2.0 B frames can be received. With the other parameters at “Enable 29 Bit COB-ID” the receive filter is configured.

Possibilities for using the SAE J1939 protocol in AC500 V3 PLCs are described in the *application example*.

Parameter	Default value	Value	Description
Bus parameters			
Transmission rate	250 kBit/s	10 kBit/s 20 kBit/s 50 kBit/s 100 kBit/s 125 kBit/s 250 kBit/s 500 kBit/s 800 kBit/s 1000 kBit/s	Transmission speed in [kBit/s]
Node settings			
Stop in case of monitoring error	Disabled	Disabled	The manager does not stop in case of a monitoring error (Node Guarding or Heartbeat Error). A loss of communication to one node has no influence to other nodes. The manager tries to reestablish the communication to the error affected nodes.
		Enabled	If this function is enabled, the manager will also stop the communication to all responding and active nodes. Not yet supported.
Send "Global Start Node"	Enabled	Disabled	No "Global Start Node" message is sent after configuring the nodes.
		Enabled	A "Global Start Node" message is sent after configuring the nodes. This synchronize all Nodes again. Not yet supported.

Parameter	Default value	Value	Description
29 Bit COB-ID			
Enable 29 bit COB-ID	Disabled	Disabled	29 bit CAN-IDs are disabled, but 11 bit CAN-IDs are still enabled.
		Enabled	29 bit CAN-IDs are additional enabled.
Acceptance mask	0	29 bit	Specifies the bits of a CAN-ID which will be evaluated by the filter. For instance, with an acceptance mask = 0x1FFFFFFF all bits are evaluated.
Acceptance code	0	29 bit	Specifies the bits of a CAN-ID which has to be set to pass the filter. Only those bits which are set in the acceptance mask are relevant.

The tab *"CM598-CAN IEC Objects"* contains the created instance of the IO driver.



Configuration of the protocols CAN 2.0 A / CAN 2.0 B

The Communication Module CM598-CAN can be used to realize CAN bus based networks in combination with library ABB_CM598Can_AC500.library.

To enable the support for the desired protocol it must be appended to CM598-CAN.

1. Right-click *"CM598_CAN (CM598-CAN)"* in the device tree and select *"Add device"* in the context menu.
   ⇨ Window *Add object below: CM598_CAN_1* appears.
2. Select *"CAN 2.0 A"* or *"CAN 2.0 B"* from the list.

Parameterization

The CAN data transmission requires a buffer for the incoming data that can be read with function blocks of library ABB_CM598Can_AC500.library.

1. Right-click *“CAN_2_0A_11_bit_identifier_ (CAN 2.0A)”* or *“CAN_2_0B_29_bit_identifier_ (CAN 2.0B)”* and select *“Add object”*.
2. Select *“Buffer for CAN 2A”* for CAN 2.0A. Or select *“Buffer for CAN 2B”* for CAN 2.0B from the list.
3. Double-click on *“Buffer_for_CAN_2A (Buffer for CAN2A)”* or *“Buffer_for_CAN_2B (Buffer for CAN2B)”* in the device tree to open the Buffer configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Identifier	0	CAN 2A: 0 ... 2047 CAN 2B: 0 ... 536870911	The value of the CAN identifier that is compared with the identifier of the incoming telegrams. The telegrams will be added to the buffer if the identifier matches.
Number of receive buffers	1	1 ... 16	The size of the buffer in number of telegrams.
Behaviour on receive buffer overflow	Overwrite	Overwrite	The oldest telegram in the buffer is overwritten by the incoming telegram.
		Discard	Incoming telegrams are discarded as long as the buffer is full.
Enable triggering of IEC task	No	No	Disables the triggering of the execution of the related IEC task.
		Yes	Enables the triggering of the execution of the related IEC task as soon as a CAN frame with the specified CAN-ID arrives ↳ *Chapter 1.6.5.2.1.1 “Triggering of event tasks with CAN-IDs” on page 3599*.

Configuration of the CANopen master

↳ *Chapter 1.6.6.2.16.1.1 “CANopen manager (master)” on page 3800*

PROFINET

CM579-PNIO - PROFINET IO controller

For Automation Builder < 2.2.0

Configuration in Automation Builder is described in PROFINET IO configuration ↳ *Chapter 1.6.6.2.18 “PROFINET IO Configurator” on page 3832*.

For Automation Builder >= 2.2.0

PROFINET IO

CM579-PNIO – PROFINET IO communication module

Configuration of the communication module

Configuration is valid as of CPU FW 3.2.0.

Append a CM579-PNIO

1. Right-click on your desired *Slot* below node *“Extension_Bus”* and click *“Add object”*.
   ⇨ Dialog *Replace object:* appears.
2. Click *CM579_PNIO* in the list and click *[Replace object]*.
3. Double-click *“CM579_PNIO (CM579-PNIO)”* to get the *“CM579-PNIO Parameters”* in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the CM579-PNIO communication module.
Bus behavior	Asynchronous (IEC bus cycle)	Asynchronous (IEC bus cycle)	The bus cycle and the IEC Application are running asynchronously. The IO update rate between the Profinet IO Controller and the IEC Application is defined with the bus cycle task.

Configuration of the PROFINET IO controller

The PROFINET IO Controller node appears automatically below the added Communication Module CM579-PNIO.

PROFINET IO controller - Configuration

▷ Double-click on *“PNIO_Controller”* and open the tab *“General”* in the editor window.

The following parameters are available:

Parameter	Default	Value	Description	Parameter
Station name	CM579	Up to 240 characters	Network name of the PROFINET IO controller station. Must be a valid hostname.	Station name
IP parameters				
IP-Address	192.168.0.1	Valid IP address	IP address of the PROFINET IO controller station.	IP address

Parameter	Default	Value	Description	Parameter
Subnetmask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO controller station.	Subnet Mask
Default gateway	0.0.0.0	Valid gateway address	Gateway address of the PROFINET IO controller station.	Default gateway
Station name	controller	Up to 240 characters	Network name of the PROFINET IO controller station. Must be a valid hostname.	Station name
Address settings for devices				
First IP-Address	192.168.0.2	Valid IP address	First IP address of the PROFINET IO devices. This parameter determines the address range of the PROFINET IO devices in combination with parameter Last IP address.	First IP address
Last IP-Address	192.168.0.254	Valid IP address	Last IP address of the PROFINET IO devices. This parameter determines the address range of the PROFINET IO devices in combination with parameter First IP address.	Last IP address
Subnetmask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO devices.	Default subnet mask
Default gateway	0.0.0.0	Valid gateway	Gateway address of the PROFINET IO devices.	Default gateway address

PROFINET IO controller - Parameters

The tab *“PROFINET-IO-Controller Parameters”* is a generic view of all PROFINET IO controller parameters. It is normally hidden and is normally not needed for configuration.

Use tab “PROFINET-IO-Controller Parameters” only, if you need to change a parameter, which is not visible in other dialogs.

Activating tab

1. Click *“Tools ➔ Options”* and select *“Device editor”*.
   ⇨ The *Device editor* dialog opens.
2. Enable checkbox *Show generic device configuration views* and click *[OK]*
   ⇨ The tab is now available.

PROFINET IO controller - I/O mapping

In this tab the bus cycle task can be specified. It is possible to select a particular task of the IEC application by its name or to use the option *“Use the parent bus cycle setting”*. In the latter case the setting of the *Bus cycle options* in *“PLC_AC500_V3 ➔ PLC settings”* are used.

Configuration of PROFINET IO devices

Add PROFINET IO device

1. Right-click on node *“PNIO_Controller (PROFINET-IO-controller)”* and click *“Add object.”*
   ⇨ A list with all installed PROFINET IO devices appears.
2. Slelect the desired device and click *[Add object]*.
   ⇨ The device is added to the Profinet IO Controller in the device tree.

PROFINET IO device - Configuration

Double-click on *“PNIO-Device”* to open the device configuration in the editor window.

The following parameters are available:

Parameter	Default	Value	Description	Parameter
Identification				
Station name	Device-specific	Up to 240 characters	This is a system wide unique name for addressing the device. Must be a valid hostname.	Slave parameters -> Identification -> Station name
Communication Parameter				
Send clock (ms)	Device-specific	0.25 0.5 1 2 4	Parameter Send clock determines the SendCycle. SendCycle = Send clock x Reducation ratio <= 512ms x	Slave parameters -> Reduction ratio
Reduction ratio	Device-specific	1...16384	The Reduction ratio determines the factor for calculating the cycle time. Cycle time = Send clock x Reducation ratio	Slave parameters -> Reduction ratio
Phase	1	1…Reduction ratio	Defines the part of the SendCycle at which an IO frame is sent.	Phase
Watchdog factor	3	1...65535	The Watchdog time is calculated as Watchdog time = SendCycle * Watchdog factor. The transfer of a IO telegram is always checked of the consumer side. Within this time the next IO telegram must be received by a consumer. Otherwise it is checked if the Datahold has been expired too.	Watchdog interval
RT Class				
RT Class	RT Class 1 Data-RTC-PDU	RT Class 1 Data-RTC-PDU	Defines the Realtime Class of cyclic data. Currently only RT Class 1 (legacy) and RT Class 1 are supported.	Slave parameters -> RT Class

Parameter	Default	Value	Description	Parameter
		RT Class 2 Data-RTC-PDU		
		RT Class 3 Data-RTC-PDU		
		RT Class UDP-RTC-PDU		
VLAN ID	0	0..4095 or 0..32767	In case of VLAN usage the parameter VLAN ID represents the ID of the virtual network. For VLAN type 802.1Q the range is 0..4095 while VLAN type ISL accepts values from 0 to 32767. The supported type depends on the used device.	Slave parameters -> VLAN ID
IP Parameter				
IP-Address	192.168.0.8	Valid IP address	IP address of the PROFINET IO Controller station.	Slave parameters -> Identification -> IP address
Subnetmask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO Controller station.	Slave parameters -> Identification -> Subnet mask
Default gateway	0.0.0.0	Valid gateway address	Default gateway address of the PROFINET IO Controller station.	Slave parameters -> Identification -> Default gateway address

PROFINET IO device – Timing parameters

In the current implementation are 2 Communication Relations (CR) between the controller and the device defined.

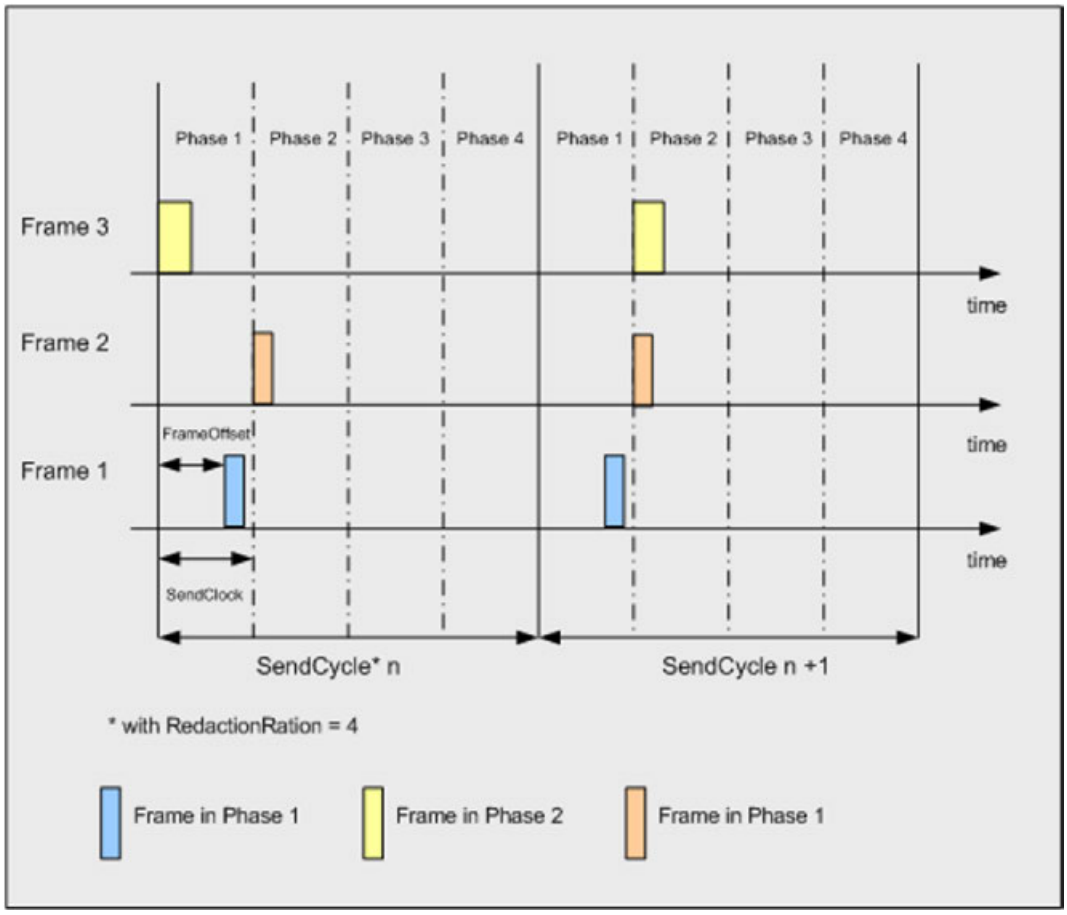
One describes the I/O telegram from the controller to the device (outputs), the other the I/O telegram from the device to the controller (inputs).

The timing of the corresponding I/O telegrams can be defined separately for each device.

Editable timing parameters are:

- Send clock
- Reduction ratio
- Phase

The relation between these parameters is shown in the following drawing.



For each device a SendCycle must be configured, which determines the sending interval of I/O frames. It is based on a time base of 31.25 µs and is calculated as:

SendClock [ms] = SendClockFactor * 31.25 µs / 1000.

The cycle time of an I/O telegram is defined by the SendCyle.

It's calculated as:

SendCyle [ms] = SendClock [ms] * Reduction Ratio.

The values of the individual parameters are limited by the maximum value 512 ms of the SendCycle. The following table summarizes the relation of the timing parameters.

Parameter	Description	Relation	Range
SendCycle	Is the cycle time of a RT telegram.	SendCyle = Send-Clock * Reduction ratio.	1ms..512 ms
SendClock	The SendCycle is divided into several time slots. The SendClock defines the size of a time slot within the SendCyle.	SendClock = SendClock factor * 31.25 µs;	SendClock * Reduction ratio ≤ 512 ms
SendClock factor	Is multiplied with the time base 31.25 µs to calculate the SendClock.	SendClock factor = Send-Clock / 31.25 µs	1..128
Reduction ratio	The reduction ratio defines the number of time slots within the SendCyle.	SendClock * Reduction factor ≤ 512 ms	1..16384
Phase	The time slot in which the IO frame is sent.	A integer value of the range 1 … Reduction ratio	1..16384

PROFINET IO device – PNIO parameters

The tab *"PNIO Parameters"* is a generic view of all PROFINET IO device parameters. It is normally hidden and is normally not needed for configuration.

> *Use tab "PNIO Parameters" only, if you need to change a parameter, which is not visible in other dialogs.*

Activating tab

1. Click *"Tools ➔ Options"* and select *"Device editor"*.
   ⇨ The *Device editor* dialog opens.
2. Enable checkbox *Show generic device configuration views* and click *[OK]*
   ⇨ The tab is now available.

Configuration of 3rd party PROFINET IO devices

Before a 3rd party PROFINET IO device can be used, the provided GSDML file has to be installed in the *Device Repository*.

Installation Go to *"Tools ➔ Device Repository ➔ Install"*.

Configuration See *Chapter 1.6.6.2.11.2.1.2.1.1.3 "Configuration of PROFINET IO devices" on page 3744*.

I/O mapping of the PROFINET IO devices

Open I/O mapping list

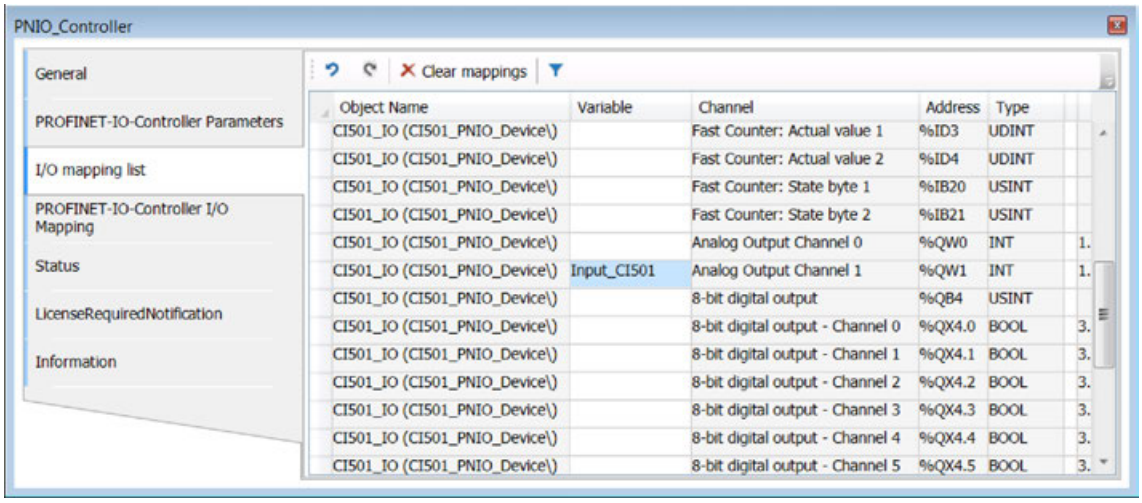
1. Doube-click on the *"PNIO_Controller"* or below on the *"<...>PNIO-Device"* or below on the *"I/O-Module"* in the device tree.
2. Select tab *"I/O mapping list"* to show the list of I/O channels.

The content of the list depends on the selected node.

For instance:

- When the *"PNIO_Controller"* node is selected all I/O channels of all configured devices are shown.
- When a *"PNIO-Device"* is selected all I/O channels of the configured modules are shown.

An IEC variable for an I/O channel that is available in the Application can be defined by double-clicking in column *Variable*.



CM589-PNIO PROFINET IO device communication module

> *Configuration is valid as of CPU FW 3.5.0 and "CM589-PNIO" FW 1.6.2.20.*

The configuration of the *"CM589-PNIO"* PROFINET IO device module has to be done in the following steps:

- Parameterization of the AC500 communication module interface *"Parameterization - CM interface" on page 3748*
- Parameterization of the PROFINET IO device protocol stack *"Parameterization - PROFINET IO stack" on page 3749*
- Configuring PROFINET IO device module structure *"Configuring PROFINET IO structure" on page 3749*
- Parameterization of the PROFINET IO device modules *"Parameterization - PROFINET IO device modules" on page 3750*

Parameterization - CM interface

For connecting a PLC as *"PROFINET IO device"*, plug *"CM589-PNIO"* at the *"Extension_Bus"* node.

Double-click on *"CM589-PNIO"* to open the *"CM589-PNIO"* configuration in the editor window.

The following parameter is available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started even in case of configuration error.

Click on tab *[CM589-PNIO I/O Mapping]* to get *"Bus Cycle Options"* in the editor window.

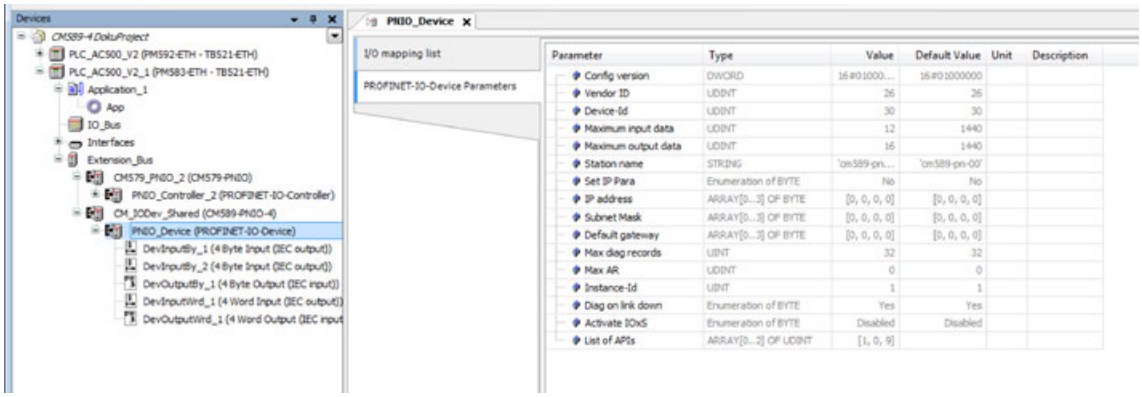
The following parameter is available:

Parameter	Default value	Value	Description
Bus cycle task	Use parent bus cycle settings	Use parent bus cycle settings	Settings from “PLC settings” tab are used.
		Name of task	Name of task that triggers the bus cycle

Click on tab *[CM589-PNIO IEC-Objects]*. Here the IO driver instance of communication module is specified.

Parameterization - PROFINET IO stack

“PROFINET IO device” protocol does not need user configuration. All needed parameters are set automatically by Automation Builder. Double-click on *“PROFINET IO device”* in tree view will show the parameter set in tab *“PROFINET IO device parameters”*. The parameters are displayed just for information and in read-only mode.

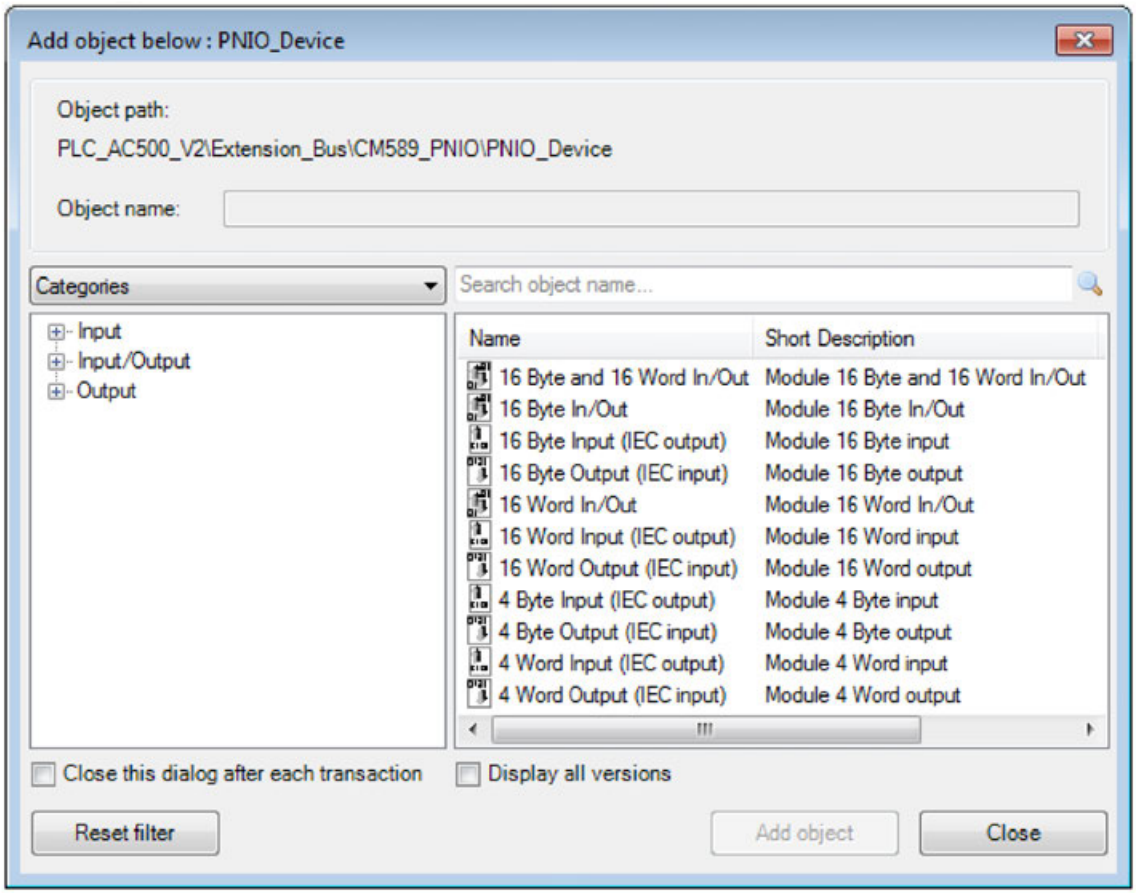


– *Station name: the default name is displayed. The real name used on acting at the field bus is combined out of this default name and the used setting of the rotary switches at the CM589 module (“cm589-pnio-00”, “00” will be replaced by rotary switch value) or the name set via PROFINET set name service.*
– *Parameter “IP address”, “Subnet Mask”, “Default Gateway”: the default values are displayed here. These values are not used as communication settings. “PROFINET IO controller” supplies the IO devices with IP settings on communication establishing.*

Configuring PROFINET IO structure

“CM589-PNIO” provides I/O data as modules with different data types and directions. Create an application specific I/O structure by compiling an appropriate combination of modules.

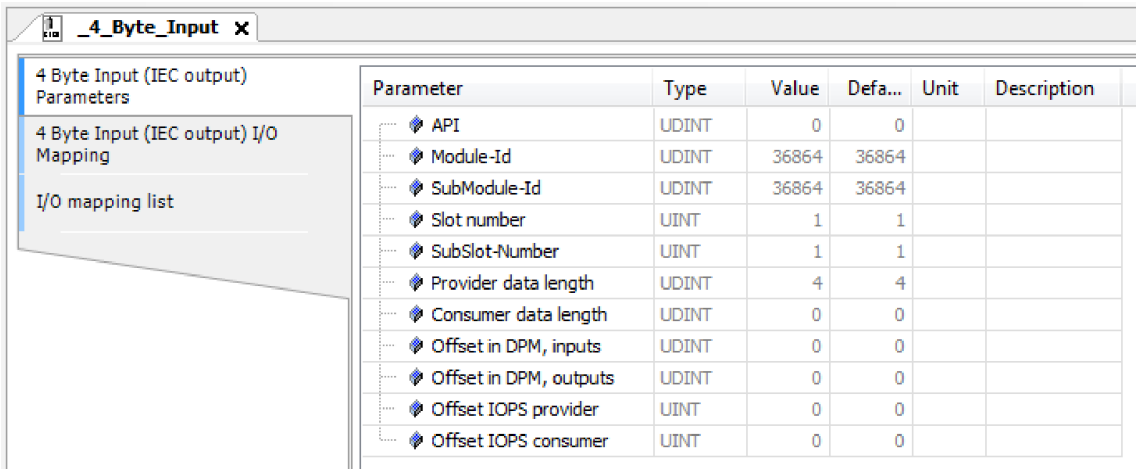
To assign I/O modules to *“PROFINET-IO-device”* node open *“Add Object”* dialog.



If *“CM589-PNIO”* module does not support the number of I/O data configured, download configuration will fail. Currently 1440 bytes are supported for inputs and 1440 bytes for outputs. See *“Calculating size of I/O data” on page 3751* how to calculate number of I/O data occupied by certain configuration.

Parameterization - PROFINET IO device modules

PROFINET-IO device modules do not need user configuration. All needed parameters are set automatically by Automation Builder. Double-click on a module node shows the parameter set just for information. This parameter set is identical for all module types and is displayed in read-only mode.



- *API: shows the API which is used by the CM589-PNIO modules. As API 0 is supported only CM589-PNIO modules do not provide configuration capabilities for this parameter*
- *Slot number, Sub-Slot-Number, Offset in DPM, inputs/outputs: will be set to default values on inserting a module. On creating configuration data Automation Builder calculates real values and overwrites the defaults*
- *Offset IOPS provider/consumer: not used and set to 0 values*

Calculating size of I/O data

PROFINET defines IO data and status information to be exchanged between IO controller and IO device. The status information is called *“Provider Status”* and *“Consumer Status”*. Both (IO data and status information) have to be considered on calculating allocated memory in input and output image.

- The number of status bytes depends on the type of module used.
- The different types of modules input, output and in/output have to be considered different.
- Some status bytes are reserved for predefined submodules have to be considered additionally.

A configured IO module allocates memory space at the corresponding IO image for data and status bytes. Additionally memory is allocated at the opposite directions IO image to store further status bytes. E.g. an input module allocates memory at the input image but additionally it allocates one byte for status at the output image. Summarized size of input and output data and status has to fit to the corresponding image.

See following table for an overview of IO module types and corresponding status bytes:

Module Type	**Input Data**			**Output data**		
	Inputs	**Provider Status Inputs**	**Consumer Status Outputs**	**Outputs**	**Provider Status Outputs**	**Consumer Status Inputs**
Reserved	0 Input Bytes	4 Bytes	0 Bytes	0 Bytes	0 Bytes	4 Bytes
Input Module	n Input Bytes	1 Byte	0 Bytes	0 Bytes	0 Bytes	1 Byte
Output Module	0 Input Bytes	0 Bytes	1 Byte	n Output Bytes	1 Byte	0 Bytes
Input/ Output Module	n Input Bytes	1 Byte	1 Byte	n Output Bytes	1 Byte	1 Byte

Following expressions calculate allocated sizes of input and output data:

Size Input =	**Input + Status + 4 bytes (reserved status)**
Size Output =	**Output + Status + 4 bytes (reserved status)**

- Input = summarized number input bytes all modules
- Output = summarized number output bytes all modules
- Status = count input modules + count output modules + 2 * count input/output modules

Mapping of the I/Os

Double-click on the desired *“PROFINET-IO-device”* module object in the device tree to show current I/O mappings connected to this module.

See chapter Symbolic Names for Variables, Inputs and Outputs ↳ *Chapter 1.6.6.2.13.7 “Symbolic names for variables, inputs and outputs” on page 3776* for further details on mapping inputs and outputs.

Diagnosis and debugging for AC500 V3 products

“CM589-PNIO” – PROFINET device diagnosis

The diagnosis messages of Communication module *“CM589-PNIO”* are displayed in tab *[Diagnosis]* of node CM589-PNIO in device tree of Automation Builder. Within PLC application they can be read with the diagnosis methods of IO driver or *“Function Block Diag”*.

↳ *Chapter 1.7.1.4.3.2 “Device state” on page 4035*

↳ *Chapter 1.7.1.4.2.2.1 “Method Ack / DiagAck: acknowledgement” on page 4029*

In PLC display the diagnosis messages of *“CM589-PNIO”* are not shown.

The following diagnosis messages are signaled by *“CM589-PNIO”*:

Error severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0	0	1000	No communication module or wrong type found	Plug the correct communication module
3	0	0	1001	Type of CM589-PNIO not supported	Exchange the communication module
3	0	0	1002	Firmware version of CM589-PNIO not supported	Update firmware of CM589-PNIO
3	0	0	1003	Identification of communication module failed	Exchange the communication module or plug the correct communication module
3	0	0	2000	Watchdog error	
3	0	0	2001	CM589-PNIO is not communicating	Check bus connection and configuration
3	0	0	2002	CM589-PNIO signals communication error	Check bus connection and configuration
3	0	0	2003	Starting of CM589-PNIO's protocol stack failed	Check bus connection and configuration
3	0	0	2004	Stopping of CM589-PNIO 's protocol stack failed	
3	0	0	2005	PLC cannot be set to run due to an error of CM589-PNIO	Check error log and correct errors
3	0	0	3000	Configuration error	Check configuration and correct errors
3	0	0	3001	Configuration version mismatch	Use matching CPU firmware version
3	0	0	4000	Ethernet link down	Check Ethernet cable connection

CM589-PNIO - PROFINET IO slave

Configuration in Automation Builder is described in PROFINET IO Slave configuration ↬ *Chapter 1.6.6.2.18 “PROFINET IO Configurator” on page 3832*.

EtherCAT

CM579-ETHCAT - EtherCAT I/O master

Configuration in Automation Builder is described in EtherCAT master configuration ↬ *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*.

Parameterization of the CM579-ETHCAT communication module interface

- Double-click on *“CM579_ECAT (CM579-ECAT)”* to open the CM579-ECAT configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the EtherCAT Communication Module.
Broken slave behavior	Leave all broken slaves down	Leave all broken slaves down	Broken slaves will not be served.
		Leave addressless slaves down	Only slaves without address will be left down.
		Leave no slaves down	Broken slaves will be ignored.
Distributed clocks	Inactive	Inactive	Distributed clocks are inactive.
		Active	Distributed clocks are active.
Bus Target State	Operational, OP	Operational, OP	Target state of the EtherCAT bus at application start.
		Safe-Operational, SAFEOP	
		Pre-Operational, PREo	
Bus behavior	Asynchronous (IEC bus cycle)	Asynchronous (IEC bus cycle)	Type of bus behavior (asynchronous/synchronous)
		Synchronous (Sync mode 1)	Minimum lag (1 bus cycle) between input and output values.
		Synchronous (Sync mode 2)	Extended application time, higher lag (2 bus cycles) between input and output values.
Optimize I/O update	Off	On	When activated, consecutive I/Os are merged in one block to optimize the performance.
		Off	

EtherCAT-Master - ABB functionality for sync units

With the EtherCAT sync units, several slaves are configured into groups and subdivided into smaller units. For each group, the working counter can be monitored for an granular input data validation. As soon as a slave is missing in a sync unit group, the input data of all other slaves in the same group becomes invalid.

Detection occurs immediately in the next bus cycle, as the working counter is continuously checked. Unaffected groups remain operable without any interference.

Right click on the *“Application”* node and press *“Create configuration data”*.

Automation Builder creates a set of global variables defining the working counter state of a SyncUnit command.

The variables use the following naming scheme:

"SLOT_" + "CouplerSlot_ " + "SyncUnitName" + "_CMD_" + "LogicalAccess" + "_FRAME_" + "FrameID CouplerSlot".

CouplerSlot

The communication module slot is the ID of the slot where the communication module is plugged in.

SyncUnitName

The sync unit name is as defined in the *“Sync Unit Assignment”* tab.

LogicalAccess

The logical access defines the command List of logical access commands:

- Read = 10;
- Write = 11
- Read/Write = 12

FrameID

The frame ID starts with 1 and increments if the cyclic exchanged data is larger than the maximum Ethernet frame boundary.

Values

- FALSE : Working counter is as expected (data from slaves is valid)
- TRUE : Working counter is different to expected value (data from slaves is invalid)

The variables can be used by conditional consumption of slave data in the application:

```
IF NOT SLOT_1_default_CMD_10_FRAME_1 THEN
(* Consume SyncUnit default slave data *)
END_IF;
```

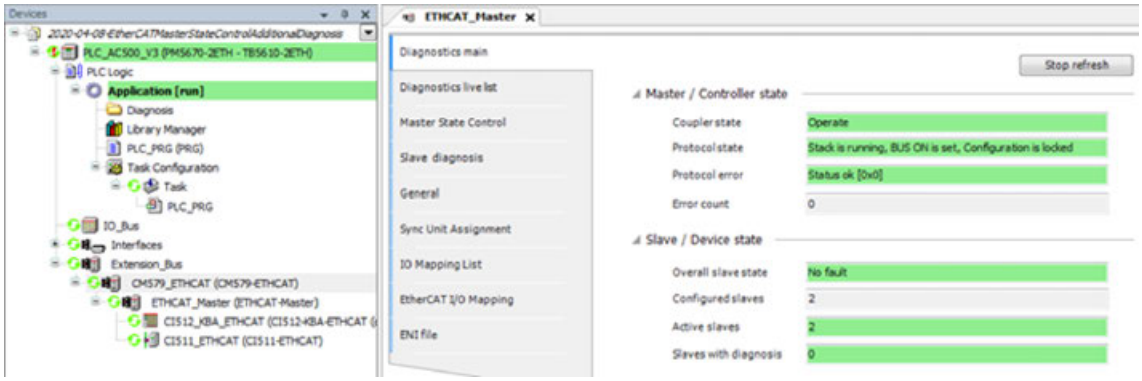
See Chapter 1.6.6.2.17.1.2 “Tab 'EtherCAT Master - Sync Unit Assignment' ” on page 3818.

EtherCAT diagnosis (V2 PLC and V3 PLC)

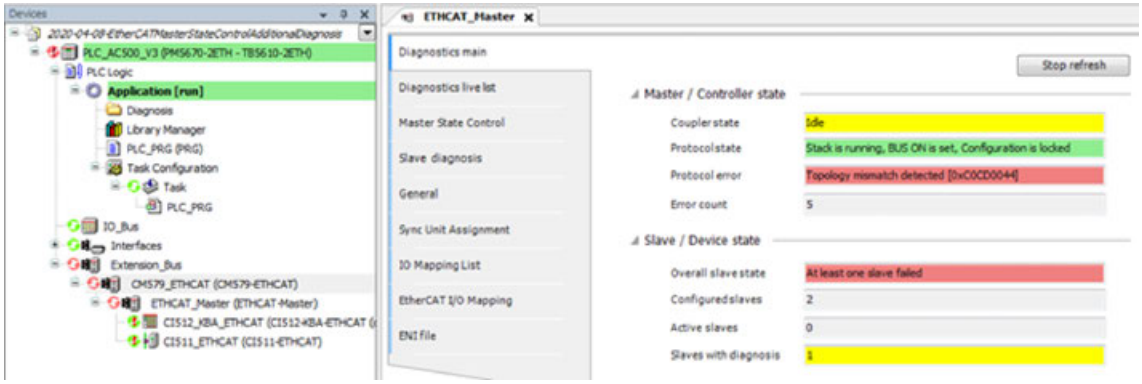
Automation Builder 2.3 provides an enhanced diagnosis interface for the EtherCAT fieldbus. The user can get EtherCAT diagnosis information from different editor views. All these views are accessible within the EtherCAT master device editor and provide information about the master and all configured or connected slaves. The main diagnosis overview is given in the EtherCAT master view *“Diagnostics main”*.

The *application example* shows how to integrate and use the function blocks to receive diagnosis messages in the CODESYS program of an AC500 V3 PLC.

“Diagnostics main” shows EtherCAT state *“Operate”*.



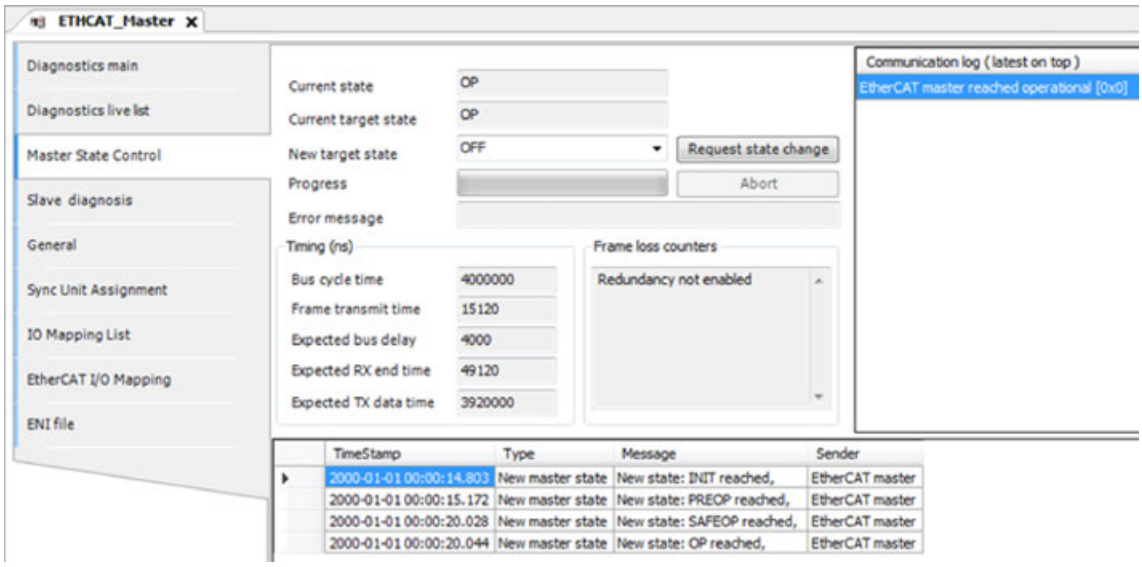
“Diagnostics main” shows EtherCAT state *“Topology error”*.



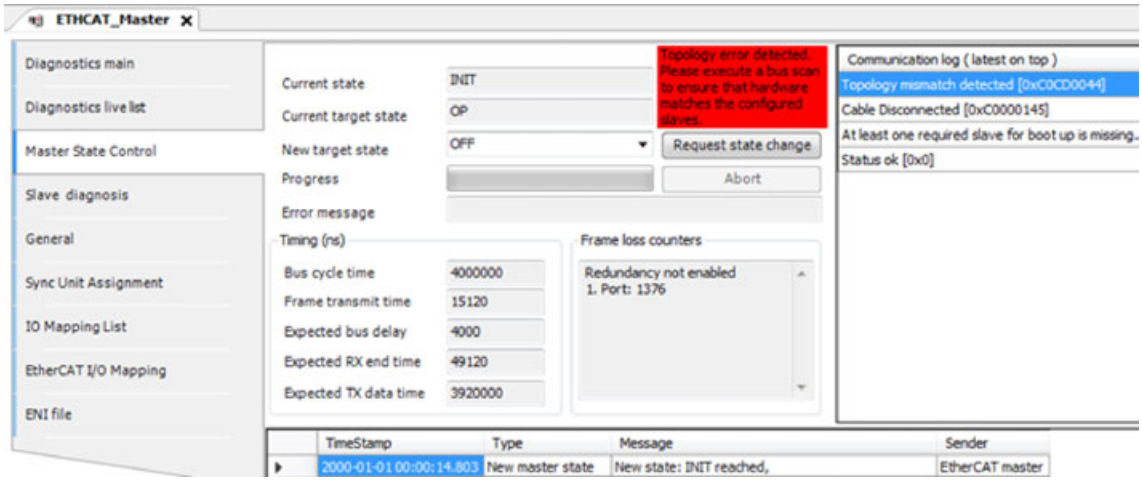
If the EtherCAT bus state shows *“Operate”*, the user does not need to check for any more information.

If the *“Diagnostics main”* shows any error, like *“Topology mismatch detected”*, the user can continue to the next level of information by opening editor view *“Master State Control”*.

“Master State Control” shows EtherCAT state *“Operate”*.

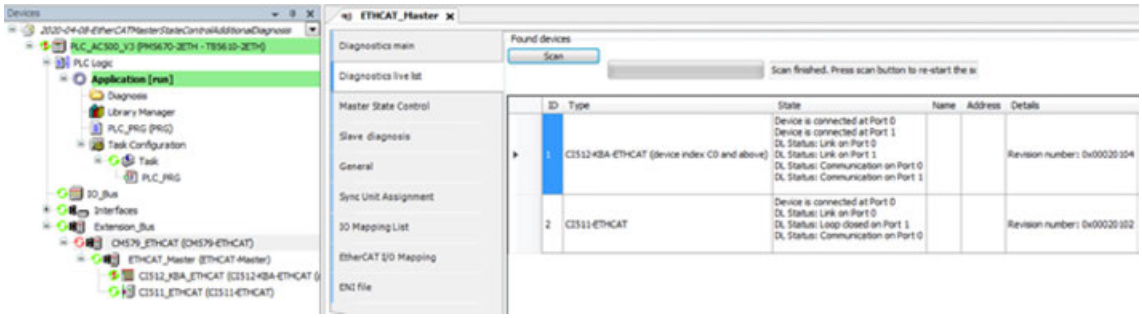


“Master State Control” shows EtherCAT state *“Toplogy error”*.



In editor view *“Master State Control”* the user can request a master state change or get information about configured parameters as well as events and latest communication errors. In case of any topology error (e.g. slaves are configured in a different order than they exist in hardware) the Automation Builder shows a hint to the user that it might be helpful to execute a bus scan in editor view *“Diagnostics live list”* to compare the scan result of the real hardware with the configures slaves in the Automation Builder project.

Bus scan result in editor view *“Diagnostics live list”* shows the connected hardware.



The bus scan result list shows the following information for each connected slave:

ID		Position of the found slave device
Type		Slave identification (name or Vendor/Device ID number)
State		The connection/link state of all ports (0-3) of the given slave
	Connected	=> Cable is plugged in
	+ Link	=> Physically connected to another slave
	+ Communication	=> Communication works fine
Name		Not used for EtherCAT
Address		Not used for EtherCAT
Details		E.g. revision number of the slave device

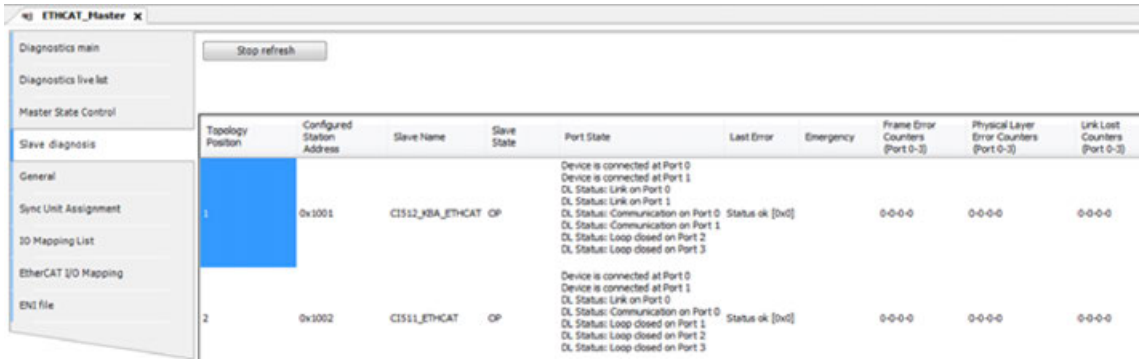
The bus scan shows information about the real connected hardware.

Note that a bus scan will always restart the EtherCAT bus.

This should not be a problem during commissioning but it might not be applicable in a running system.

For runtime diagnosis the Automation Builder provides cyclic information of all configured slaves and their states in the editor view *"Slave diagnosis"*.

Slave diagnosis information shows that configured slaves are ok.



"Slave diagnosis" view shows wrong slave at position 1.

The editor view *"Slave diagnosis"* shows information about the configured slaves. If these slaves are found in hardware, the states of all slaves are ok. If there is a mismatch between hardware and configuration the view shows at which position that mismatch was detected.

The *“Slave diagnosis”* shows the following information for each configured slave:

Topology Position		Position of the configured slave device.
Configured Station Address		Address that is defined by configuration. This address is not topology dependent.
Slave Name		Configured name of the slave device.
Slave State		The state of the slave. Possible slave states are:
	`NOT CONNECTED` [1])	
	`INIT`	
	`PREOP`	
	`SAFEOP`	
	`OP`	
	`INIT ERR`	
	`PREOP ERR`	
	`SAFEOP ERR`	
Port State [2])		The state of all ports (0-3) of the given slave.
		Shows how many connections this slave has to other slaves and if the connections are working fine:
	Connected	=> Cable is plugged in
	+ Link	=> Physically connected to another slave
	+ Communication	=> Communication works fine
Last Error		The last error that occurred in this slave. As text, if available, and error number. If this is any topology error, the editor view will show a hint to perform a bus scan.
Emergency [CAN application protocol over EtherCAT (CoE)]		This column contains up to 5 CoE emergency entries. Each entry has
	Error code	
	Address of the error register	
	Error data (1 byte)	
		If there are more than five emergencies reported by the slave, the columns show a hint that some emergency entries have been lost. The column is empty, if no CoE emergencies exist.
Frame Error Counters [2])		Counts transmission errors on frame layer, detected by CRC check of frames. Fast growing values show a serious problem. Possible root causes include damaged cables, high electromagnetic noise or misbehavior of EtherCAT slave devices. Four counter values are shown, one for each port 0-3. Column has red background in case of any value other than 0.
Physical Layer Error Counters [2])		Counts transmission errors on physical layer. Possible root causes include electromagnetic disturbance or faulty devices. Four counter values are shown, one for each port 0-3. Column has red background in case of any value other than 0.

Link Lost Counters [2])	Optional feature of EtherCAT slave devices, not supported by every device.
	Counts loss of physical connection (no link, LED off). Even short interruptions can be detected. Possible root causes include power dips, device reset, poor cables or connectors, loose contact. Four counter values are shown, one for each port 0-3. Column has red background in case of any value other than 0.

[1]) Note that columns *"Port State"*, *"Frame Error Counters"*, *"Physical Layer Error Counters"* and *"Link Lost Counters"* show *"LLD: Timeout"*, if this state is `NOT CONNECTED`, because this information is not accessible.

[2]) Note that this column contains *"LLD: Timeout"*, if slave state is `NOT CONNECTED`.

General note on the counters

Please note that this kind of errors will be detected by devices when power state changes, e.g. when the device itself or a neighboring device is powered on, caused by switching artifacts on the cable. This does not signal an issue, only counters increasing during normal operation should trigger deeper analysis. Counters can be reset by the PLC program using corresponding function blocks.

PROFIBUS

The fieldbus PROFIBUS is supported in AC500 PLC as master and slave. The communication modules *"CM592-DP PROFIBUS DP V0/V1 master module"* and *"CM582-DP PROFIBUS DP slave module"* are provided for these purposes.

Parameterization of the CM592-DP/CM582-DP communication modules

Configuration is valid as of CPU FW 3.5.0.

To append a communication module, add the communication module to the *"Extension_Bus"* node.

- Right-click the desired slot and select *"Add object"*.
- Select the communication module from the list and click *[Replace object]*.

- Double-click the new node to open the CM592-DP/CM582-DP PROFIBUS DP configuration in the editor window. Click on tab *"CM592-DP/CM582-DP Parameters"* if not already opened.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the PROFIBUS communication module.

Click on tab *"CM592-DP/CM582-DP I/O Mapping"* to open the *"Bus Cycle Options"* in the editor window.

The following parameter is available:

Parameter	Default value	Value	Description
Bus cycle task	Use parent bus cycle settings	Use parent bus cycle settings	Settings from PLC settings tab are used.
		Task	Name of the task that triggers the bus cycle

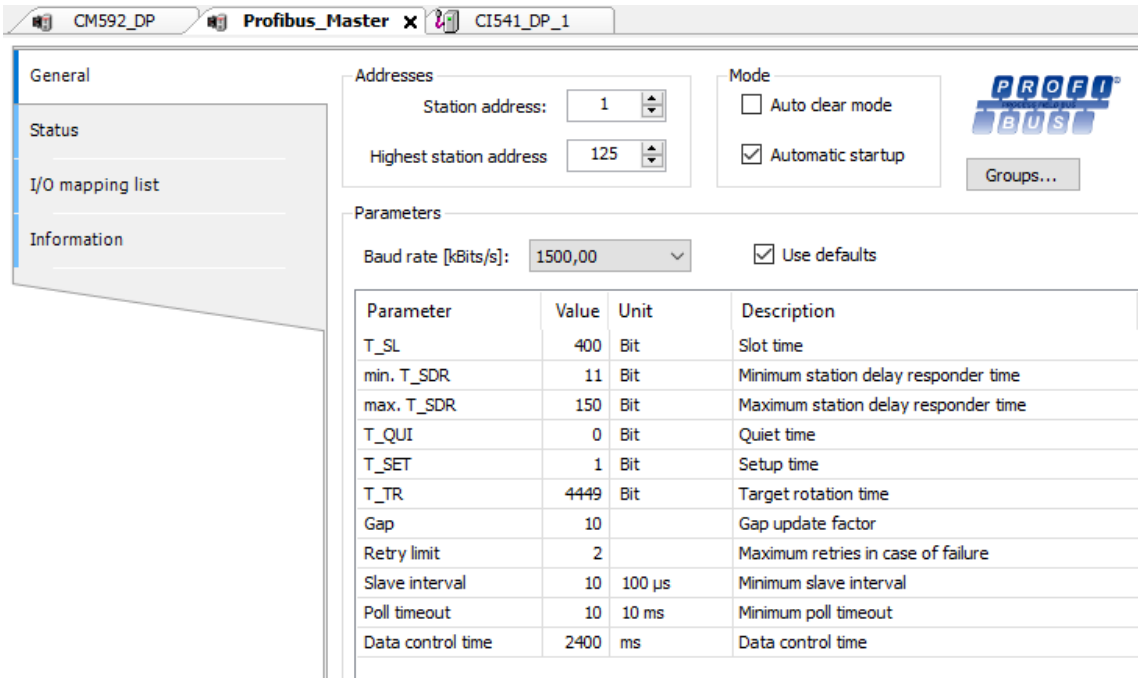
Click on tab CM592-DP/CM582-DP IEC-Objects to open the list of used IEC Objects. For information instantiated I/O driver function block class is shown.

CM592-DP PROFIBUS DP master communication module

Configuration of a PROFIBUS DP master

Double-click on *"Profibus_Master_x (Profibus_Master)"* to open the *"Profibus_Master"* configuration in the editor window:

Click on tab *"General"* if not already opend.



Most of the parameters are calculated automatically. Uncheck *[Use defaults]* to enable values to be edited individually. Checking *[Use defaults]* again will revert all parameters to default values.

All times for the PROFIBUS parameters are given in bit time [tBit]. The bit time is the result of the reciprocal of the transmission rate:

tBit = 1 / transmission rate in [bit/s]

The conversion from milliseconds into a bit time is shown in following formula:

*tBit = Time in [ms] * transmission rate in [bit/s]*

The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1)
Adresses				
Station address	1	0...125	The individual device address of the master device on the bus.	DpParameter -> Station address
Highest station address	126	0...126	The highest bus address up to which a master searches for another master at the bus in order to pass on the token. This station address must on no account be smaller than the master station address.	DpParameter -> Highest station address
Mode				
Auto-Clear mode	Enabled	Disabled	The master operation mode will stay in the mode 'Operate' and the communication to all available slaves is kept up.	AutoClear-Supported
		Enabled	The masters operation mode will change from 'Operate' to 'Clear' and it shuts down the communication to all assigned slaves, if at least 1 slave is not responding within the data control time.	
Automatic startup	Enable	Disable	Do not perform automatic startup	AutoStart
		Enable	Perform automatic startup	
Parameters				
Baud rate	1500	9.6 19.2 45.45 93.75 187.5 500 1500 3000 6000 12000	Data transfer speed in [kBits/s]. The baud rate must be set to the same value for all devices on the bus. The result of changing the baud rate is that all other parameters must be recalculated.	DpParameter -> Baudrate
T_SL Slot time	300	37.. 65535	Monitoring time of the sender (requester) of a telegram for the acknowledgement of the recipient (responder). After expiration, a retry occurs in accordance with the value of maximum telegram retries.	DpParameter -> TSL

Parameter	Default	Value	Description	Parameter (Remark 1)
min. T_SDR Minimum station delay responder time	11	1...65535	Shortest time period that must elapsed before a remote recipient (responder) may send an acknowledgement of a received query telegram. The shortest time period between reception of the last bit of a telegram to the sending of the first bit of a following telegram.	DpParameter -> min. TSDR
max. T_SDR Maximum station delay responder time	150	1...65535	Longest time period that must elapse before a sender (requestor) may send a further query telegram. Greatest time period between reception of the last bit of a telegram to the sending of the first bit of a following telegram. The sender (requestor, master) must wait at least for this time period after the sending of an unacknowledged telegram (e.g. broadcast only) before a new telegram is sent.	DpParameter -> max. TSDR
T_QUI Quiet time	0	0...127	Time delay that occurs for modulators (modulator-trip time) and repeaters (repeater-switch time) for the change over from sending to receiving.	DpParameter -> TQUI
T_SET Setup time	1	0...255	Minimum period reaction time between the receipt of an acknowledgement to the sending of a new query telegram (reaction) by the sender (requestor).	DpParameter -> TSET
T_TR Target rotation time	11894	1.. 2 -1 (=16777215)	Pre-set nominal token cycle time within the sender authorization (token). The available time for the master to send data telegrams to the slaves depends on the difference between the nominal and the actual token cycle time. The Target rotation time (TTR) is shown in Bit times [tBit] like the other bus parameters. Below the displayed bit time, the Target rotation time is also displayed in [ms]. The default value depends on the number of slaves attached to the master and their module configuration.	DpParameter -> TTR

Parameter	Default	Value	Description	Parameter (Remark 1)
Gap Gap update factor	10	0...255	Factor for determining after how many token cycles an added participant is accepted into the token ring. After expiry of the time period G*TTR, the station searches to see whether a further participant wishes to be accepted into the logical ring.	DpParameter -> Gap update factor
Retry limit Maximum retries	1	1...15	Maximum number of repeats in order to reach a station.	DpParameter -> max. retry limit
Data control time	120	1..2^{24}-1	Defines the time in [ms] within the Data_Transfer_List is updated at least once. After the expiration of this period, the master (class 1) reports its operating condition automatically via the Global_Control command. The default value depends on the transmission rate.	DpParameter -> Data control time
Slave interval Minimum slave interval	2000	1...65535	Defines the minimum time period between two slave list cycles in [µs]. The maximum value the active stations require is always given. The default value depends on the slave types.	DpParameter -> min. slave interval
Poll timeout Minimum poll timeout	10		Sets the maximum period of time in [ms] during which the response has to be received.	DpParameter -> Poll timeout

Remark 1:

To display the parameters of this column, enable the option *“Show generic device configuration views”* under *“Tools ➔ Options ➔ Device editor”*.

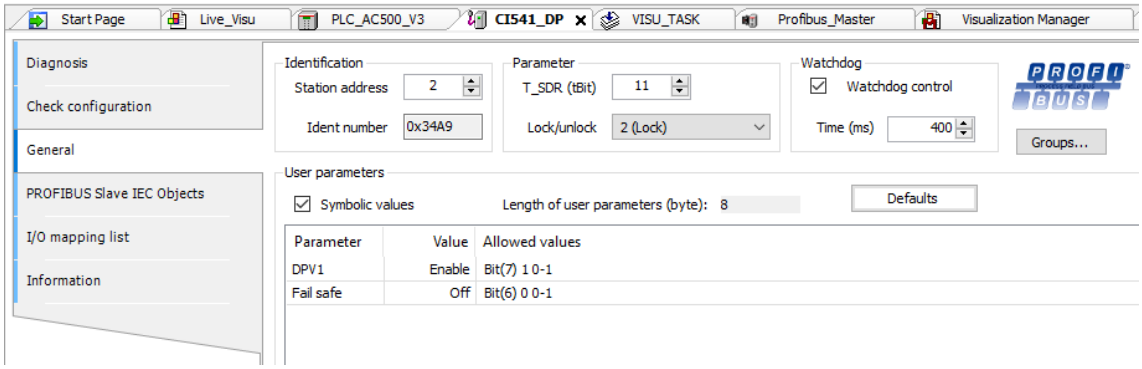
Configuration of a PROFIBUS DP slave

A PROFIBUS DP slave can be added by right-clicking on *“Profibus_Master_x (Profibus_Master)”* and selecting *“Add object”*.

If the desired device is not listed it can be installed via the *“Device Repository”* (menu item *“Tools” -> “Device Repository”*).

The slave configuration parameters can be edited in slave related editor window. To open this editor window, double-click the corresponding slave in the device tree.

Click on tab *“General”* if not already opened.



All times for the PROFIBUS parameters are given in bit time [tBit]. The bit time is the result of the reciprocal of the transmission rate:

tBit = 1 / transmission rate in [bit/s]

The conversion from milliseconds into a bit time is shown in following formula:

*tBit = Time in [ms] * transmission rate in [bit/s]*

The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1)
Identification				
Station address	1	0...126	Station address of the PROFIBUS DP slave device.	StationAddress
Ident number	GSD file specific	----	Ident number of the PROFIBUS DP slave device.	SlavePrmData -> identNumber
Parameter				
T_SDR (tBit)	11	11...255	The parameter T_SDR (tBit) represents the minimum station delay of a responder (time a responder waits before generating the reply frame).	SlavePrmData -> minTsdr
Lock/unlock	2 (Lock)	0 (T_SDR unlock)	The TSDR and slave-specific parameter may be overwritten.	Bit 6 = 0 and bit 7 = 0 of bitmask SlavePrmData -> stationStatus
		1 (Will be unlocked)	The slave is released to other masters.	Bit 6 = 1 and bit 7 = 0 of bitmask SlavePrmData -> stationStatus
		2 (Lock)	The slave is locked to other masters, all parameters are accepted.	Bit 6 = 0 and bit 7 = 1 of bitmask SlavePrmData -> stationStatus

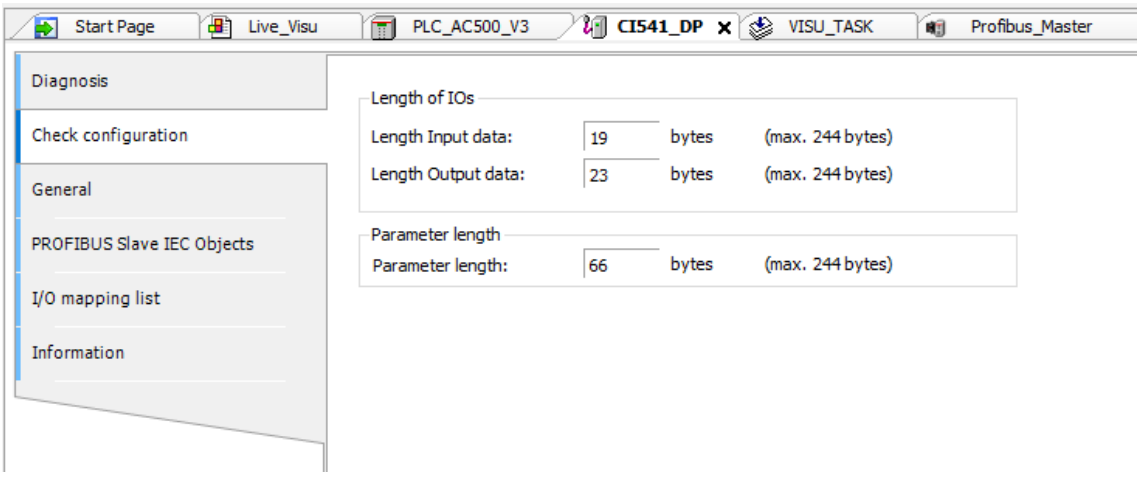
Parameter	Default	Value	Description	Parameter (Remark 1)
		3 (Unlock)	The slave is released to other masters.	Bit 6 = 1 and bit 7 = 1 of bit-mask Slave-PrmData -> stationStatus
Watchdog				
Watchdog control	Enabled	Disabled	The PROFIBUS Slave does not utilize the Watchdog Control Time setting.	SlavePrmData -> wdFact1
		Enabled	The PROFIBUS slave utilizes the Watchdog Control Time setting in order to detect communication errors to the assigned Master. When the Slave finds an interruption of an already operational communication, defined by a Watchdog time, then the Slave carries out an independent Reset and places the outputs into the secure condition.	
Time (ms)	400	0...2540	Watchdog time in [ms]. The default value depends on the number of slaves attached to the master and their configuration.	SlavePrmData -> wdFact2
User parameter				
Symbolic values	Enabled	Disabled	No symbolic names for the user parameters.	-
		Enabled	The values for the parameters are shown with symbolic names.	-
Length of user parameter (Byte)	3	Device-specific	The length of the user parameters in [bytes]. By default this value is 3 due to the existing reserved values.	-
Defaults	-	-	The button restores the default values of the user parameters.	-

Remark 1:

To display the parameters of this column, enable the option *“Show generic device configuration views”* under *“Tools ➔ Options ➔ Device editor”*.

Click on tab *“Check configuration”*.

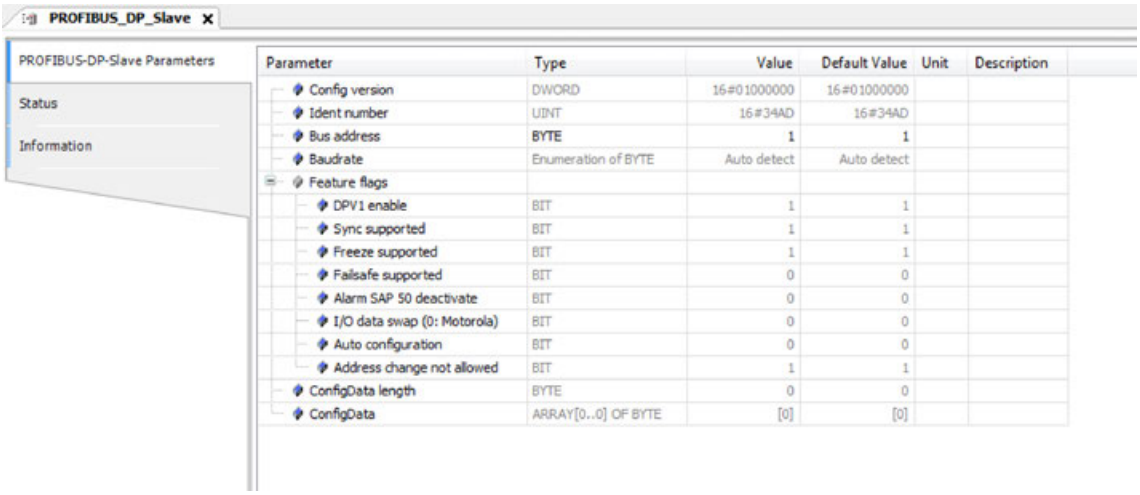
Following dialog shows values for input/output/parameter data occupied by your configuration. Here it can be checked how much data is left for further configuration.



CM582-DP PROFIBUS DP slave communication module

Configuration of PROFIBUS DP slave

Double-click on *"PROFIBUS_DP_Slave"* to open the PROFIBUS slave configuration in the editor window:



The following parameters can be modified:

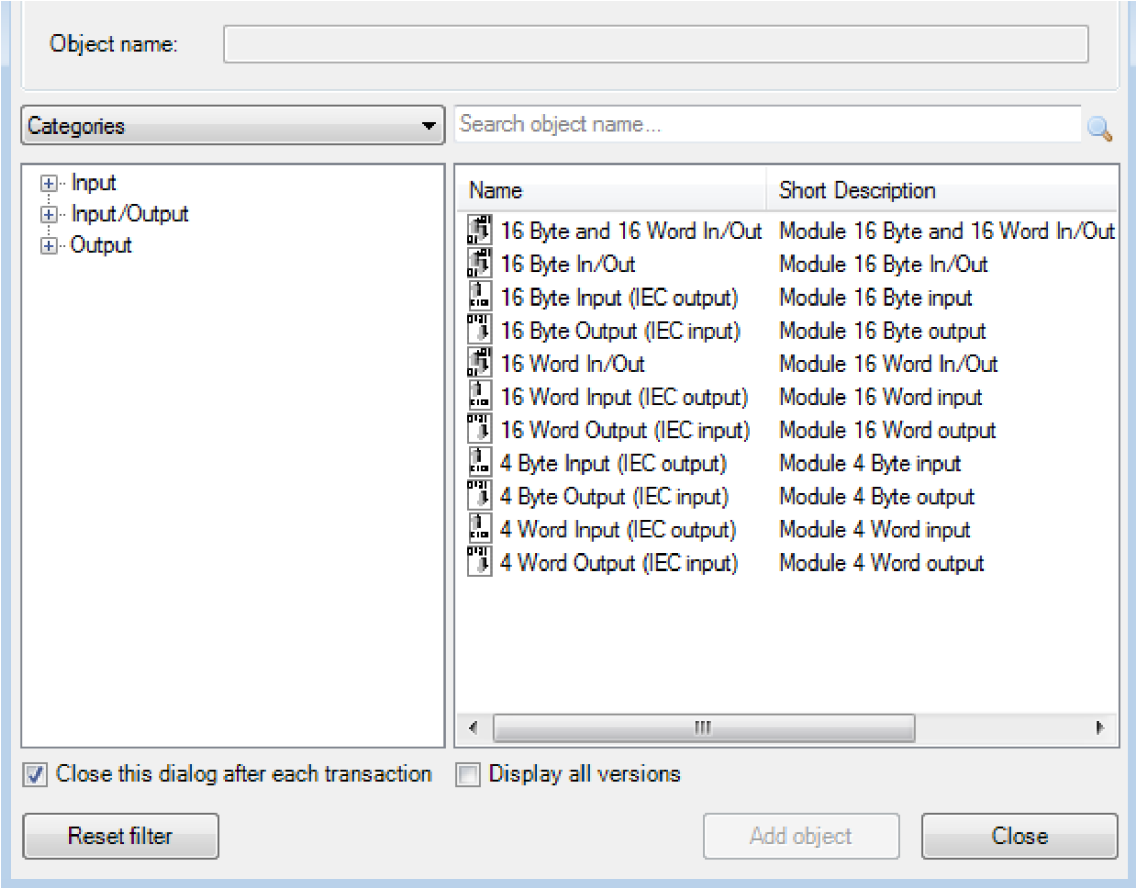
Parameter	Default value	Value	Description
Bus address	1	0...126	The bus address is the individual device address of the slave device on the bus.

Configuration of I/O data objects

To append I/O data, add the desired input / output objects to the Communication Module node.

Right-click the Communication Module node and select *"Add object"*.

Different types of data objects group I/O variables by size and direction. The I/O driver of the PLC firmware copies the amount of data bytes configured by these data objects cyclically.



Select the desired I/O objects from the list and click *[Add object]*.

> *To keep basic load of PLC low, only configure as much I/O data objects as actually required. If further I/O variables need to be added later, additional data objects can be inserted.*

Technical details on the device such as the maximum amount of bytes used for I/O data is described in the device specification for .

Double-click an added I/O object node to open the preset configuration. As the I/O objects do not need user configuration all parameters in the *"Parameters"* tab are read-only.

Open the *"I/O Mapping"* tab to configure the mapping configuration for the I/O object.

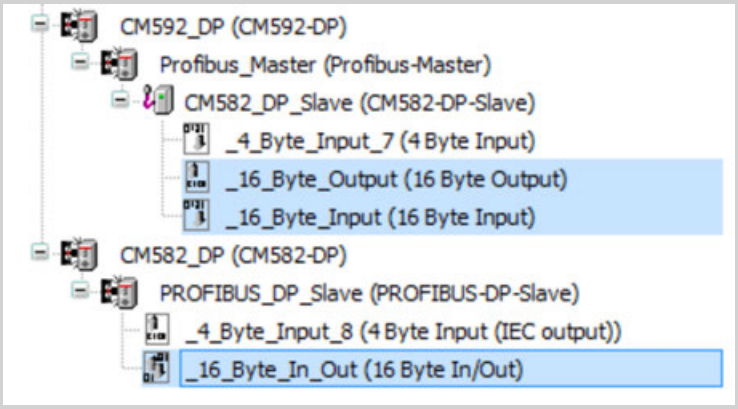
Possible inconsistency

On using CM582-DP slave device configured with modules types combining input and output data the following situation may happen:

Example

CM582-DP Communication Module configuration uses module type 16 Byte In/Out.

The device representation assigned to CM592-DP master uses module types 16 Byte Output and 16 Byte Input at the same place instead.



This mismatch will not be detected; neither by Automation Builder nor by PROFIBUS master and slave.

However, the communication will run stable and I/O data exchange is executed successfully.

Reason:

AC500 defines modules combining input and output directions to be split to two separated module configurations internally with output direction first.

Thus in AC500 the PROFIBUS configuration data for one module of type 16 Byte In/Out looks the same as for the combination of module types 16 Byte Output and 16 Byte Input.

Mapping of the I/Os

Double-click on the desired I/O data object in the device tree to show current I/O mapping connected to this data object.

See chapter Symbolic Names for Variables, Inputs and Outputs for further details on mapping inputs and outputs ↳ *Chapter 1.6.6.2.13.7 “Symbolic names for variables, inputs and outputs” on page 3776*.

1.6.6.2.12 Communication interface modules

Configuration of communication interface modules

Automation Builder can be used to configure the parameters of CI5xx devices.

> *Configuration of S500 I/O modules can be performed without CI5xx devices connected.*

Adding CI5xx device to the device tree

1. Right click in the device tree on the node *“Slot1”* or *“Slot2”* of the *“Extension_Bus”* and click *“Add object”*.

   ⇨ The window *Replace object : Slot* <...> opens.

2. Select your *CM5xx master module* and click *[Add object]*.
   ⇨ The *CM5xx master* appears in the Slot.
3. Right click on the *CM5xx master module* and click *“Add object”*.
   ⇨ The window *Add object below : <...>_Master* opens.
4. Select your *“CI5xx”* device and click *[Add object]*.
   ⇨ The *“CI5xx”* device appears in your device tree.

Adding S500 I/O modules

1. Right click on your *“CI5xx”* device and click *[Add object]*.
   ⇨ The window *Add object below:* opens.
2. Select your I/O module and click *[Add object]*.
   ⇨ The I/O module is added.

Configure parameters

▷ Double-click the *“CI5xx”* device to open editors and select the *“CI5xx_IO Parameters”* tab.

This editor shows the parameters that can be set for each device. For more information see ↳ *Chapter 1.6.3.7 “Communication interface modules (S500)” on page 3043*, and ↳ *Chapter 1.6.3.6 “I/O modules” on page 2569*.

CI521-MODTCP/CI522-MODTCP

Unbundled CI52x-MODTCP configuration

Automation Builder can be used to configure the parameters of CI52x-MODTCP devices.

A direct Ethernet connection is required between the PC running Automation Builder and the CI52x-MODTCP module.

Configuration of S500 I/O modules can be performed without CI52x-MODTCPs modules connected.

Start a project from template

1. Select *“New Project”* in menue item *“File”*.
   ⇨ The window *“New Project”* appears.
2. Select the *“CI52x-MODTCP Configuration Project”* and click *“OK”*.
   ⇨ The window *“Select PLC”* opens.
3. Select a *“CI52x-MODTCP”* device and click *“Add device”*.
   ⇨ A project is created. More modules can be added.

Add CI52x-MODTCP to a project

1. Right click in the device tree on the root of the *"Project"* and click *"Add object"*.
   ⇨ The window *"Add object below"* opens.
2. Select *"Modbus devices"* and click *"Add object"*.
   ⇨ The node*"Modbus_devices"* appears in your device tree.
3. Right click on the node *"Modbus_devices"* and click *"Add object"*.
   ⇨ The window *"Select PLC"* opens.
4. Select your *"CI52x-MODTCP"* device and click *"Add device"*.
   ⇨ The *"CI52x-MODTCP"* device appears in your device tree.

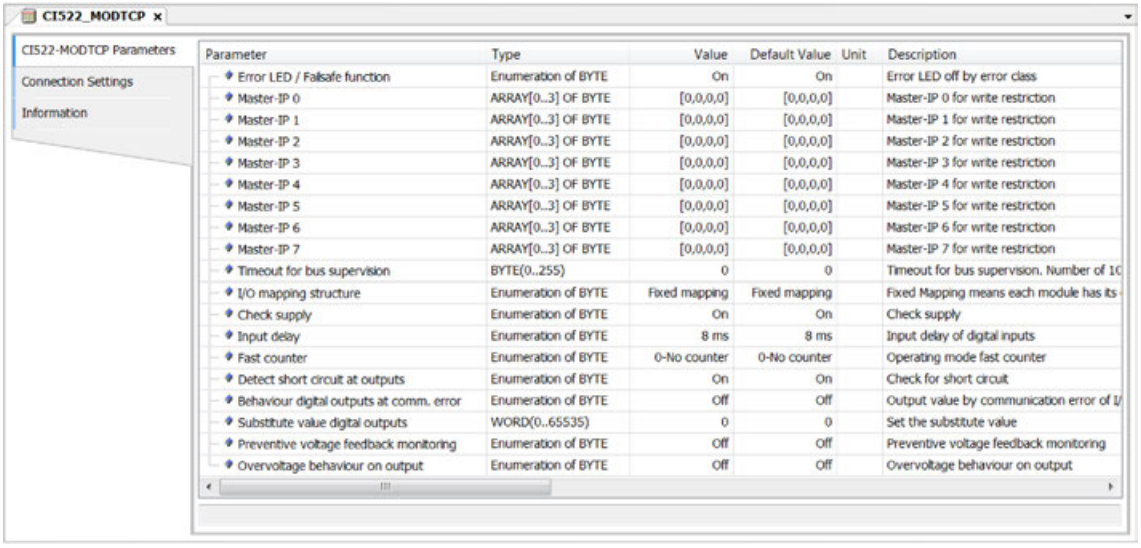
Add S500 I/O modules

1. *"Add object"* to your *"CI52x-MODTCP"* device.
   ⇨ The window *"Add object below: CI52x-MODTCP"* opens.
2. Select your I/O module and click *"Add object"*.
   ⇨ The I/O module is added.

Configure parameters

▷ Double-click the device to open editors and select the *"CI52x-MODTCP Parameters"* tab.

This editor shows the parameters that can be set for each device. For more information see ↳ *Chapter 1.6.3.7.4.1.7.1 "Parameters of the module" on page 3176* CI521, ↳ *Chapter 1.6.3.7.4.2.7.1 "Parameters of the module" on page 3206* CI522 and ↳ *Chapter 1.6.3.6 "I/O modules" on page 2569*.



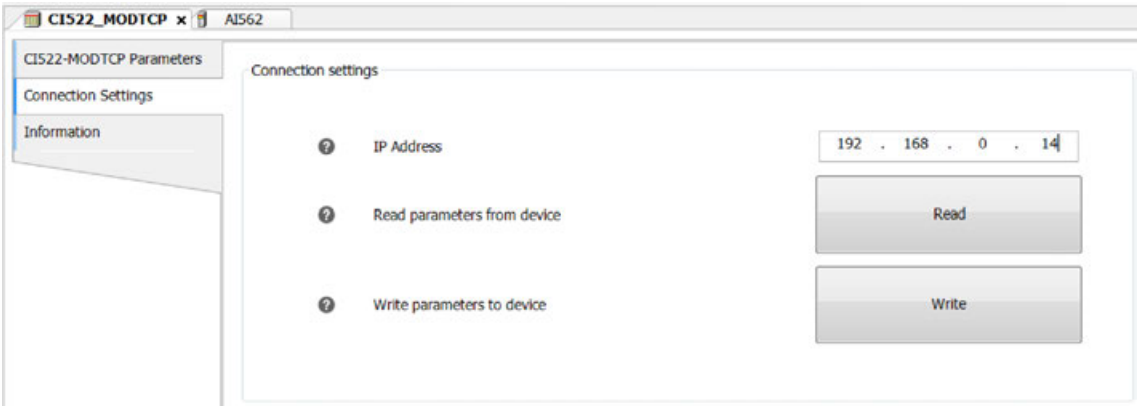
Parameter	Type	Value	Default Value	Unit	Description
Error LED / Failsafe function	Enumeration of BYTE	On	On		Error LED off by error class
Master-IP 0	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 0 for write restriction
Master-IP 1	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 1 for write restriction
Master-IP 2	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 2 for write restriction
Master-IP 3	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 3 for write restriction
Master-IP 4	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 4 for write restriction
Master-IP 5	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 5 for write restriction
Master-IP 6	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 6 for write restriction
Master-IP 7	ARRAY[0..3] OF BYTE	[0,0,0,0]	[0,0,0,0]		Master-IP 7 for write restriction
Timeout for bus supervision	BYTE(0..255)	0	0		Timeout for bus supervision. Number of 10
I/O mapping structure	Enumeration of BYTE	Fixed mapping	Fixed mapping		Fixed Mapping means each module has its
Check supply	Enumeration of BYTE	On	On		Check supply
Input delay	Enumeration of BYTE	8 ms	8 ms		Input delay of digital inputs
Fast counter	Enumeration of BYTE	0-No counter	0-No counter		Operating mode fast counter
Detect short circuit at outputs	Enumeration of BYTE	On	On		Check for short circuit
Behaviour digital outputs at comm. error	Enumeration of BYTE	Off	Off		Output value by communication error of I/
Substitute value digital outputs	WORD(0..65535)	0	0		Set the substitute value
Preventive voltage feedback monitoring	Enumeration of BYTE	Off	Off		Preventive voltage feedback monitoring
Overvoltage behaviour on output	Enumeration of BYTE	Off	Off		Overvoltage behaviour on output

Connect to device

To read or write parameters, the CI52x-MODTCP module must be connected to the PC with an Ethernet connection.

See ↳ *Chapter 1.6.3.7.4.1.5 "Addressing" on page 3175* CI521 and ↳ *Chapter 1.6.3.7.4.2.5 "Addressing" on page 3206* CI522 of the CI52x-MODTCP hardware documentation for information on configuring the IP address of the device.

On the CI52x-MODTCP device editor, the *"Connection Settings"* tab allows the IP address of the device to be entered.



Read Reads the parameters from the CI52x-MODTCP and also for the attached S500 I/O modules.

Write Sends the parameters from the editors to the CI52x-MODTCP and also the S500 I/O modules.

Device checking

The CI52x-MODTCP module knows which I/O modules are attached.

While reading and writing parameters, the project must match the physical hardware. Otherwise an error will be given.

Communication errors will also result in error messages.

When the parameters have been read or written correctly, a message is seen in the *"All messages"* window:

Attached S500 modules

It is possible to read and write parameters when the S500 I/O modules are not attached to the CI52x-MODTCP module.

> *To perform a read, the project structure must still match the configuration of CI52x-MODTCP.*

A warning will be shown if an I/O module is not detected:

When writing parameters, the CI52x-MODTCP configuration is overwritten so the current configuration of missing (unplugged) modules does not matter.

If the I/O modules are attached, then the project must match the hardware, otherwise an error will be given.

Firmware update

As of Automation Builder 2.2.1, the IP Configuration Tool can be used to perform firmware updates for CI52x-MODTCP devices.

1.6.6.2.13 I/O bus and I/O modules

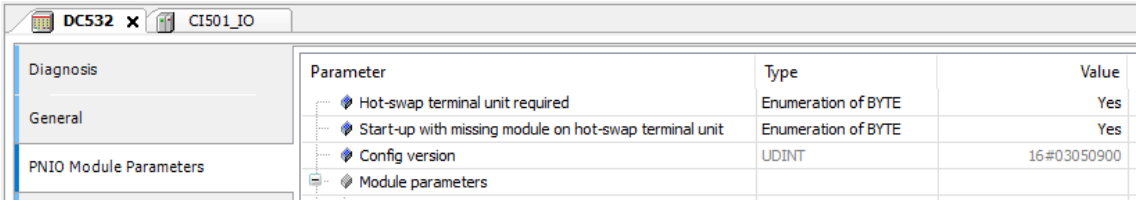
Hot swap configuration

Parameter configuration

I/O extension modules include the below parameters for hot swap configuration

Parameter	Purpose	Value
Hot-swap terminal unit required	To include diagnosis for missing hot-swap terminal unit	Yes: Communication Interface provides extended diagnosis for missing hot-swap terminal unit
		No (default): Extended diagnosis not available
Start-up with missing module on hot-swap terminal unit	Ignore missing module during start-up on hot-swap terminal unit. Incomplete I/O configurations must not prevent the system from starting.	Yes: Module is optional, start-up if there is no module available on hot-swap terminal unit
		No (default): Module is mandatory, start-up only if correct module is available

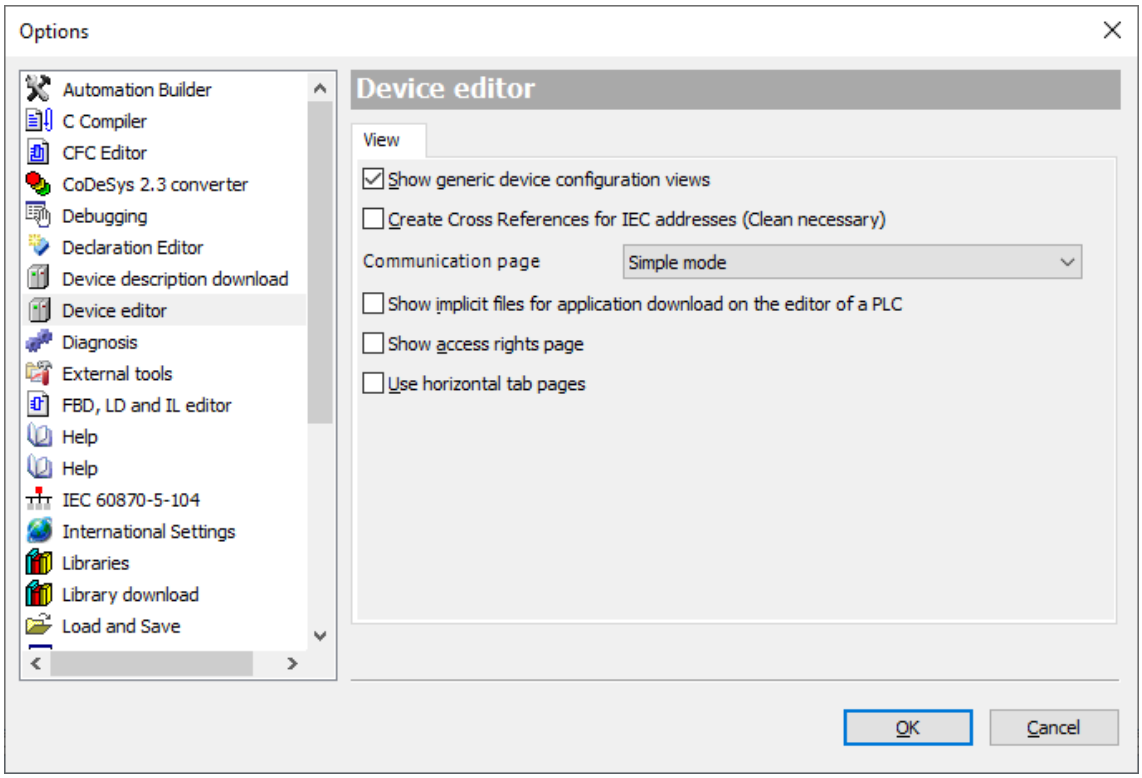
In the Automation Builder projects for V3 PLCs, hot-swap parameters can be configured from Module Parameters tab of respective I/O module.



By default, the Module Parameters tab is not visible for parameter configuration.

Follow the below steps to enable this tab

1. In the Automation Builder menu select *“Tools ➔ Options”*
2. Select *“Device editor”* option from *“Options”* dialog.
3. Enable the option *“Show generic device configuration views”* (if not already done)
4. To save the settings and close the dialog select *“OK”*



Parameterization of the I/O bus

Double-click the *"IO_Bus"* node in the device tree to open the I/O bus configuration.

The following parameters are available:

Parameter	Default	Value	Description
Run on config fault	No	No	In case of configuration fault the user program will not be launched.
		Yes	The user program will be also launched in case of configuration error on the I/O Bus.
Max wait run	3000	0...120000	Maximum waiting time for valid inputs.

In case of a digital I/O Module, the channels are provided as WORD, BYTE and BOOL. Because the analog inputs can also be configured as digital inputs, bit 0 of each channel is also available as BOOL.

The symbolic name of a channel can be entered in front of the string "AT" in the channel declaration.

All channels should have a symbolic name and only symbolic names should be used in the program code. If the hardware configuration has changed or if you want to download the project to a PLC with another hardware configuration and thus the PLC configuration has to be changed, the addresses of the inputs and outputs can change. In case of symbolic programming (i.e., symbolic names are used), the program code does not have to be changed.

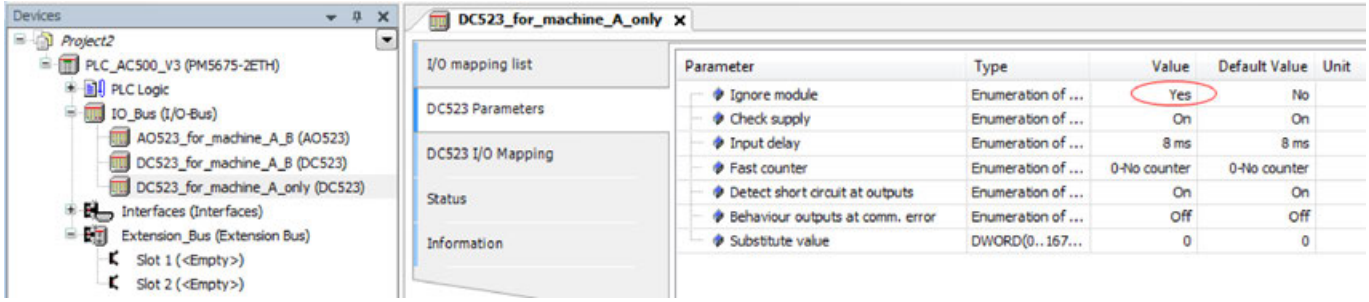
Parameter 'Ignore module'

All I/O devices provide the parameter "Ignore module". This parameter can be used for simulation purposes and determines whether an I/O device is considered or ignored during a PLC configuration check.

This allows to use an existing Automation Builder project/PLC configuration though some hardware devices are not physically available in a hardware installation.

Example

The Automation Builder project for machine A shall be used for machine B. However, the second DC523 device is missing in the hardware installation of machine B. Hence, for machine B the value for 'Ignore module' is set to 'YES'.



I/O bus - Bus cycle task

General information

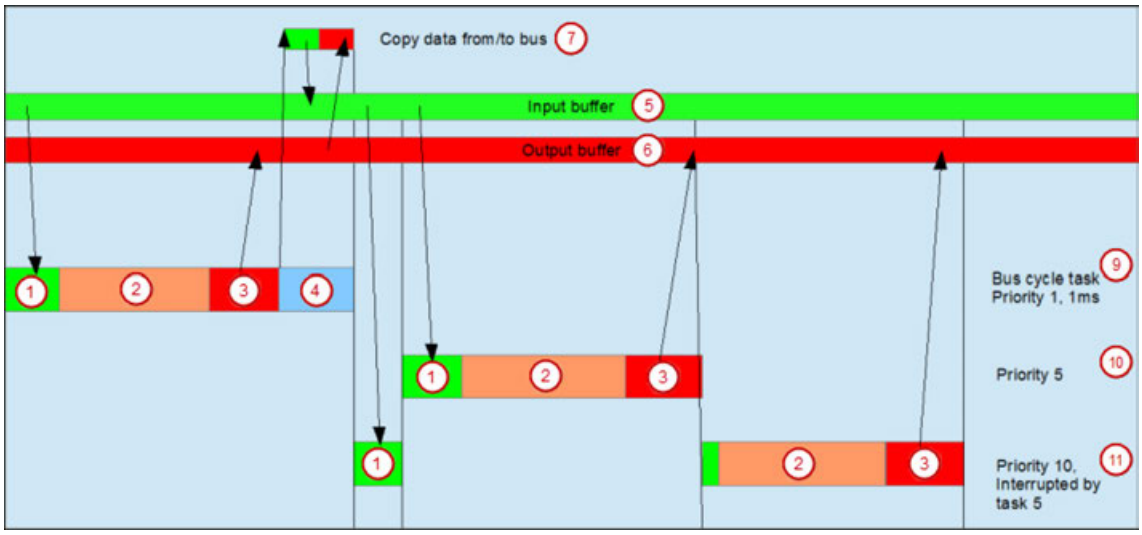
By "bus" it means all fieldbuses including I/O bus. There is no bus cycle task for Modbus because it is controlled by POUs. Modbus does not provide IO mapping.

It's recommended to define a dedicated bus cycle task for each fieldbus configured in the project. It's strongly recommended not to use "unspecified" in the "*"PLC Settings"*" to avoid unexpected behavior. The task defined in *"PLC Settings"* determines the bus cycle task of I/O bus and, depending on the configuration, of the additional fieldbuses (the setting is by default inherited).

Especially in case of EtherCAT, a dedicated bus cycle task should be used which is not shared with other fieldbuses. If *[unspecified]* is set in *"PLC Settings"*, the EtherCAT task might be automatically used by other fieldbuses, potentially causing EtherCAT task processing to fail. This should be avoided by specifying a task different to the EtherCAT task in *"PLC Settings"*.

As a rule, for each IEC task the used input data is read at the start of each task and the written output data is transferred to the I/O driver at the end of the task . The implementation in the I/O driver is decisive for further transfer of the I/O data. The implementation is therefore responsible for the timeframe and the specific time when the actual transmission occurs on the respective bus system.

Other tasks copy only the I/O data from an internal buffer that is exchanged only with the physical hardware in the bus cycle task.



(1) Read inputs from input buffer
(2) IEC task
(3) Write outputs to output buffer
(4) Bus cycle
(5) Input buffer
(6) Output buffer
(7) Copy data to/from bus
(9) Bus cycle task, priority 1, 1 ms
(10) Bus cycle task, priority 5
(11) Bus cycle task, priority 10, interrupted by task 5

Using tasks

The *"Task Deployment"* provides an overview of used I/O channels, the set bus cycle task, and the usage of channels.

WARNING!

If an output is written in various tasks, then the status is undefined, as this can be overwritten in each case.

When the same inputs are used in various tasks, the input could change when a task is processed. This happens if the task is interrupted by a task with a higher priority and causes the process map to be read again. Solution: At the beginning of the IEC task, copy the input variables to variables and then work only with the local variables in the rest of the code.

Conclusion: Using the same inputs and outputs in several tasks does not make any sense and can lead to unexpected reactions in some cases.

Insertion of S500 I/O devices

1. Right-click *"IO_Bus"* in device tree and select *[Add object]*.

   -> The Add Device dialog window where all available S500 I/O Devices are listed will open.

2. Append the S500 I/O Devices in the same order as they are mounted on the hardware.

   Input and output modules connected to the I/O bus occupy the I/O following area: %IB0 .. %IB999 or %QB0 .. %QB999.

AC500 (Standard): PM56xx support up to 10 S500 I/O Devices.

Configuring the input and output modules and channels

The I/O channel configuration depends on the corresponding S500 I/O Device. See hardware documentation of the I/O Device for more information.

The individual configuration parameters can be opened in the editor window via double-click on the corresponding module and are listed in tab [S500 I/O device name] Configuration.

Symbolic names for variables, inputs and outputs

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

Input and output mapping

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

The corresponding variable declarations are automatically available in the project.

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

Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
b_Input_IB0		Digital inputs I0 - I7	%IB0	BYTE			
x_Input_IB0_IX0		Digital input I0	%IX0.0	BOOL			
x_Input_IB0_IX1		Digital input I1	%IX0.1	BOOL			
x_Input_IB0_IX2		Digital input I2	%IX0.2	BOOL			
x_Input_IB0_IX3		Digital input I3	%IX0.3	BOOL			
x_Input_IB0_IX4		Digital input I4	%IX0.4	BOOL			
x_Input_IB0_IX5		Digital input I5	%IX0.5	BOOL			
x_Input_IB0_IX6		Digital input I6	%IX0.6	BOOL			
x_Input_IB0_IX7		Digital input I7	%IX0.7	BOOL			
		Digital inputs I8 - I15	%IB1	BYTE			
		Digital inputs I16 - I23	%IB2	BYTE			
		Digital inputs I24 - I31	%IB3	BYTE			
Fast counter							

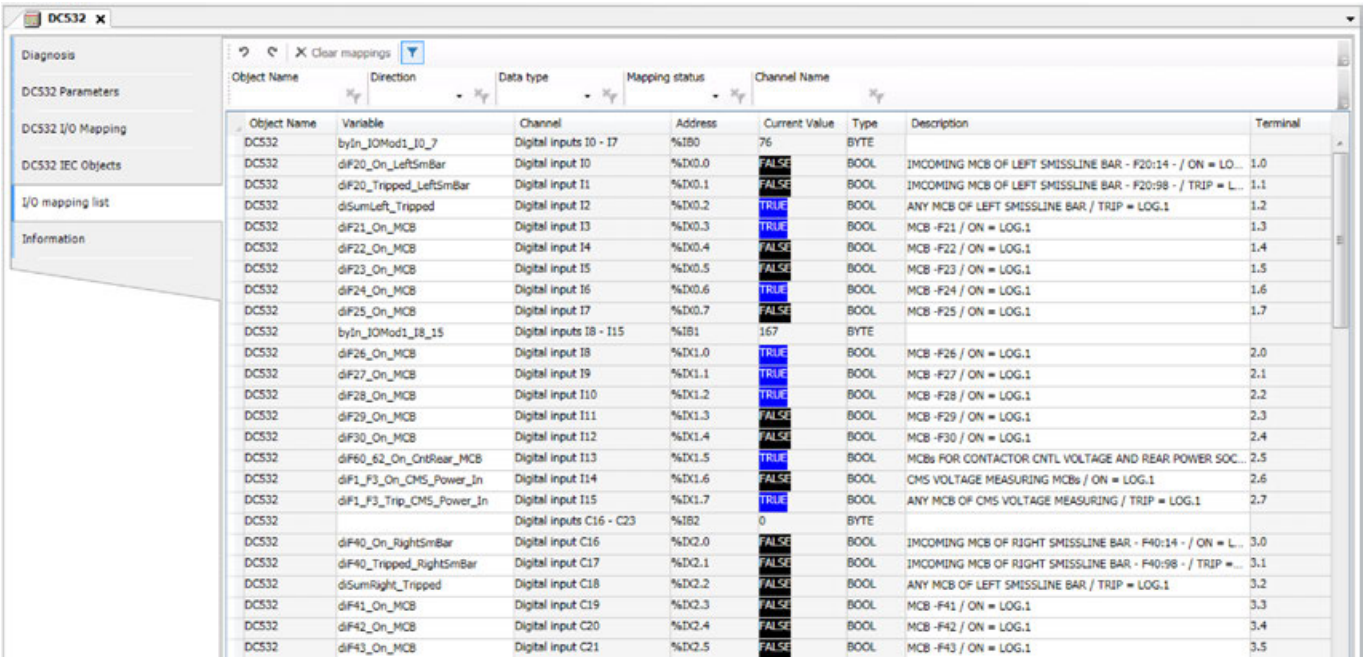
AC500 uses Intel Byte Order (Little Endian).

Only entries with a data type set in column "Type" can be mapped. These entries can be expanded to show the available I/O channels.

If the project has been imported from a previous Automation Builder version, all variables should be checked to avoid inconsistencies concerning the I/O mapping.

I/O mapping list

Automation Builder contains an I/O mapping list feature for creating mapping variables with better usability support compared to the tree structured view. Details on the tree structured view is provided in the CODESYS Development System *Chapter 1.4.1.7.1 “Configuring Devices and I/O Mapping” on page 213*.



Functionalities of the I/O mapping list:

- Displays I/O mappings for current node and all valid subsequent child nodes.
- Displays channel information with additional columns.
- Supports keyboard functions such as *cut*, *copy*, *paste*, *delete*, and *select all* within the editor and within Excel spreadsheet (for bulk editing).
- Contains a toolbar for various actions, e.g. filtering, undo/redo and clear mappings.
- Supports single click edit and easy navigation using arrow keys.
- Improvised error handling:
  - Allows to enter invalid mapping variables. This provides flexibility in bulk editing. Only when saving the project, the errors - according to IEC 61131 standard - are displayed.
  - In the message window, the error log is visible. The user can track the errors to their corresponding channel in the editor.
- Allows multi-selection of rows and columns. (Random selection is not allowed.)

Configuring I/O mapping list

Automation Builder supports tree and list based editors for creating I/O mapping variables.

1. From the **Tools** menu, select **Options**.
2. Under **Automation Builder**, select the **Editors** tab.
3. Choose your desired mapping dialog and click **OK**.
   - Choose **tree based** to display the I/O mapping in tree structure.
   - Choose **list based** to display the I/O mapping as list with the functionalities of the ToolBar.
   - Choose **both** to display both the tree structure (**I/O Mapping** tab) and the list view (**I/O mapping list** tab).

Available channel information

The I/O mapping list displays the channel information in offline and online mode. In online mode, all columns are read-only. In offline mode, some columns are editable.

The order of the devices in I/O mapping list is synchronized with the order in the device tree.

The channels of a device are ordered by the device description file. If channels have a section, the channel information is represented in a specific format.

Example: Fast counter: Actual value 1. These channels are listed at last position of a device.

Editing I/O mapping list

1. In the device tree, double-click **IO_Bus** to configure entire I/O mapping list of different I/O devices.
2. Enter the variables and descriptions to map the I/O devices.

> *Do not start variable names with a number or a special character. When saving the project, this generates an error. Example: 12input3, @input4.*

3. Click **Save Project** to save the I/O mapping changes.

Toolbar

Filtering

Especially in case of long I/O mapping lists, it might be helpful to filter the I/O mappings. For this, click the *"Filter"* icon to display all available criteria for filter options.

> *When reducing the width of the editor, some filters might be hidden.*

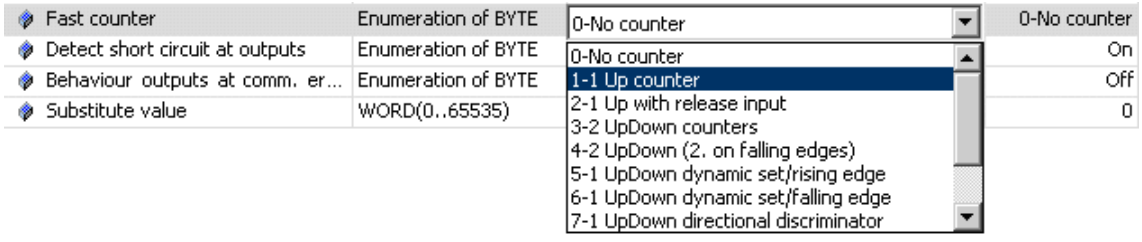
Undo, redo and clear

- **Undo**: Cancels the last change.
- **Redo**: Repeats the last change.
- **Clear mappings**: Deletes all variables and descriptions.

Fast counter

Configuration for S500 I/O modules

1. In the device tree, add a digital I/O module to the *"IO-Bus"* node.
2. Double-click the node for the I/O module, open the *"Parameters"* tab and set the counting mode *Chapter 1.6.6.2.13.9.2.1 "Counting modes" on page 3786* of the *"Fast counter"* parameter.



3. In the *"I/O Mapping"* tab channel configuration is displayed. ↳ *Chapter 1.6.6.2.13.9.3 "Control of the fast counter" on page 3790*

Operands

Table 656: Input information

Description of the input information	Output information of the user program	Description	
Start value 1	Output double word 0	Double word	Set values for the counters 1 and 2: Each counter can be set to a start value. Start values are loaded into the counter by the user program. Using the set signal (depending on the operating mode either via a terminal or the bit SET within the control byte 1 or 2), the values of the double word variables are loaded into the counter 1 or 2.
Start value 2	Output double word 1	Double word	
End value 1	Output double word 2	Double word	End value for the counters 1 and 2: The end values for the two counters are stored as comparison values into the module by the user program. Both counters compare continuously whether or not their programmed end value is equal to their actual value. When the counter (actual value) reaches its programmed end value, the binary output CF of the status byte is set permanently.
End value 2	Output double word 3	Double word	

Description of the input information	Output information of the user program	Description	
Control byte 1 see 1)	Output byte 0	Byte: Bit 0 = UP/DWN Bit 1 = EN Bit 2 = SET Bit 3 = CF_HW Bit 4 to Bit 7 free	Control bytes for the counter 1: **UP/DWN**: In some operating modes, the counter can count downwards, too. If counting down is desired, set the bit UP/DWN to TRUE and the bit SET to 1. When doing so, the counter starts counting downwards from the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. **EN**: Processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. **SET**: The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table. **CF_HW** 0 = state of CF is set to hardware channel (only for mode 1 and 2) 1 = normal output is set to hardware channel Bit 3 is evaluated only in control byte of counter 1.
Control byte 2 see 1)	Output byte 0	Byte: Bit 0 = UP/DWN Bit 1 = EN Bit 2 = SET Bit 3 to Bit 7 free	Control bytes for the counter 2: **UP/DWN**: In some operating modes, the counter can count downwards, too. If counting down is desired, set the bit UP/DWN to TRUE and the bit SET to 1. When doing so, the counter starts counting downwards from the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. **EN**: Processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte.

Description of the input information	Output information of the user program	Description	
			SET: The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table.

1) Only for CI581-CN/CI582-CN: Control bytes 1 and 2 are available twice on grounds of data consistency. Hence, a Start and End evaluation is only effected if the signals "Control Byte1_0" and "Control Byte1_1" or "Control Byte2_0" and "Control Byte2_1" (process image) are identical.

Table 657: Output information

Output information	Input information for the user program	Description	
Actual Value 0	Input double word 0	Double word	Actual value of the counter 0
Actual Value 1	Input double word 1	Double word	Actual value of the counter 1
Status Byte 0	Input byte 0	Byte: Bit 0 = CF Bit 1 to Bit 7 free	CF: When the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only when the counter is set again (set value), CF is reset to FALSE.
Status Byte 1	Input byte 1		

Operating modes

Inputs and outputs which are not used by the counters, are available for other tasks.

Legend:

- A refers to input channel A
- B refers to input channel B
- C refers to output channel C

Operating mode	Function	Used inputs and outputs	Notes
0	No counter	none	This operating mode is selected if the integrated fast counter is not necessary.
1	One count up counter	A = Counting input C = End value reached	The counting input and the output "End value reached) are enabled by the bit EN = TRUE within the control byte.

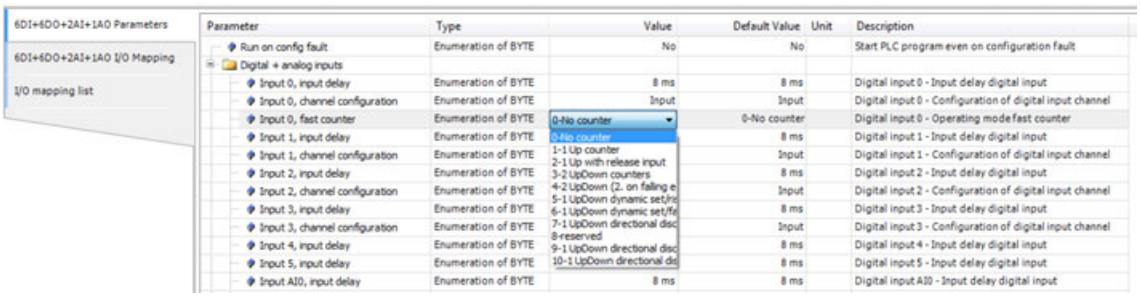
Operating mode	Function	Used inputs and outputs	Notes
2	One count up counter with enable input via terminal	A = Counting input B = Enable input C = End value reached	The enable input enables the counting input and the output "end value reached". The counter is only enabled if the enable input = TRUE (signal 1) AND the bit EN = TRUE within the control byte.
3	Two up/down counters	A = Counting input 0 B = Counting input 1	With this operating mode, two counters exist, which are independent of each other. The state "End value reached" is only readable from the two status bytes. It is not readable from output terminals. The counting direction is defined by the bit UP/DWN within the control byte.
4	Two up/down counters (1 counting input inverted)	A = Counting input 0 B = Counting input 1	This operating mode equals operating mode 3 with one exception: The counting input B (of counter 1) is inverted. It counts the TRUE/FALSE edges at input B.
5	One bidirectional counter with a dynamic set input via terminal	A = Counting input B = Dynamic set input	With this operating mode, one bidirectional counter is available which has a dynamic set input. Dynamic means that the set operation is performed at the FALSE/TRUE signal edge (0/1 edge) of the set input and not while the signal is TRUE. The state "End value reached" is only readable from the status byte, not from an output terminal.
6	One bidirectional counter with a dynamic set input via terminal	A = Counting input B = Dynamic set input	This operating mode equals operating mode 5 with one exception: The dynamic set input operates at the TRUE/FALSE edge (1-0 edge).

Operating mode	Function	Used inputs and outputs	Notes
7	One bidirectional counter for position sensors	A = Trace A of the position sensor B = Trace B of the position sensor	With this operating mode, incremental position sensors can be used which interchange their counting signals on tracks A and B in a 90° phase sequence. Depending on the sequence of the signals at A and B, the counter counts up or down. There is no pulse-multiplier function (e.g. x2 or x4). The position sensor must provide 24 V signals. Signals of 5 V sensors must be converted. Zero traces are not processed. The state "End value reached" is only readable from the state byte 0, not from an output terminal. The bit UP/DWN within the control byte must be FALSE. Otherwise, a parameter error occurrs. In this operating mode, the maximum counting frequency is: I/O modules 35 kHz. Communucation interface modules 50 kHz.
8	Reserved		

Operating mode	Function	Used inputs and outputs	Notes
9	One bidirectional counter for position sensors (pulse multiplier x2)	A = Trace A of the position sensor B = Trace B of the position sensor	This operating mode equals operating mode 7 with one exception: There is a pulse multiplication x2 with the evaluation of the counting inputs. This means, that the counter counts both the positive edges and the negative edges of trace A. This results in the double number of counting pulses. The precision increases correspondingly. In this operating mode, the maximum counting frequency is: I/O modules 30 kHz. Communucation interface modules 35 kHz.
10	One bidirectional counter for position sensors (pulse multiplier x4)	A = Trace A of the position sensor B = Trace B of the position sensor	This operating mode equals operating mode 7 with one exception: There is a pulse multiplication x4 with the evaluation of the counting inputs. This means that the counter counts the positive and negative edges of the traces A and B. This results in the fourfold number of counting pulses. The precision increases correspondingly. In this operating mode, the maximum counting frequency is: I/O modules 15 kHz. Communucation interface modules 20 kHz.

Configuration for onboard I/Os

1. In the device tree, double-click the *"Onboard I/O"* node (OBIO).
2. In the *"Parameters"* tab set the counting mode ↳ *Chapter 1.6.6.2.13.9.2.1 "Counting modes" on page 3786* for the fast counter.



3. In the *"I/O Mapping"* tab channel configuration is displayed. ↳ *Chapter 1.6.6.2.13.9.3 "Control of the fast counter" on page 3790*

Configuring the fast counter

The parameter of the fast counter channels of the Onboard I/O must be configured before they can be used. User should take these steps to configure the fast counter:

Channel	Direction	Width	Description
Actual value X	Input	DWORD	Current value of the fast counter.
State byte X	Input	BYTE	**Bit 0 = CF** If the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only, if the counter is set again (set value), CF is reset to FALSE. **Bit 1 to Bit 7 free**
Start value X	Output	DWORD	Each counter can be set to a start value. Start values are loaded into the counter by the user program. Using the set signal (dependent on the operating mode either via a terminal or the bit SET within the control byte X), the values of the double word variables are loaded into the counter X.
End value X	Output	DWORD	The end values for the two counters are stored as comparison values into the module by the user program. Both counters compare continuously, whether or not their programmed end value is equal to their actual value. If the counter (actual value) reaches its programmed end value, the binary output CF of the status byte is set permanently.

Channel	Direction	Width	Description
Control byte 1	Output	BYTE	**Bit 0 = UP/DWN** In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. When doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0, the counter jumps to 4,294,967,295. **Bit 1 = EN** The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. **Bit 2 = SET** The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table). **Bit 3 = CF_HW** 0 = state of CF is set to hardware channel (only for mode 1 and 2) 1 = normal output is set to hardware channel Bit 3 is evaluated only in control byte of counter 1. **Bit 4 to Bit 7 free**
Control byte 2	Output	BYTE	**Bit 0 = UP/DWN** In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. When doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0, the counter jumps to 4,294,967,295. **Bit 1 = EN** The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. **Bit 2 = SET** The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table). **Bit 3 to Bit 7 free**

Counting modes

The fast counter can be configured as one mode out of 10 possible modes. The desired operating mode is selected in the PLC configuration using configuration parameters. Inputs and outputs which are not used by the counter are available for other tasks. In the following table, A means input channel A, B means input channel B and C means output channel C.

CPUs	Integrated fast counter	Assigned inputs		Assigned Outputs	Remarks
		Channel A	Channel B	Channel C	
PM55x, PM56x	Yes	Input channel 0	Input channel 1	Output channel 0	Only 1 fast counter is available on the module. Input channel 0 is the default channel for fast counter. Input channel 1 can be used as another fast counter channel depending on fast counter mode.

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
0	No counter	None	Fast counter is disabled	-
1	1 count up counter	A = Counter input C = End value reached	Counting up A from 0 to 0xFFFFFFFF When the end value is reached, C will be set to high.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
2	1 count up counter with release input	A = Counter input B = Enable input C = End value reached	Counting up A from 0 to 0xFFFFFFFF The counter is enabled if B is high When the end value is reached, C will be set to high.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
3	2 Up/Down counters	A = Counter input 1 B = Counter input 2	2 independent counters. Status "End value reached" is only readable from the 2 status bytes, not from output terminals. The counting direction is defined by the Boolean parameters UD1 and UD2 of function block ONB_IO_CNT (Handle fast counter on Onboard I/O)	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
4	2 Up/Down counters (2nd on falling edges)	A = Counter input 1 B = Counter input 2	Same as operating mode 3, but counting input B is inverted (counts at TRUE/FALSE edges at input B).	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
5	1 Up/Down counter with dynamic set/ rising edge	A = Counter input B = Dynamic set input	1 Up/Down counter is available which counts on the rising edge of A and has a dynamic set input on B. Dynamic set input will set the start value at the rising edge of B.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
6	1 Up/Down counter with dynamic set/ falling edge	A = Counter input B = Dynamic set input	1 Up/Down counter is available which counts on the rising edge of A and has a dynamic set input on B. Dynamic set input will set the start value at the falling edge of B.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
7	1 UpDown directional discriminator	A = Phase A B = Phase B	With this mode, incremental encoders can be used which give their counting signals on phase A and B in a 90° phase sequence to each other. Dependent on the sequence of the signals at A and B, the counter counts up or down. There is no pulse multiplier function.	12 kHz (before firmware V2.0.6) 35 kHz (since firmware V2.0.6)
8	Reserved	-	-	-

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
9	1 UpDown directional discriminator X2	A = Phase A B = Phase B	This mode is the same as mode 7 with one exception: There is a pulse multiplication x2 with the evaluation of the counting inputs. This means that the counter counts both the positive edges and the negative edges of phase A. This results in the double number of counting pulses. The precision increases correspondingly.	11 kHz (before firmware V2.0.6) 30 kHz (since firmware V2.0.6)
10	1 UpDown directional discriminator X4	A = Phase A B = Phase B	This mode is the same as mode 7 with one exception: There is a pulse multiplication x4 with the evaluation of the counting inputs. This means that the counter counts both the positive edges and the negative edges of phase A and B. This results in the fourfold number of counting pulses. The precision increases correspondingly.	10 kHz (before firmware V2.0.6) 15 kHz (since firmware V2.0.6)

If channel 0 is configured as fast counter, the other channels 1,2 and 3 cannot be configured as interrupt inputs. Otherwise, a configuration error will appear and the CPU will be stopped.

Control of the fast counter

To control the fast counter configuration open the *“I/O Mapping”* tab.

The channels can be mapped as described in Symbolic Names for Variables, Inputs and Outputs and have the following meaning ↳ *Chapter 1.6.6.2.13.7 “Symbolic names for variables, inputs and outputs” on page 3776*:

Fast counter						
		Actual value 1	%ID17	DWORD		Actual value 1
		Actual value 2	%ID18	DWORD		Actual value 2
		State byte 1	%IB76	BYTE		State byte 1
		State byte 2	%IB77	BYTE		State byte 2
		Start value 1	%QD11	DWORD		Start value 1
		End value 1	%QD12	DWORD		End value 1
		Start value 2	%QD13	DWORD		Start value 2
		End value 2	%QD14	DWORD		End value 2
		Control byte 1	%QB60	BYTE		Control byte 1
		Control byte 2	%QB61	BYTE		Control byte 2

Channel	Direction	Width	Description
Actual value X	Input	DWORD	Current value of the fast counter
State byte X	Input	BYTE	Bit 0 = CF If the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only if the counter is set again (set value), CF is reset to FALSE. Bit 1 to Bit 7 free
Start value X	Output	DWORD	Each counter can be set to a start value. Start values are loaded into the counter by the user program. Using the set signal (dependent on the operating mode either via a terminal or the bit SET within the control byte X), the values of the double word variables are loaded into the counter X.
End value X	Output	DWORD	The end values for the 2 counters are stored as comparison values into the module by the user program. Both counters compare continuously whether or not their programmed end value is equal to their actual value. When the counter (actual value) reaches its programmed end value, the binary output CF of the status byte is set permanently.

Channel	Direction	Width	Description
Control byte 1	Output	BYTE	Bit 0 = UP/DWN In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. If doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. Bit 1 = EN The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. Bit 2 = SET The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table. CF = 0 Bit 3 = CF_HW 0 = state of CF is set to hardware channel (only for mode 1 and 2) 1 = normal output is set to hardware channel Bit 3 is evaluated only in control byte of counter 1. Bit 4 to Bit 7 free
Control byte 2	Output	BYTE	Bit 0 = UP/DWN In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. If doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. Bit 1 = EN The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte.

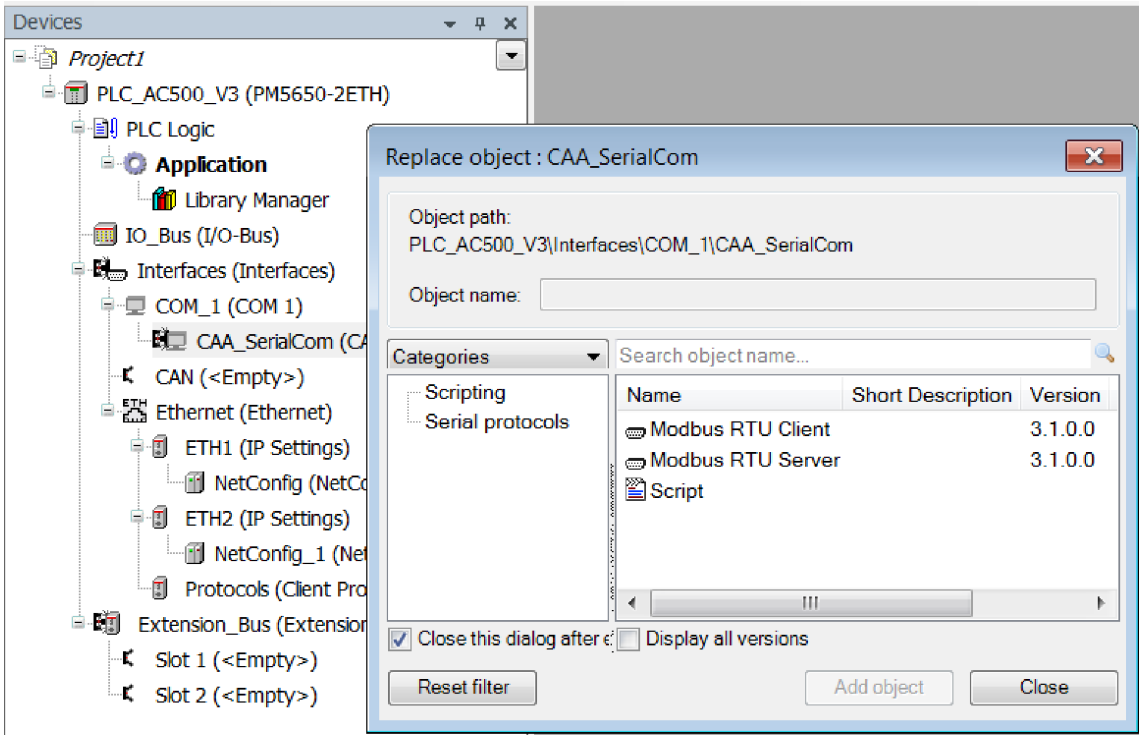
Channel	Direction	Width	Description
			Bit 2 = SET The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table. Bit 3 to Bit 7 free

1.6.6.2.14 Serial interface

Configuring Modbus RTU on serial interface

To enable Modbus RTU on a serial interface the protocol setup per default has to be replaced by either Modbus RTU Client or Server, depending on required operation mode.

A serial interface supports only one protocol/operation mode at once.



Replace object *"CAA_SerialCom"*

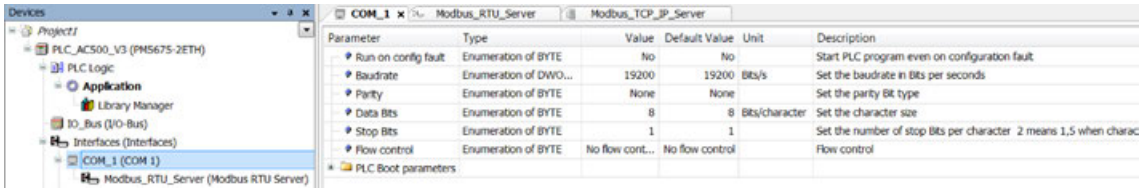
1. Right-click node *"CAA_SerialCom"* and click *"Add object"*.
2. Select *"Modbus RTU Client"* or *"Modbus RTU Server"* and click *"Add object"*.

   ⇨ *"CAA_SerialCom"* is replaced by your selection.

Parameters

Serial

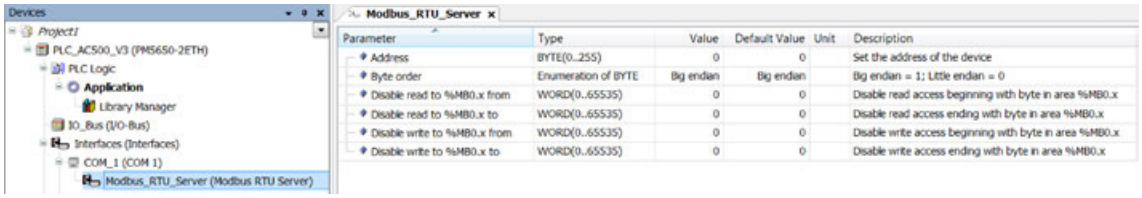
Serial parameters to be set selecting the interfaces node *"COM_1"*. They are common for both operating modes client and server.



> *The parameter "Data bits" always **has to be** set to "8" for Modbus.*

Modbus RTU server

Server specific parameters to be set selecting the protocol's node *"Modbus_RTU_Server"*.



Address

Bus address of the PLC as Modbus RTU Server on that interface

Byte Order

Format/Endianess for the transmission of WORD values (register) within the request/response telegram (default: Big Endian)

Disable

Parameter	Default	Value	Description
Disable write to %MB from	0	0 ... 65535	Disable write access starting at %MBx
Disable write to %MB to	0	0 ... 65535	Disable write access up to %MBx
Disable read from %MB from	0	0 ... 65535	Disable read access starting at %MBx
Disable read from %MBx to	0	0 ... 65535	Disable read access up at %MBx

It is possible to disable read and/or write access to individual segments. Reading/writing is disabled beginning at the set start address and is valid up to the set end address (inclusive).

Modbus RTU client

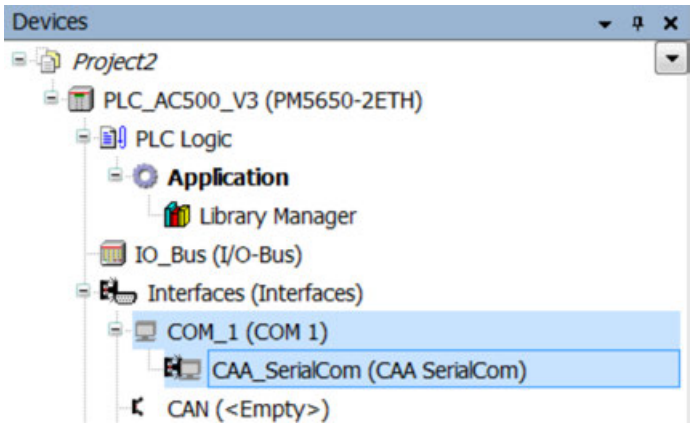
"Modbus RTU Client" does not have any protocol parameters.

Configuring CAA SerialCom on serial interface

The protocol *CAA SerialCom* represents the standard serial protocol provided by 3S and allows the users to implement their own custom protocol.

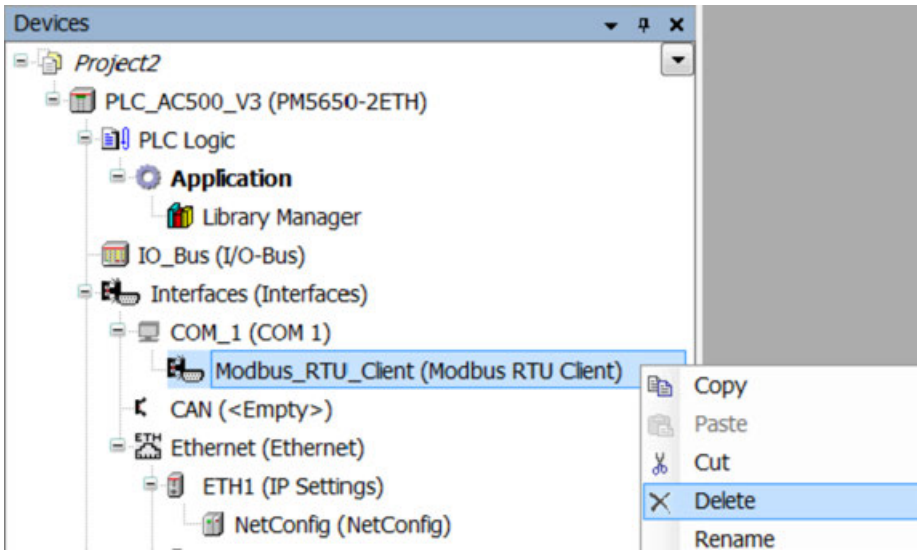
> *For details on CAA SerialCom, refer to standard 3S V3 documentation .*

Default setting



When creating a new project, the protocol *"CAA SerialCom"* is automatically attached to the *"COM_1"* port of a V3 PLC.

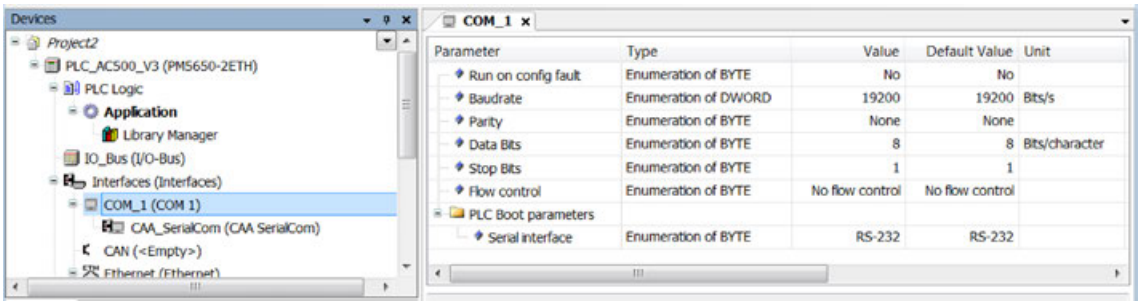
Switch to default setting



▷ Right-click on the node attached to *"COM_1"* node in the device tree and click *"Delete"*.

⇨ The node is switched back to the *"CAA SerialCom"* protocol.

Parameters



Since *CAA SerialCom* doesn't represent a "real" protocol, there are no specific parameters required. All common settings can be found at the Tab *"COM_1"* after double-click on the *"COM_1"* node (see also ↳ *Chapter 1.6.6.2.14.3 "Setting up a serial interface" on page 3798*).

Activate particular configuration parameters

The parameters set up in the Automation Builder device tree are NOT automatically taken over in the PLC.

It is still required to use the *3S IEC POUs* to activate the particular configuration parameters.

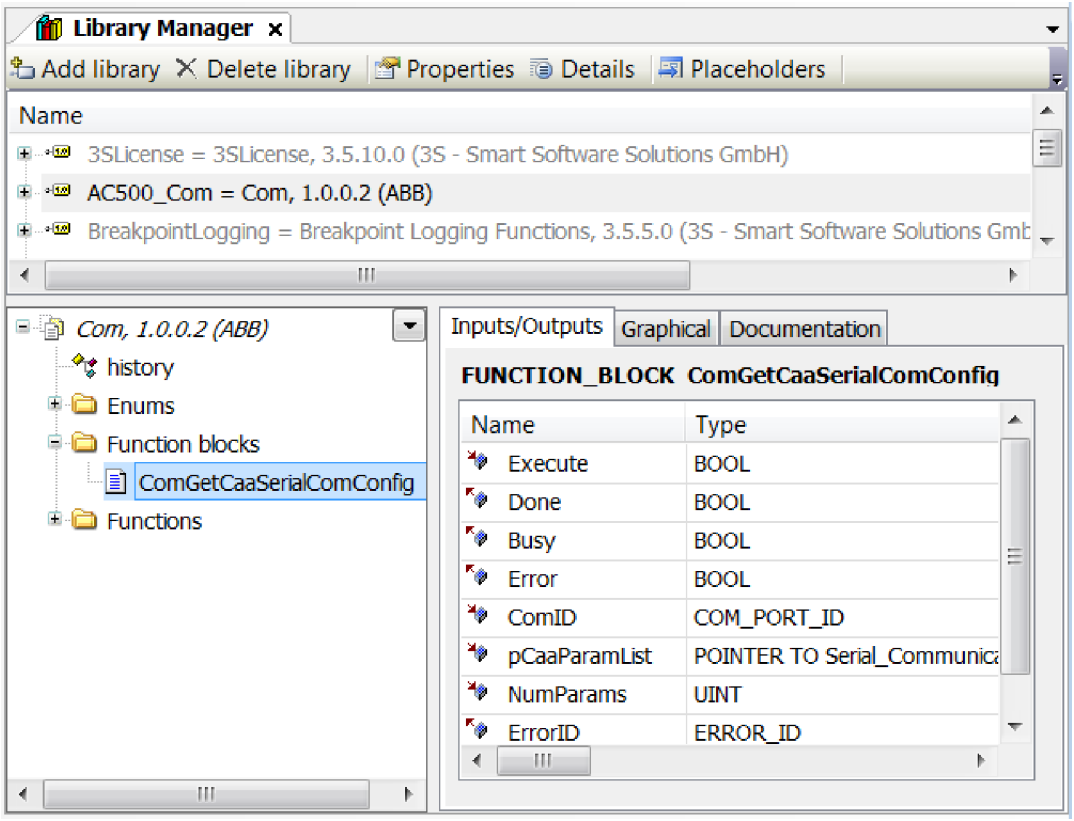


ABB provides the library *AC500_Com* (*"ABB - AC500 / Use Cases / Serial Communication)"* which contains a *POU* called *"ComGetCaaSerialComConfig"*.

The function block can be used to obtain the configuration data which is set up in Automation Builder to directly pass it to *CAA SerialCom-POU Open*. This avoids manual creation of a parameter list.

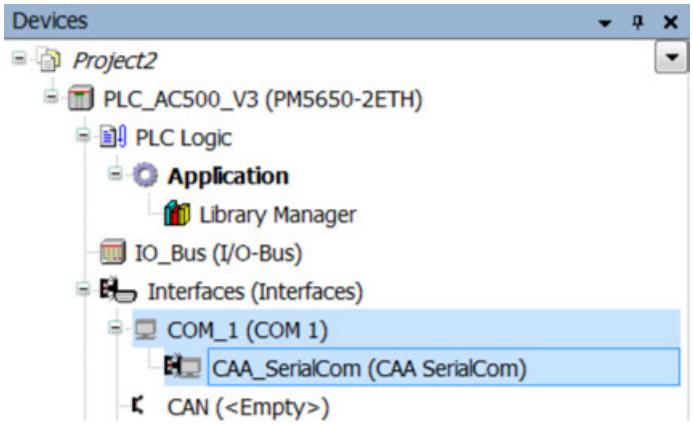
The following code snippet shows, how the COM port is identified by its node name and how the parameter list for the function block is read from the configuration data of the currently loaded IEC application:

```
FUNCTION_BLOCK GET_CAA_COM_CFG
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR

     ComGetCaaSerialComConfig:    ComGetCaaSerialComConfig;
        bExecGetCfg:              BOOL := FALSE;
        bDoneGetCfg:              BOOL := FALSE;
        bErrorGetCfg:             BOOL := FALSE;
        bBusyGetCfg:              BOOL := FALSE;
        ErrorIdGetCfg:          AC500_Com.ERROR_ID :=
AC500_Com.ERROR_ID.NO_ERROR;
        asParamList:              ARRAY[0..31] OF
AC500_Com.Serial_Communication.COM.PARAMETER;
        uiNumParams:              UINT := 32;

     uiStep:                    UINT := 0;
        szNodeName:               STRING(80) := 'COM1';
        ComID:                       AC500_Com.COM_PORT_ID;
        bSuccess:                 BOOL := FALSE;
        bError:                   BOOL := FALSE;
END_VAR
```

```
VAR CONSTANT
       STEP_INIT:                          UINT := 0;
       STEP_GET_ID:                        UINT := 1;
       STEP_FAILED_GET_ID:              UINT := NOT STEP_GET_ID;
       STEP_GET_CFG_CAA:                UINT := (STEP_GET_ID + 1);
       STEP_FAILED_GET_CFG_CAA:     UINT := NOT STEP_GET_CFG_CAA;
       STEP_DONE_SUCCESS:               UINT := (STEP_GET_CFG_CAA + 1);

END_VAR

IF uiStep = STEP_GET_ID THEN
        ComId := ComGetIdByName(szNodeName);
        IF ComId = AC500_Com.COM_PORT.COM_ID_INVALID THEN
                uiStep := STEP_FAILED_GET_ID;
        ELSE
                bExecGetCfg := TRUE;
                uiStep := STEP_GET_CFG_CAA;
        END_IF
END_IF

IF uiStep = STEP_GET_CFG_CAA THEN
        ComGetCaaSerialComConfig(
                 Execute:= bExecGetCfg,
                 Done=> bDoneGetCfg,
                 Busy=> bBusyGetCfg,
                 Error=> bErrorGetCfg,
                 ComID:= ComID,
                 pCaaParamList:= ADR(asParamList[0]),
                 NumParams:= uiNumParams,
                 ErrorID=> ErrorIdGetCfg);
        IF bDoneGetCfg THEN
                 uiStep := STEP_DONE_SUCCESS;
        ELSIF bErrorGetCfg THEN
                 uiStep := STEP_FAILED_GET_CFG_CAA;
        END_IF
END_IF

IF uiStep = STEP_DONE_SUCCESS THEN
         bSuccess := TRUE;
END_IF

IF uiStep = STEP_FAILED_GET_ID THEN
         bError := TRUE;
END_IF

IF uiStep = STEP_FAILED_GET_CFG_CAA THEN
         bError := TRUE;
END_IF
```

Setting up a serial interface

General



The configuration for serial interfaces and their protocols is done via two nodes:

- One node represents the common serial parameters related to the hardware port.
- The node below represents the parameterization for the particularly attached protocol.

Protocols supported by AC500 V3 PLCs

- 3S CAA SerialCom (common serial communication, send/receive data ↳ *Chapter 1.6.6.2.14.2 “Configuring CAA SerialCom on serial interface” on page 3794*)
- Modbus RTU (client & server ↳ *Chapter 1.6.6.2.14.1 “Configuring Modbus RTU on serial interface” on page 3793*)

How to switch between the protocols, see ↳ *“Default setting” on page 3795*.

Configuration

The following parameters are available in the configuration view of the COM port node:

Parameter	Value ranges	Description
Run on config fault	No	If this parameter is set to “Yes” the IEC application will not be prevented from switching to RUN state, independent from possibly existing configuration errors of the particular COM port.
	Yes	
Transmission rate	9600 baud/sec	Sets up the transmission rate to use for the COM port.
	19200	
	38400	
	57600	
	115200	
Parity	None	Sets up the parity to use for the COM port.
	Odd	
	Even	
Data bits	5 data bits	Sets up the number of data bits to use for the COM port
	6 data bits	
	7 data bits	
	8 data bits	
Stop bits	1 stop bits	Sets up the number of stop bits to use for the COM port.
	2 stop bits	

Parameter	Value ranges	Description
Flow control	No flow control	Allows to switch between different flow control modes (either RTS/CTS hardware or Xon/Xoff software or none). This setting is only valid for RS-232 serial interface mode. In case RS-485 is used for parameter “Serial interface”, flow control must set to “No flow control”. Otherwise a configuration error is triggered.
	Hardware RTS/CTS	
	Software Xon/Xoff	
Boot parameter Serial interface	RS-232	Allows to switch between RS-232 and RS-485. Due to technical reasons, it’s not possible to dynamically switch between the modes. This means, a reboot (or power cycle) of the PLC is required to activate the particular setting once changed.
	RS-485	

Comparison to V2

The following table shows the differences between V2 and V3 PLCs regarding the parameter set for serial interfaces:

V2 Parameter	Representation in V3	Remark
Run on config fault	Run on config fault	Exactly the same
RTS control	Flow control (partially)	Special modes which allow to use PLC as modem and mode implicitly setting RS-485 will not be taken over. Flow control settings will be limited and only support hardware, software or none.
Transmission rate	Transmission rate	For V3, the transmission rate range will be 9600 to 115200. Low modes will not be supported due to lack of support in Linux. High rates were only realized in V2 to support field bus plug as well as CS31 field bus. Both protocols are not supported anymore in V3, so these transmission rates won’t be available in V3. Approach: Only support most common transmission rates
Parity	Parity (subset)	A500 V3 doesn’t allow to configure parity modes “mark” and “space”. This means, only “none”, “odd” and “even” are configurable.
Data Bits	Data Bits	Exactly the same
Stop Bits	Stop Bits	Exactly the same

1.6.6.2.15 Gateway configuration

1. In the Automation Builder project, right-click the topmost PLC tree node and select *“Communication Settings”*.

   ⇨ The dialog window Communication Settings appears.

2. Click *“Advanced Settings”* to open the ↬ *Chapter 1.4.1.20.2.8.2 “Tab 'Communication Settings'” on page 840* dialog.

   ⇨ This information will be stored in the project file.

3. Click *“Gateway”* and select the desired action from the Gateway menu either to change the local gateway (see *Chapter 1.4.1.20.3.18.2 “Command 'Configure the Local Gateway'” on page 1125*) or to add a new gateway channel (see *Chapter 1.4.1.20.3.18.1 “Command 'Add New Gateway'” on page 1124*).

   Confirm your settings with *“OK”*.

Gateway settings on windows server 2012

Gateway as a service

To allow multiple concurrent users from different user sessions on the server to connect to PLCs, the user has to run CODESYS gateway as a system service. This is managed by a service called "CoDeSys V2.3 Gateway Service Wrapper". The service starts on system start-up and launch the gateway.

If you want to restart the gateway, use "Services management console" to restart "CoDeSys V2.3 Gateway Service Wrapper".

Gateway settings

You can set the communication settings in the Automation Builder project for every PLC. Otherwise, an error message is displayed while trying to open CODESYS.

See the description for *Chapter 1.6.6.2.15 “Gateway configuration” on page 3799* and select "TCP/IP" under *“Connection”*.

1.6.6.2.16 CAN onboard

CANopen

In Automation Builder, a CANopen network consists of one CANopen manager which acts as master device and optional CANopen remote devices which act as slave devices.

CANopen manager (master)

Tab 'CANopen Manager - General'

Table 658: “General”

“Node-ID”	The node number identifies the CANopen Manager as unique (range of values: 1...127).
“Check and Fix Configuration”	Opens the dialog of the same name. See below for details.
“Autostart CANopen Manager”	☑: The CANopen Manager starts automatically (switches to OPERATIONAL mode) after all required slaves are ready. ☐: The CANopen Manager has to be started from the application. The function block `CiA405.NMT` can be used to do this. Hint: As long as the CANopen Manager is not in OPERATIONAL mode, no PDOs are sent (outputs refreshed).
“Polling of optional slaves”	☑: When a slave does not respond during the boot sequence, the CANopen Manager interrogates it every second until it does respond. Constantly polling the slave increases the bus cycle time, which can interfere with the application (especially motion applications). You can deactivate polling to avoid this behavior. If polling is deactivated, then a slave is detected again when it sends a bootup message.
“Start slaves”	☑: The CANopen Manager is responsible for starting the slaves. ☐: You have to start the slaves from the application. Use the CiA405 NMT function block to do this.

"NMT start all (if possible)"	☑: If the *"Start slaves"* option is activated, then the CANopen Manager starts all slaves with an "NMT Start All" command. The "NMT Start All" command is not executed as long as optional slaves are not yet ready to be started. In this case, the CANopen Manager starts each slave individually. The "NMT Start All" command can be guaranteed only in a project without optional slaves.
"NMT error behavior"	• *"Restart slave"*. If an error occurs during slave monitoring (NMT Error Event), then the slave is restarted automatically by the stack (NMT Reset + SDO Configuration + NMT Start). • *"Stop slave"*. If an error occurs during slave monitoring (NMT Error Event), then the slave is stopped. Then you have to reset the slave from the application, using the CiA405 NMT function block.

Table 659: "Guarding"

Working with heartbeat messages is an alternative method of monitoring. It can be executed from both master and slave nodes, as opposed to node guarding. Normally the master sends heartbeat messages to the slaves.	
"Enable heartbeat producing"	The master sends heartbeats. They define the time interval in the *"Producer time"*. When the slaves are provided with the heartbeat function, a heartbeat consuming entry from the slave is created for the master. Then the Node-ID and the 1.5x heartbeat interval of the master are applied. ☐: Node guarding is enabled for the slaves. The settings from the EDS file of the slaves are used for this. If the values there cannot be used, then default values are used. Note that a CANopen Slave device can also be configured as a heartbeat producer.
"Node-ID"	Unique identification (1-127) of the heartbeat producer on the bus
"Producer time (ms)"	Interval length between successive heartbeats (in milliseconds)
"Redundancy Node-ID"	Requirement: A *"Redundancy Configuration"* object is inserted below the application. Unique identification (1-127) of the redundant heartbeat producer on the bus
"Redundancy wait time (µs)"	Requirement: A *"Redundancy Configuration"* object is inserted below the application. Duration of how long the passive controller waits for the heartbeat of the active controller. If this time is exceeded, then the passive controller takes on the active role.

Table 660: "SYNC"

"Enable SYNC producing"	☑: The CANopen manager sends SYNC telegrams (disabled by default) The synchronous PDOs are sent directly after the SYNC telegram.
"COB-ID (Hex)"	CAN-ID of the SYNC telegram. Range of possible values: [1...2047].
"Cycle period (µs)"	Interval length (in microseconds) after which the SYNC telegram is sent
"Window length (µs)"	Length of the time frame for synchronous PDOs (in microseconds)
"Enable SYNC consuming"	☑: (disabled by default). Another device must produce the SYNC telegrams that are received by the CANopen Manager.

NOTICE!
If SYNC producing is enabled for the CANopen manager, then you are not permitted to select the *"Enable SYNC producing"* option for all other bus devices.

Table 661: “TIME”

“Enable TIME producing”	☑: (disabled by default). The CANopen Manager sends TIME messages.
“COB-ID (Hex)”	(Communication Object Identifier): identifies the time stamp of the message. Default values: [0...2047], preset 16#100
“Producer time (ms)”:	Interval (in milliseconds) when the time stamp is sent. This value has to be a multiple of the task cycle time. Possible values [0, 65535]

The run time has to support high resolution timestamps. If not, then an error message is displayed.

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*
- *Chapter 1.6.6.2.16.1.2.1 “Tab 'CANopen Remote Device - General'” on page 3803*

Dialog 'Check and Fix Configuration'

If you insert several devices below the CANopen manager, then error messages may report multiple assigned Node-IDs or invalid COB-IDs. The *“Check and Fix Configuration”* button opens a dialog for solving these conflicts.

For conflicts with Node-IDs or PDO COB-IDs, you can click *“Edit Conflicts”* to open a dialog with detailed information.

Table 662: “Node-ID and COB-ID conflicts”

“Doubled node number”	List of all devices with identical IDs. In the field of the *“Node-ID”* column, you can enter new node numbers for the affected devices.
“Incorrect and double assignment of PDO COB-IDs”	The COB-IDs that are generated automatically from the device description files may not be permitted. All incorrect entries are listed with the respective device names, Node-IDs, and indexes. There are three options for correcting invalid COB-IDs: ● Correct the displayed formula for calculating the COB-IDs so that a valid COB-ID results. You can change the formula in the respective table element. ● Accept the automatic suggestion for the COB-ID by clicking the respective button. ● Accept all automatic suggestions by clicking the *“Use Suggested COB-ID”* button.

Corrected entries are removed from the displayed list automatically.

You can solve timing problems automatically by using the *“Automatic Repair”*. The command modifies all timing values to compatible values. (The time should be a multiple of the task time.)

CANopen remote device (Slave)

In CODESYS, a CANopen Remote Device is a slave device that you insert below a CANopen Manager in the device tree of a project. A distinction is made between modular and non-modular slaves:

- **Modular slaves**: You can insert CANopen modules (submodules) below a modular slave. These modules provide a *“I/O Mapping”* tab to map their inputs and outputs. Modular slaves can also have fixed I/Os. Then these devices also provide the *“I/O Mapping”* tab. Modular devices provide the *“Configure PDO mapping automatically”* option, which we recommend for standard applications. You find this option in the *“CANopen Remote Device”* dialog, on the *“General”* tab.
- **Non-modular slaves**: You cannot insert additional modules below a non-modular device. The inputs and outputs of these devices are mapped in the *“I/O Mapping”* dialog. Automatic mapping is not possible here.

See also

- *Chapter 1.6.6.2.16.1.2.1 “Tab 'CANopen Remote Device - General'” on page 3803*

Tab 'CANopen Remote Device - General'

The general settings of the CANopen Slave are defined in this dialog of a CANopen Remote Device (slave).

Table 663: “General”

“Node-ID”	The node number identifies the CANopen Remote Device uniquely. It corresponds to the number (value between 1 and 127) set on the device (hardware). You have to provide the Node-ID as a decimal.
“Expert settings”	☑: All settings are displayed that are predefined by the device description (EDS file) for the device.
“SDO channels (...)”	Click this button to open a dialog for activating the SDO channels that are predefined in the EDS file. Service data objects (SDOs) allow access to all entries in the CANopen object directory. An SDO creates a peer-to-peer communication channel between two devices (SDO server and client channel).
“Optional device”	☑: The slave is optional and not required for starting the CAN network.
“Sync producing”	Available only when the *“Enable sync producing”* option is cleared in the CANopen Manager. ☑: The I/O transmission is synchronized on the bus. The slave works as a sync producer. The parameters of the sync interval are defined in the settings of the CANopen Manager.
“No initialization”	This option is for non-configurable slave that already start with a valid configuration. ☑: The master does not send configuration SDOs or NMT start commands to the slave. PDO communication and monitoring (heartbeat, node guarding) are performed when this has been configured in the configurator. If the slave does not start automatically, then the user can use the CiA405 NMT function block to send an NMT start command to the slave.
“Default settings”	The availability of this option depends on the contents of the device description file. ☑: Activated by default. The slave nodes are reset to the default parameters before the configuration is loaded to the device or always when the slave is configured. Which parameters can be set is device-specific. The concrete task is performed from the subindex of the list box. ● *“Sub:001”*: All parameters are reset. ● *“Sub:002”*: Communication parameters (index 1000h - 1FFFh manufacturer-specific communication parameters) are reset. ● *“Sub:003”*: Application parameters (index 6000h - 9FFFh manufacturer-specific application parameters) are reset. ● *“Sub:004” - “Sub:127”*: Manufacturer-specific, individual selection of parameters is reset. ● *“ Sub:128” - “Sub254”*: Reserved for future purposes
“Autoconfig PDO mapping”.	This option is available for modular devices only. ☑ PDO mapping is generated automatically from the definitions in the device description and then cannot be changed in the two mapping dialogs. If the automatically generated mapping does not match your application, then you can deactivate the option and configure the mapping manually. We recommend that this option is activated for standard applications.

Table 664: “Node Guarding”

Node guarding is an outdated monitoring method and should not be used anymore because is uses RTR frames. You should always use heartbeats whenever possible. In some exceptions, such as for older slaves, you can use only node guarding.	
“Enable node guarding”	☑: The CANopen Manager sends a message to the slave in the *“Guard time (ms)”* interval. If the slave does not respond with the given *“Guard COB-ID”* (Communication Object Identifier), then the CANopen Manager resends this message as many times as defined in *“Lifetime factor”* or until the slave responds. If the slave does not respond, then it is marked as "unavailable".
“Guard time (ms)”	Interval for sending messages (default: 200 ms)
“Lifetime factor”	When the slave does not respond, a node-guarding error is established according to the *“Lifetime factor”* time multiplied by the *“Guard time”*.
“Enable heartbeat producing”	☑: The module sends heartbeats in the time intervals as given in *“Producer time (ms)”*.
“Producer time (ms)”:	The default setting is 200 as long as there is no special entry or the entry in the device description file is 0.
“Heartbeat consuming (...)”	Opens a *“Heartbeat Consuming Properties”* dialog. There you activate the slaves that you want to watch. The number of possible slaves to be monitored is defined in the EDS file. To do this, you must select the "Enable" check box and enter the Node-ID of the slave and the required values in the *“Heartbeat time”* field (in milliseconds). Then the slave monitors the heartbeats that are sent from the affected slaves (defined by the Node-ID). When no more heartbeats are received, the slave switches off the I/Os. When a slave is monitoring, a green check mark is displayed on the *“Heartbeat Consuming”* button. **Note**: When you insert a device with the heartbeat function, its heartbeat settings are harmonized automatically with the master (CANopen Manager).

Table 665: “Emergency”

“Activate Emergency”	☑: When internal errors occur, the slave sends emergency messages with a unique COB-ID. You can read these messages by using the function blocks from the library `CAA Can Low Level Extern` (`RECV_EMCY_DEF`, `RECV_EMCY`).
“COB-ID”	CAN ID of the EMCY message. Range of possible values: [1...2047].

Table 666: “TIME”

The availability of this function depends on the device description.	
“Enable TIME producing”	☑: The device sends TIME messages.
“COB-ID (Hex)”	(Communication Object Identifier): identifies the time stamp of the message.
“Enable TIME consuming”	☑: The device processes TIME messages.

Table 667: “Checks at Startup”

The respective information is read from the firmware of the CANopen Slave (0x1018 identity object) and compared to the information from the EDS file. In case of disparities, the configuration is stopped and the slaves are not started.	
“Vendor ID”	☑: Check of the vendor ID at startup
“Product number”	☑: Check of the product number at startup
“Revision number”	☑: Check of the revision number at startup

See also

- *Chapter 1.6.6.2.16.1.1.1 “Tab 'CANopen Manager - General'” on page 3800*
- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

Tab 'CANopen Device - PDOs'

Object: CANopen Remote Device, CANopen Local Device

This dialog is available only in the device editor of a CANopen Slave of version V3.5.6.0 or higher. It shows all PDOs and their default settings. In this dialog, you can add new objects and delete or edit existing objects.

On the left side, there are the PDOs that the slave receives from the master. On the right side, there are the PDOs that the slave sends to the master.

“Add PDO”	Opens the *“Select PDO”* dialog where all available PDOs are displayed. In this dialog, you select the PDOs to be added to *“Receive PDOs”* or *“Transmit PDOs”*.
“Add Mapping”	Opens the *“Select Item from Object Directory”* dialog. Objects are listed there that you can add to the PDO mapping.
“Edit”	When a PDO is selected, the *“PDO Properties”* dialog opens. When a PDO mapping is selected, the *“Select Item from Object Directory”* dialog opens.
“Delete”	Deletes the selected objects from the list
“Move Up”	Moves the selected object upwards by one line.
“Move Down”	Moves the selected object downwards by one line.

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

Dialog 'PDO Properties'

“COB-ID”	Every PDO message must have a COB-ID (Communication Object Identifier). You can input explicit values (example: 16#201) or formulas (example: $NODEID+16#200).
“RTR”	Remote Transmission Request. This option is available for transmit PDOs only. ☑ You can use an RTR frame for interrogating the PDO externally.
“Inhibit time (x 100µs)”	You can edit this field only if the device supports this functionality. The inhibit time is the minimum time between two messages of a specific PDO. You can use this setting for preventing PDOs from being sent too often when their values are edited. Default: "0". Possible values: 0–65535.

"Transmission type"	● *"Acyclic - synchronous"*: When a change is made, the PDO is transmitted synchronously, but not periodically. (default) ● *"Cyclic - synchronous"*: The PDO is transmitted every nth sync. ● *"Synchronous – only RTR"*: Available for transmit PDOs only. After a synchronization message, the PDO is updated, but not transmitted. Transmission is by explicit request only (Remote Transmission Request). ● *"Asynchronous – only RTR"*: Available for transmit PDOs only. The PDO is updated and transmitted by explicit request only (Remote Transmission Request). ● *"Asynchronous – manufacturer specific"*: The PDO is transmitted only after specific events. ● *"Asynchronous – device profile"*: The PDO is transmitted according to the CiA device profile.
"Number of syncs"	For transmission type *"Cyclic - synchronous"* only. Indicate the interval for transmitting the PDOs. The value is a multiple of the *"Cycle period (µs)"* of the CANopen Manager. Default: 1. Possible values: 1–240. Example: Number of syncs = 4, Cycle Period = 1000 µs → transmission interval = 4000 µs
"Event time (x 1ms)"	Only for transmission types *"Asynchronous - manufacturer specific"* and *"Asynchronous - device profile"*. You can edit this field only if the device supports this functionality. Indicate the time span that should be between two PDO transmissions PDOs (in milliseconds). Default: "0". Possible values: 0–65535.
"Processing by CANopen Manager"	☑: Default settings ☐: The CANopen Manager does not process the PDO any longer. It is no longer transmitted or received.

Dialog 'Select Item from Object Directory'

For modular slave, you have to clear the *"Autoconfig PDO mapping"* option to be able to configure the mapping manually.

The table shows all object directory entries from the EDS file of the device. For receive PDOs, CODESYS provides only the objects here with write permission (flag = w); for transmit PDOs, read permission.

"Name"	COB-ID of the PDO or the name of the mapped object as it is used in the device description and in the object directory.
"Index"	Index of the object
"Subindex"	Subindex of the object
"Access type"	● *"RW"*: Read/Write ● *"RO"*: Read Only ● *"WO"*: Write Only ● *"RWW"*: Read/Write per SDO; write permission per PDO (==> RxPDO, output from the master viewpoint, input from point of view of the slave). ● *"RWR"*: Read/Write per SDO; read permission per PDO (==> TxPDO, input from the master viewpoint, output from the point of view of the slave). ● CONST=constant
"Type"	Data type of the object
"Default value"	Default value of the object
"Bit length"	Length of the object

Tab 'CANopen Remote Device - SDOs'

During the initialization of the CAN bus, CODESYS transmits the current configuration settings by using SDOs (service data objects).

On this tab, you configure the necessary SDOs. You configure the necessary SDOs and determine the transmission order of the objects and the actions taken in case of a transmission error.

The object order in this list corresponds to the transmission order of SDOs to the module.

> **NOTICE!**
> If the *"Expert settings"* option is not activated for the current device, then only the user-defined SDOs are shown here.

"Add SDO"	Opens the *"Select Item from Object Directory"* dialog where all available SDOs are displayed. The selected object is inserted after the selected object.
"Modify"	Opens the *"Select Item from Object Directory"* dialog and marks the corresponding object. You can modify the object parameters or replace the object with another one.
"Delete"	Deletes the selected objects from the list
"Move Up"	Moves the selected object upwards by one line.
"Move Down"	Moves the selected object downwards by one line.
"Abort on Error"	☑: If an error is detected for this SDO, then the stack stops the configuration phase of the current slave. The slave remains in PREOPERATIONAL mode.
"Jump to Line on Error"	☑: The transmission is continued with the SDO that you indicated in the *"Next Line"* column.
"Next Line"	Line number where processing continues if there an error is detected
"SDO Timeout (ms)"	Timeout for the SDO transmission. If the slave does not respond to the SDO request within this time, then the transmission is canceled with a timeout.
"Create all SDOs"	☑: Creates an SDO for all writable objects starting at index 16#2000 for which a default value is given in the EDS. Only experts should use this option. It should be deactivated for standard use.
"Write complete PDO configuration"	☑: This option forces the writing of all PDO configuration objects. In this way, you make sure that the settings in the project correspond to those of the slave. ☐: PDOs are not deactivated explicitly. The requirement is that the *"Default settings"* option is activated in the common settings of the slave and the PDOs are also deactivated in the EDS. If the default values in the EDS do not match the default settings of the slave firmware, then this procedure may cause problems. In this case, you should activate this option.

See also

- *Chapter 1.6.6.2.16 "CAN onboard" on page 3800*

Dialog 'Select Item from Object Directory'

The table shows all object directory entries from the device EDS file for each SDO that are writable and not larger than 4 bytes. Before you add an SDO for selection in the SDO dialog, you can modify its parameters in the fields below the table. In this way, you can also created an SDO that is not writable in the EDS file by entering a new index/subindex value.

"Name"	COB-ID of the PDO or the name of the mapped object as it is used in the device description and in the object directory.
"Index"	Index of the object

"Subindex"	Subindex of the object
"Access type"	• *"RW"*: Read/Write • *"WO"*: Write Only • *"RWW"*: Read/Write per SDO; write permission per PDO (==> RxPDO, output from the master viewpoint, input from point of view of the slave). • *"RWR"*: Read/Write per SDO; read permission per PDO (==> TxPDO, input from the master viewpoint, output from the point of view of the slave).
"Type"	Data type of the object
"Default value"	Default value of the object
"Bit length"	Length of the object
"Value"	

CANopen module

CANopen modules are components that you insert below a CANopen remote device.

J1939

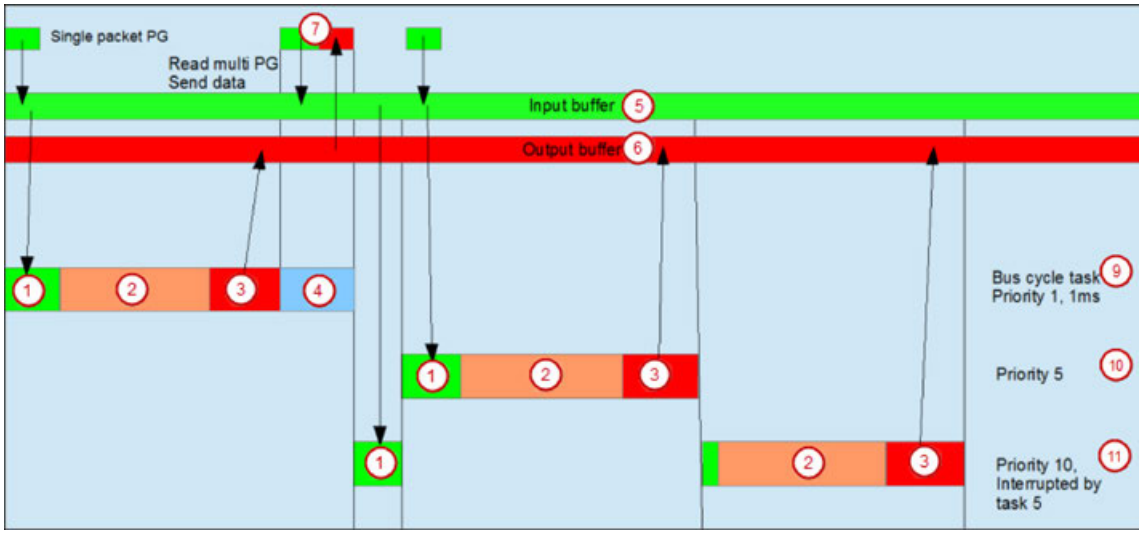
J1939 is a CAN-based protocol (CAN stands for "Controller Area Network"). It was developed for serial data transmission between electronic control units (ECU) in heavy goods vehicles. The CODESYS plug-in 'DeviceEditorJ1939' provides dialogs to configure J1939 devices according to SAE J1939 standards.

See also

- *Chapter 1.6.6.2.16 "CAN onboard" on page 3800*

Bus Cycle Task

Behavior of the bus cycle for J1939



(1) Receive single package PG (4) Receive multi-package PGs, send PGs

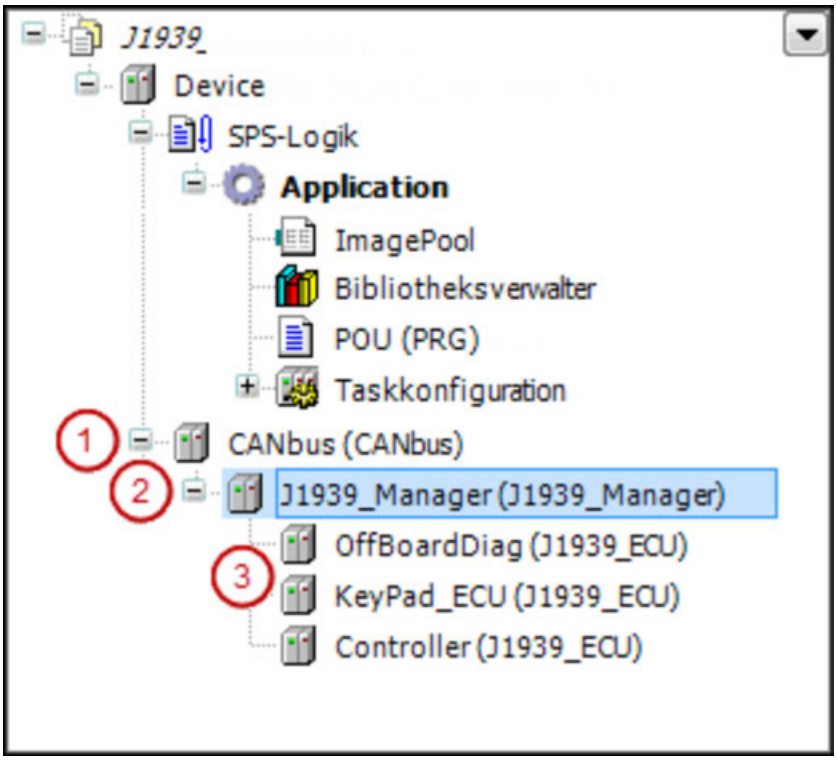
See also

- *Tab '<device name> I/O Mapping'*

J1939 manager

The J1939 Manager is inserted in the device tree below the CAN bus node. It provides the J1939 parameter groups and signal database. The ECUs are inserted below the J1939 Manager.

The *“Scan Devices”* command is not available for J1939.



(1) CANopen Manager (2): J1939 Manager (3) J1939 ECU

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*
- *Chapter 1.6.6.2.16.2.2.1 “Tab 'J1939 Manager - General'” on page 3809*

Tab 'J1939 Manager - General'

In CODESYS version 3.5 SP17 and higher, the J1939 configurator is no longer supplied with a parameter group / signal database. The old database is no longer supported.

However, you can post-install a DBC database in the J1939 Manager.
A database can be purchased, for example from CSS Electronics: https://www.csselectronics.com/screen/product/j1939-dbc-file-pgn-spn

If you do not install a database, then you can also configure parameter groups and signals manually on the “User-Defined” tab.

“Databases”	List with names of installed databases (DBC or DB format)
“Install”	Opens the file manager to select a J1939 file (DBC or DB format). The J1939 files are usually stored in "`C:\ProgramData\CODESYS\J1939 Databases`".

“Uninstall”	Uninstalls the selected database
“Set as default”	Sets a database as the default database. This database is then set as default in the *“Add Parameter Group”* and *“Add Signal”* dialogs.

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

J1939 ECU

Tab 'J1939 ECU - General'

In this dialog of the J1939 ECU editor, the general parameters of a J1939 ECU can be displayed and modified.

Table 668: “General”

“Preferred address”	Address of the ECU. If more than one ECU with the same address exists in the network, then all affected ECUs get a new address. The requirement is that the ECUs allow an address change (*“Arbitrary Address Capable”*).
“Local Device”	☑ You can configure any number of local ECUs. Then every local ECU is its own ECU instance in the J1939 network. For local devices, an additional *“RX Signals (P2P)”* dialog is provided to configure received signals.

Table 669: “ECU NAME”

“NAME (64 bit): 16#”	Hexadecimal 64-bit code that contains complete information about the subsequent parameters. Each time this code is modified, the respective parameter is also modified. The same is true for the other direction.
“Arbitrary address capable”	☑ If the ECU detects an address conflict, then it tries independently to set another address.
“Industry group”	List of industry groups according to the definition from SAE J1939.
“Vehicle system instance”	The parameter depends on the *“Vehicle system”*. The 4-bit value assigns a number to each instance of the *“Vehicle system”*.
“Vehicle system”	The value is defined in the SAE J1939 standard.
“Reserved”	Always deactivated and reserved for future SAE definitions.
“Function”	The parameter is defined and assigned by SAE. The range of values is 0...255, but not all values are assigned. The interpretation of values, which are greater than or equal to 127, depends on the *“Industry”* selection. For example, the value "133" means "Product Flow" in the "Agricultural and Forestry Equipment" industry. If "Construction Equipment" is selected for *“Industry”*, then the same value means "Land Leveling System Display". If the value is less than 128 (0 – 127), then there is no dependency to other parameters.

"Function instance"	The parameter is related to the *"Function"* field. A J1939 network can consist of multiple ECUs with the same *"Function"*. The 5-bit *"Function instance"* assigns a number to each instance of the *"Function"*, where 0 is assigned to the first instance.
"ECU instance"	A J1939 network can include multiple ECUs that have the same task. For example, a vehicle can have two identical ECUs, where one measures vehicle speed and the other measures the trailer speed.
"Manufacturer code"	The 11-bit manufacturer code is assigned by SAE and indicates the company that manufactured the ECU. This code is defined in the SAE J1939 document.
"Identity number"	The 21-bit identity number is assigned by the manufacturer and should be used for assuring unique names in a product line. The manufacturer can also add more information to the identity number, such as serial number and date of manufacture.

Table 670: "Communication Watchdog"

"Enable communication watchdog"	☑ The stack checks whether the ECU transmits data within the given *"Watchdog time"*. If this does not happen, then the device is classified as "not available" and highlighted in red in the device tree.

See also

- *Chapter 1.6.6.2.16 "CAN onboard" on page 3800*

Tab 'J1939 ECU - TX Signals'

This dialog shows the parameter groups that are transmitted to all other ECUs (broadcast) or to a specific ECU (P2P). In this dialog, you can activate and deactivate individual groups and modify their parameters. You can also add new groups or signals to the list.

"Enable"	☑: The parameter group is transmitted. ☐ The parameter group is not transmitted for a local device. For a remote device, the PLC does not process this group.
"Type"	• *"Broadcast"*: The parameter group is transmitted to all ECUs. • *"Peer-to-Peer P2P"*: The parameter group is transmitted to a specific ECU.
"Add PG"	Opens the *"Add Parameter Groups"* dialog.
"Add Signal"	Opens the *"Add Signal"* dialog. The button is enabled only if you have selected a parameter group.

Parameter group properties

Table 671: "General"

"PGN"	The "parameter group number" is a unique number for addressing a parameter group.
"Name"	Name of the parameter group
"Description"	Description of the parameter group
"Length"	The length of the message data (0...1785 bytes). Due to the maximum array length of 8 bytes, messages with over 8 bytes are transmitted as multipackages.

Table 672: “Transmission settings”

These settings are provided only for the TX parameter groups of local devices.	
“Priority”	Priority of the parameter group (0..7). Priority 0 is the highest and 7 is the lowest.
“Target address”	The target address is needed for P2P parameter groups only.
“Transmission mode”	Determines the time when a parameter group is transmitted (for local devices). ● *“Mode change”*: The PG is transmitted when the value of the signal changes. ● *“Cyclic”*: The PG is transmitted after a specified number of PLC cycles (see cycle time factor). ● *“On request”*: The PG is transmitted on request of another device. ● *“Application-controlled”*: The PG is transmitted when triggered by the application.
“Cycle time factor”	Number of PLC cycles after which the parameter group is transmitted. Only applies for cyclic transmission.

Signal parameters

Table 673: “General”

“SPN”	Suspect Parameter Number. One of the numbers assigned by the SAE for a specific parameter in a parameter group.
“Name”	Name of the parameter
“Description”	Description of the parameters
“Length (bits)”	Length of the signal (in bits: 1...14280).
“Byte position (0..1784)”	Start byte in the parameter group (0...1784).
“Bit position (0..7)”	Bit position of the start byte (0..7).

Table 674: “Conversion”

“Conversion”	TRUE: The value is calculated with scaling and offset.
“RAW data type”	Format of the raw data: Unsigned / Signed / Float / Double
“Byte order”	Little endian or big endian of the raw signal
“Scaling”	Factor (for *“Conversion”* =TRUE)
“Offset”	Offset (for *“Conversion”* =TRUE)
“Minimum value”	Expected minimum value of the converted signal (for informational purposes only)
“Maximum value”	Expected maximum value of the converted signal (for informational purposes only)
“Unit”	Unit of the converted signal
“ICE data type”	Resulting data type of the I/O channels

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

Tab 'J1939 ECU - P2P RX Signals'

This dialog is available for local ECUs only. It shows all PGs (parameter groups) that should be received by other ECUs. In this dialog, individual groups can be activated and deactivated as well as their parameters modified. New groups or signals can also be added to the list.

The commands and parameters of this dialog are the same as those on the *“TX Signals”* tab.

See also

- *Chapter 1.6.6.2.16.2.3.2 “Tab 'J1939 ECU - TX Signals'” on page 3811*

Command 'Scan for Devices'

Function: The command establishes a brief connection to the hardware and determines the devices in the network. Then you can apply the devices found into the device tree of your project.

Call: Menu bar: *“Project”*; context menu of a device object in the device tree

Requirement: The communication settings to the controller are correct. The gateway and the PLC are started. The device supports the scan function.

The following devices provide the scan function: EtherCAT master, EtherNet/IP Scanner (IEC), Sercos master, CANopen Manager, CANopen Manager SIL2, PROFINET controller und PROFIBUS DP master.

You can perform the device scan immediately if the scan function is permanently implemented in the PLC. When scan function is implemented in a library, you have to log in only one time to download the library to the controller.

The command refers to the master controller selected in the device tree. For example, an already inserted PROFINET IO controller can be selected and the command used to determine the I/O devices and I/O modules assigned to it.

After performing the scan operation, the *“Scan Devices”* dialog opens and displays the found devices.

Dialog 'Scan Devices'

Table 675: “Scanned Devices”

“Device name, Device type, Address, Station name, etc.”	Data about the scanned device depending on network type. When you change a value in the list of scanned devices, the value is shown in italics. This indicates that the new value has been changed in the editor in CODESYS, but not in the device. When you download the value to the device, it is shown normally. Value that indicate differences between the project and the scanned device are shown in orange. If multiple device descriptions are available for the scanned device, then the name is displayed in bold. The selection of the matching device description is resolved differently for different fieldbuses. For more detailed information, see the corresponding fieldbus chapters. If a device description cannot be found, then the following message is shown: "Attention! The device was not found in the repository." Depending on the bus system, additional information is displayed, such as manufacturer number and product number. The device cannot be inserted into the project without the installed device description.
“Show differences to project”	☑: The table in the dialog also shows additional configured devices (in the device tree of the project). ☐: The table shows all scanned devices. The configured devices are not shown.
“Scan for Devices”	Starts a new search.
“Copy All Devices to Project”	The device that is selected in the table is inserted into the device tree in the project. If nothing is selected, then all scanned devices are shown.

NOTICE!
If you insert devices, which are available in the device tree, to the device tree with *“Copy All Devices to Project”*, then the following should be noted. The data of the *“Process Data”* and *“<...> I/O Mapping”* tabs of the existing devices can be overwritten with the data of the recently inserted devices.

Table 676: “Configured Devices”

This part of the dialog is visible only when you select the *“Show differences to project”* option. Differences between the scanned and configured devices are color-coded. Devices displayed in green are identical on both sides. Devices displayed in red are available only in the view of the scanned or configured devices.	
	If you have selected a device in both views, then the scanned devices are inserted above the selected configured device.
	If you have selected a device in both views, then the scanned devices are inserted below the selected configured device.
	If you have selected a device in both views, then the configured devices are replaced by the selected scanned device.
	All scanned devices are copied to the project.
	Deletes the selected configure device.

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

Tab 'CANbus - General'

Table 677: “General”

“Network”	Number of the CAN network that is linked via the CAN bus interface. Permitted values: 0 to 100.
“Baud rate”	Baud rate (in bits per second) for transmitting data on the bus. The default value is used from the device description file (*.devdesc) of the CAN bus device. You can select the baud rate from the list box or type it directly into the input field.

See also

- *Chapter 1.6.6.2.16 “CAN onboard” on page 3800*

1.6.6.2.17 EtherCAT configurator

Refer to the general description for information about the following tabs of the device editor.

- *Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*
- *Chapter 1.4.1.20.2.8.12 “Tab '<device name> IEC Objects'” on page 859*
- *Chapter 1.4.1.20.2.8.3 “Tab 'Parameters'” on page 844*
- *Chapter 1.4.1.20.2.8.18 “Tab 'Status'” on page 870*
- *Chapter 1.4.1.20.2.8.19 “Tab 'Information'” on page 870*

Only in the case of special features is there an additional help page for the specific device editor.

If the "<device name> Parameters" tab is not shown, then select the “Show generic device configuration editors” option in the CODESYS options (“Device Editor” category).

The configuration of EtherCAT modules is based on the device description files for the master and slave devices employed and can be adapted in the project in configuration dialogs. In order to ensure the simplest and most error-free use possible, we recommend for standard applications that you activate the option for the *“Automatic Configuration”* of the master, so that the majority of the configuration settings are performed automatically.

Requirements

The requirement for the combination of EtherCAT devices with a CODESYS Control Win V3 is the installation of the program library WinPCap (freely downloadable, e.g. from winpcap.org). Furthermore, add the following entries to the CODESYS configuration file (...\ \GatewayPLC\\CODESYSSP.cfg):

- `component.<subsequent number>=CmpEt100Drv`
  Required only with the RTE. The RTE requires special network drivers. Available for Realtek RTL81x9/RTL8169, Intel Pro 100 / 1000
- `component.<subsequent number>=CmpRTL81x9Mpd`
  Required only with the RTE. Available for RTL8139.
- `component.<subsequent number>=8169Mpd`
  Required only with the RTE. Available for Realtek RTL8169 or RTL8168 (PCIe version))

The bus cycle task is set in the general PLC settings.

Access to the EtherCAT configuration by the application takes place via instances of the EtherCAT master and EtherCAT slave. If the EtherCAT master or EtherCAT slaves are inserted as objects into a project, instances are automatically created for master and slaves that can be addressed in the application program. For example a restart, a stop or a status check of the EtherCAT device can be performed from the application.

Furthermore, the EtherCAT library offers function blocks for the reading and writing of individual parameters, even during bus operation.

See also

- *Chapter 1.6.6.2.17.1.1 “Tab 'EtherCAT Master - General' ” on page 3816*
- *Chapter 1.4.1.20.4.13.6 “Dialog 'Options' - 'Device Editor'” on page 1190*
- *Chapter 1.4.1.20.2.8.9 “Tab 'PLC Settings'” on page 850*

EtherCAT master

Tab 'EtherCAT Master - General'

Object: EtherCAT Master

The tab is used for the configuration of the basic settings for the EtherCAT Master. The basic settings are preset from the device description file.

Settings of the configuration parameters

NOTICE!

The auto-configuration mode (*“Autoconfig master/slaves”* option) is selected by default and is adequate for standard applications. If the mode is not selected, then all configuration settings for the master and the slave(s) have to be done manually. Expert knowledge is required to do this. The auto-configuration mode option has to be switched off to configure slave-to-slave communication.

“Autoconfig master/slaves”	☑ The main part of the master and slave configuration is done automatically, based on the device description file and implicit calculations. The dialog for the FMMU/Sync settings is not available. Even if this option of the master is selected, an expert mode can be enabled explicitly for each individual slave, which allows for manual editing of the automatically generated process data configuration.

Table 678: "EtherCAT NIC Settings"

"Target address (MAC)"	MAC address of the device in the EtherCAT network that is to receive the telegrams. Options ● *"Broadcast"*: A *"Target address (MAC)"* does not have to be specified. ● *"Redundancy"*: Enabled when the bus is constructed in a ring topology and redundancy is to be supported. With this function, the EtherCAT network remains functional even in the case of a broken cable. When this function is enabled, the parameters have to be defined in the *"Redundancy EtherCAT NIC Settings"* area.
"Source address (MAC)"	MAC address of the controller (target system) or network name (name of the adapter or PLC (target system))
"Network name "	Name or MAC of the network, depending on which of the following options is selected:
"Select network by MAC"	☑: The network is specified by the MAC ID. Then the project cannot be used on another device because each network adapter has a unique MAC ID.
"Select network by name"	☑: Network is identified by the network name and the project is device-independent.
"Scan"	Scans the network for the MAC IDs or names of the target devices that are currently available.

Table 679: "Redundancy EtherCAT NIC Settings"

These settings are displayed only when the *"Redundancy"* option is selected. Here the parameters of the additional device are defined according to the description for *"EtherCAT NIC Settings"*.

Table 680: "Distributed Clock"

"Cycle time (µs)"	Time span after which a new data telegram is dispatched on the bus. When the *"Distributed Clock"* function is enabled in the slave, the master cycle time specified here is transferred to the slave clocks. As a result, a precise synchronization of the data exchange can be achieved. This is particularly important when spatially distributed processes require simultaneous actions. An example of a simultaneous action is applications in which multiple axes have to execute coordinated movements at the same time. A very precise, network-wide time base with a jitter of considerably less than 1 microsecond can be achieved in this way.
"Sync offset"	Parameter for setting the delay time between the DC time base of the EtherCAT Slave and the cycle start of the PLC. With the default value of 20%, the PLC cycle starts 20% of the bus cycle time after the sync interrupt of the slave. This means in the case of ● `FrameAtTaskStart` = `FALSE` when the EtherCAT data is sent at the end of the PLC cycle: The PLC cycle may require 80% of the bus cycle time minus the delay time in the runtime, and this without the master no longer placing the current process data on the bus in time (assuming that the EtherCAT Slave expects the new data exactly with the sync interrupt). ● `FrameAtTaskStart` = `TRUE` (default value when using CODESYS SoftMotion): For the controller program, nearly 100% of the cycle is always available. Here the *"Sync offset"* determines only when the EtherCAT data of the master is exchanged to and from the slaves relative to the time base of the EtherCAT Slave.

“Sync window monitoring”	☑ Synchronization of the slaves can be monitored.
“Sync window”	Time for *“Sync window monitoring”*. When the synchronization of all slaves is within this time window, the variable `xSyncInWindow` (`IoDrvEthercat`) is set to `TRUE`, otherwise to `FALSE`.

Table 681: “Options”

“Use LRW instead of LWR/ LRD”	☑ Direct communication from slave to slave is possible. Combined read/write commands (LRW) are used instead of separate read commands (LRD) and write commands (LWR).
“Messages per task”	☑ Read and write commands (the handling of the input and output messages) can be controlled by means of various tasks.
“Automatically restart slaves”	☑ The master immediately attempts to restart the slaves in the case of a communication breakdown.

Table 682: “Master Settings”

These settings can be edited only when the *“Autoconfig master/slaves”* option is deactivated. Otherwise this is done automatically and they are not visible here.	
“Image In Address”	First logical address of the first slave for input data
“Image Out Address”	First logical address of the first slave for output data

See also

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Master - Sync Unit Assignment'

Object: EtherCAT Master

The tab shows all slaves that are inserted below a specific master with an assignment to the sync units.

With the EtherCAT sync units, multiple slaves are configured into groups and subdivided into smaller units. For each group, the working counter can be monitored for an improved and more exact error detection. As soon as a slave is missing in a sync unit group, the other slaves in the group are also shown as missing. Detection occurs immediately in the next bus cycle because the working counter is checked continuously. With the device diagnosis, the missing group can be remedied as quickly as possible.

Unaffected groups remain operable without any interference.

Sync unit support is defined by the device description of the EtherCAT Master and can be disabled for vendor-specific device descriptions. By default, it is provided with a device description of version 3.5.8.0 and higher.

“Device name”	Name of the slave
“Sync unit”	Name of the selected sync unit. You can combine single devices or entire groups (multiple selection) into on sync unit group.
“Add”	When you type a name in the text field, you can create a new sync unit.
“Delete”	Deletes the selected sync unit. When slaves are assigned to the group to be deleted, a warning dialog opens. If you click *“Yes”* to acknowledge the dialog, then these devices are reassigned to the default group.

See also

- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Master - Parameters'

Object: EtherCAT Master

The tab contains the master parameters which are defined in the device description file.

When the auto-configuration mode is selected in the *“Master”* dialog, the parameters are set here automatically according to the specifications from the device description file and the network topology. Nothing should be changed in the generic editor because an invalid configuration can be set here.

“Value”	Editable: A change is effective only when the auto-configuration mode is disabled. Whether or not the change becomes effective depends on the respective parameter.

See also

- *Chapter 1.6.6.2.17.2.6 “Tab 'EtherCAT Slave - Parameters' ” on page 3827*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*
- *Configuration*

EtherCAT slave

Tab 'EtherCAT Slave - General'

Object: EtherCAT Slave

The basic settings for the EtherCAT Slave are configured on this tab. The basic settings are preset from the device description file.

Table 683: “Address ”

Fields can be edited only when the auto-configuration mode of the EtherCAT Master is disabled.	
“AutoInc address”	Self-incrementing address (16-bit) that results from the position of the slave in the network. The address is used only during the system boot when the master assigns the EtherCAT addresses to its slaves. When the first message runs through all the slaves for this purpose, each slave increments its *“AutoInc address”* by 1. The slave with address 0 then gets the data. A possible input here is "-8".
“EtherCAT address”	Final address of the slaves, assigned by the master during bootup. The address is independent of the position of the slave in the network.

Table 684: “Additional”

“Expert settings”	☑: Additional settings are possible for the startup checking and time monitoring (see below). The *“Expert Process Data”* tab is also available in the device editor. However, expert settings are not required for standard applications. The auto-configuration mode is recommended and sufficient for standard applications.
“Optional”	At the start of the stack, the system checks whether optional devices are available. ☑: The slave is defined as optional and no error message is generated if the device is missing from the bus system. If a device is not found, then it is disabled automatically and displayed in gray in the device tree. A corresponding message is displayed in the logger. Note: If you define a slave as "optional", then it has to have a unique identification. You can change this by means of the three possible settings in the *“Identification”* section. Available only when the *“Autoconfig master/slaves”* option is selected in the settings of the EtherCAT Master and the EtherCAT Slave supports this function.

Table 685: “Distributed Clock”

“Select DC”	List box with all settings for the distributed clocks of the device description file
“Enable ”	☑: Cycle time for the data exchange. It is displayed in the *“Sync unit cycle (µs)”* input field and determined by the cycle time of the master. As a result, the master clock can synchronize the data exchange in the network.

The *“Sync0”* and *“Sync1”* settings described below are slave-dependent:

Table 686: “Sync0”

“Enable Sync 0”	☑: Synchronization unit *“Sync0”* is used. A synchronization unit describes a set of process data that is exchanged synchronously.
“Sync unit cycle”	☑: The master cycle time (multiplied by the factor selected from the list box) is used as the synchronization cycle time for the slave. *“Cycle time (µs)”* displays the cycle time currently set.
“User-defined”	☑: A custom cycle time (in microseconds) can be specified in the *“Cycle time (µs)”* field.

Table 687: “Sync1”

“Enable Sync 1”	☑: Synchronization unit *“Sync1”* is used. A synchronization unit describes a set of process data that is exchanged synchronously.
“Sync unit cycle”	☑: The master cycle time (multiplied by the factor selected from the list box) is used as the synchronization cycle time for the slave. The *“Cycle time (µs)”* field displays the cycle time currently set.
“User-defined”	☑: A custom cycle time (in microseconds) can be specified in the *“Cycle time (µs)”* field.

Table 688: “Diagnosis”

This area section appears in online mode only.	
“Current State”	State of the slave Possible states: *“Init”*, *“Preoperational”*, *“Safe Operational”*, and *“Operational”* The state *Operational* indicates that the slave configuration has been correctly completed and that process data (inputs and outputs) are being accepted.

Table 689: “Startup Checking”

“Check vendor ID”	By default the vendor ID and product ID of the device are checked against the current configuration settings when the system boots up. If they do not agree, then the bus is stopped and no further actions are executed. This is done to prevent an incorrect configuration from being loaded onto the bus system. Options for deactivating the corresponding check.
“Check product ID”	
“Check revision number”	☑: The revision number is checked during the system bootup according to your selection in the list box.
“Download expected slot configuration”	☑ For online verification of the configured and actual module configuration. If the configurations do not match, then the device still switches to "Run". In this case, an entry is made in the device logbook.

Table 690: “Timeouts”

By default, watchdog is not defined for the following actions. If necessary, an appropriate timeout can be specified here (in milliseconds):	
“SDO access”	Transmits the SDO list at system start. Specified in milliseconds.
“I -> P”	Switch from *“Init”* mode to *“Preoperational”* mode. Specified in milliseconds.
“P -> S / S -> O”	Switch from *“Preoperational”* mode to *“Safe Operational”* mode, or from *“Safe Operational”* mode to *“Operational”* mode. Specified in milliseconds.

Table 691: “DC Cyclic Unit Control: Assign to Local µC”

One or more options for the *“Distributed Clock”* function can be activated here that should be used on the local microprocessor. The check is performed in the registry at 0x980 in the EtherCAT Slave. Possible settings:	
“Cycle unit”	
“Latch unit 0”	
“Latch unit 1”	

Table 692: “Watchdog”

“Set multiplier”	The PDI watchdog and SM watchdog receive their impulses from the local terminal clock divided by the watchdog multiplier.
“Set PDI watchdog”	This watchdog triggers when there is no PDI communication with the EtherCAT Slave controller for longer than the PDI (Process Data Interface) watchdog time which has been set and activated.
“Set SM watchdog”	This watchdog triggers when there is no EtherCAT process data communication with the terminal for longer than the SM (SyncManager) watchdog time that has been set and activated.

Table 693: “Identification”

In this section, you set the device identification of the slave. As a result, you can make the address of the slave independent of its position in the bus. The following options are visible only when the *“Activate expert settings”* option or *“Optional”* option is selected. If you have identified the slave as *“Optional”*, then you have to assign a unique ID to it.	
“Disabled”	The identification of the slave is not checked.

“Configured station alias (ADO 0x0012)”	Address that is stored in the EEPROM of the device. You can change the value in the *“Scan Devices”* dialog or in online mode. For stock devices, you need to assign this number one time. This means that you have to connect the device one time to an EtherCAT Master and save the number.
“Write to EEPROM”	Visible in online mode only for *“Configured station alias”*. Writes the defined address for *“Value”* to the EEPROM of the slave.
“Explicit device identification (ADO 0x0134)”	The device identification is hard set on the hardware (for example, by DIP switches). It is displayed in *“Actual address”*.
“Data Word (2 Bytes)”	A 2-byte value for the identification is saved in the slave.
“Value”	Expected value for the check. If the actual value does not correspond to this setting, then an error is issued.
“ADO (hex)”	Initial value from the device description. You can change this value in the *“Data word”* option.
“Actual address”	Visible in online mode only. Displays the address of the slave. You can use this display for checking the success of the *“Write to EEPROM”* command.

See also

- *Chapter 1.6.6.2.17.1.1 “Tab 'EtherCAT Master - General' ” on page 3816*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Slave - FMMU/Sync'

Object: EtherCAT Slave

The tab shows the FMMUs and Sync Manager of the EtherCAT Slave as they are defined in the device description file. There is an option to edit the FMMUs and Sync Manager (for example, for the configuration of slave-to-slave communication).

Requirement: The auto-configuration mode in the EtherCAT Master is disabled.

Note that these are expert settings which are not usually required for standard applications.

Table 694: “FMMU ”

The table shows the *Fieldbus Memory Management Units* of the slave, which are used for handling the process data. In each case the allocation of the logical address (*“Global Start Address”*) to a physical address (*“Phys. start address”*) is defined. Bit-by-bit mapping is possible.	
“Modify”	
“Add”	
“Delete”	

Table 695: “Edit FMMU”

“Global Start Address”	
“Length”	
“Start bit”	
“End bit”	
“Phys. start address”	

“Phys. start bit”	
“Access”	*“Read”* *“Write”*
“Flags”	*“Enable”*

Table 696: “Sync Manager ”

Display and editing of the synchronization manager of the slave. The physical start address, the type of access, the buffer, and the physical address to which the interrupts are to be sent (as well as others) are defined for each available Sync Manager type (mailbox in, mailbox out, inputs, outputs).

Table 697: “Edit Syncman”

“Phys. start address”	
“Length”	
“Buffer”	*“1”* *“3”*
“Access”	*“Read”* *“Write”*
“Interrupts”	*“to EtherCAT”* *“to PDI”*
“Flag control”	*“Enable”*
“Watchdog”	*“Trigger”*
“SyncMan type”	*“”*

See also

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*
- *Chapter 1.6.6.2.17.1.1 “Tab 'EtherCAT Master - General' ” on page 3816*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Slave - Expert Mode Process Data'

Object: EtherCAT Slave

The tab provides another more detailed view of the process data, which is also displayed in the *“Process Data”* dialog. Moreover, the download of the PDO assignment and the PDO configuration is enabled here.

Requirement: The expert settings for the slave are selected.

See also:

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*

Table 698: “Sync Manager ”

List of the Sync Managers with data size and PDO type

Table 699: “PDO assignment (16#1C12)”

List of the PDOs assigned to the selected *“Sync-Manager”*.
When a check box is selected, the PDOs are enabled and I/O channels are created. This is similar to the simple PDO configuration view.

Table 700: “PDO list”

List of the PDOs assigned to the selected *“Sync-Manager”*.
You can add new entries or edit or delete existing entries by executing the respective commands (*“Add”*, *“Delete”*, *“Edit”*) in the command bar or context menu.

Table 701: “Edit PDO list”

“Name”	
“Index”	
“Direction”	• *“TxPDO (input)”*: ☑: The PDO is transmitted from the master to the slave. • *“RxPDO (output)”*: ☑ The PDO is transmitted from the slave to the master.
“Flags”	• *“Required”*: The PDO is required and cannot be disabled in the *“PDO assignment”*. • *“Windows contents”*: The contents of the PDO are fixed and cannot be modified. It is then not possible to add entries in *“PDO contents”*. • *“Virtual PDO”*: Reserved for future use
“Exclude PDOs”	It is possible to define an exclusion list. When a PDO is enabled in the *“PDO assignment”*, others are disabled and cannot be enabled.
“Sync unit”	ID of the Sync Manager to which the PDO is to be assigned

Table 702: “PDO Contents”

Displays the contents of the PDOs selected in the *“PDO list”*. You can add new entries or edit or delete existing entries by executing the respective commands (*“Add”*, *“Delete”*, *“Edit”*) in the command bar or context menu. You can change the PDO order by clicking *“Move Up”* and *“Move Down”*.

Table 703: “Download”

“PDO assignment”	☑: Specific CoE commands for initializing the `0x1cxx` objects are generated and written to the slave.
“PDO configuration”	☑: The CoE commands for `0x16xx` or `0x1axx` are generated, and then the PDO mapping is downloaded to the slave. Normally, the default values originate from the ESI file and the device has to support this functionality. For example, if a device has a fixed configuration, then these commands are regarded as flawed.
“Load PDO info from the device”	The current PDO configuration is read from the slave and entered into the configuration. The lists in the upper and lower right are then deleted and filled with the read data. This is especially useful when the ESI file is incomplete and the configuration is available only on the slave.

See also

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*
- *Chapter 1.6.6.2.17.2.4 “Tab 'EtherCAT Slave - Process Data' ” on page 3825*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Slave - Process Data'

Object: EtherCAT Slave

The tab of the EtherCAT configurator displays the process data for the inputs and outputs of the slave. The data is preset from the device description file.

Table 704: "Select the Outputs"

The table shows the outputs of the slave defined by *"Start address"*, *"Type"*, and *"Index"*. If outputs of the device are enabled here (for writing), then these outputs can be assigned to project variables in the *"EtherCAT I/O Mapping"* dialog.

Table 705: "Select the Inputs"

The table shows the inputs of the slave defined by *"Name"*, *"Type"*, and *"Index"*. If inputs of the device are enabled here (for reading), then these inputs can be assigned to project variables in the *"EtherCAT I/O Mapping"* dialog.

See also

- *Chapter 1.6.6.2.17.2.1 "Tab 'EtherCAT Slave - General' " on page 3819*
- *Chapter 1.6.6.2.17 "EtherCAT configurator" on page 3815*

Tab 'EtherCAT Slave - Startup Parameters'

Object: EtherCAT Slave

On the tab, the SDOs (service data objects) for 'CAN over EtherCAT' (CoE) or the IDNs (identification numbers) for 'Servodrive over EtherCAT' (SoE) are defined for the current slave. These parameters are determined for the device when the system is started.

The object directory with the required data objects is described in the EtherCAT XML description file or in an EDS file that is referenced in the XML file.

Requirement: The device supports 'CAN over EtherCAT' or 'Servodrive over EtherCAT'.

Some modules that are inserted below a slave have their own startup parameters. These parameters are also displayed in this list but cannot be edited here. The parameters are modified in the editor of the corresponding module.

List of SDOs or IDNs The order (from top to bottom) specifies the order in which the objects are transferred to the module.	
"Line"	Line number
"Index:Subindex"	For CoE only
"IDN"	For SoE only Identification number
Name	
"Bit length"	Bit length of the SDO or IDN
"Abort on Error"	☑: In case of error, the transfer is aborted with an error status.
"Jump to Line on Error "	☑: In case of error, the transfer is resumed with the SDO or IDN at the specified *"Line"*.

“Next Line ”	☑: The transfer is resumed with the SDO or IDN at the next line.
“Comment”	Input field for comments
“Move Up”	Moves the selected line upwards by one line
“Move Down”	Moves the selected line downwards by one line
“Add”	Opens the *“Select Item from Object Directory”* dialog adding SDOs or IDNs.
“Delete”	Removes the selected entry.
“Modify”	Opens the *“Select Item from Object Directory”* dialog for changing the parameters of the selected SDO or IDN

Dialog 'Select Item from Object Directory'

The dialog lists all available object directory entries as defined in the XML file. The parameters of the objects can be modified in this dialog. New objects can also be created. This is useful when none or only an incomplete object directory exists.

Table 706: “CAN over EtherCAT ”

List of available object directory entries as defined in the EDS file.	
Column *“Index:Subindex”*	Identifies the entry in the object directory
Column *“Name”*	
Column *“Flags”*	Display of access flags: RW (read/write), RO (read only), WO (write only)
Column *“Base Value”*	Editable (double-click to open)
Input fields	
“Name”	Input field for displaying and changing the name
“Index: 16#”	By specifying new index/subindex entries, a new object can be added to the SDO that is not yet described in the EDS file.
“Subindex: 16#”	
“Bit length”	Range of values of the object
“Value”	Each value may be max. one byte (0-255). It can also be a hexadecimal in IEC syntax (for example, 16#ad). If the *“Byte array”* option is enabled, then the values have to be specified as a comma-separated list (for example, 1,2,3,4).
“Full access”	The complete object is written with one access and all subindexes are set at the same time. The time needed for the transfer is reduced because not every subindex has to be transferred individually.
“Byte array”	☑ Values can be specified as a comma-separated byte array.

Table 707: Servodrive over EtherCAT

List of available object directory entries as defined in the XML file.	
Column *“IDN”*	Identification number
Column *“Base Value”*	Base value of the IDN. Double-click to modify.
Input fields	
“IDN”	Identification number: Composed from the subsequent parameters ● *“S”*: Standard data ● *“P”*: Product-specific data ● *“PSet”*: Parameter set ● *“Offset”*

“Bit length”	List box for selecting the bit length
“Value”	List box for selecting the value
“Channel”	If the object has multiple subobjects, then this list box is displayed automatically.
“As list”	Parameters are loaded as a list. The first four bytes indicate the length. ☑: The length is calculated automatically.

See also

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Slave - Parameters'

Object: EtherCAT Slave

The tab contains the slave parameters which are defined in the device description file.

When the auto-configuration mode of the master is selected, the parameters are set here automatically according to the specifications from the description file and the network topology. For standard applications, it is also not normally required to edit them.

“Value”	Only a few parameters are editable. A change is effective only when the auto-configuration mode is disabled. Basically, the user should not modify anything here because doing so could create an invalid configuration, which would prevent the slave from entering into the operational state.

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*
- *Chapter 1.6.6.2.17.1.3 “Tab 'EtherCAT Master - Parameters' ” on page 3819*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

Tab 'EtherCAT Slave - EoE Settings'

Object: EtherCAT Slave

This tab is used to configure the communication settings for the individual slaves that support Ethernet over EtherCAT (EoE).

Requirement:

- When using CODESYS Control Win V3, the Microsoft Loopback Adapter has to be installed as a virtual Ethernet adapter. Installation instructions can be found online.

Table 708: “Settings”

“Virtual Ethernet Port”	☑: Enables the EOE functionality of the slave. A unique *“Virtual MAC ID”* has to be defined.
“Virtual MAC ID”	Input field for the *“Virtual MAC ID”*
“Switch port”	◉: The device acts as a switch. No additional network settings are required.
“IP port”	◉: The device acts as an IP port. The *“IP Settings”* have to be configured.

Table 709: “IP Settings”

The Ethernet communication parameters have to be set according to the parameters of the virtual Ethernet adapter.	
“IP address”	IP address of the slave in the network (length: 4 bytes) The IP port has to be in the same range as the virtual Ethernet adapter. For example, if the address of the network adapter is 192.168.1.1 and the subnet mask is 255.255.255.0, then the IP port has to be in the range from 192.168.1.2 to 192.168.1.254.
“Subnet mask”	Subnet mask (length: 4 bytes)
“Default gateway”	Default gateway (length: 4 bytes)
“DNS server”	IP address of the DNS server
“DNS name”	Name of the DNS server

See also

- *Chapter 1.6.6.2.17.2.1 “Tab 'EtherCAT Slave - General' ” on page 3819*
- *Chapter 1.6.6.2.17 “EtherCAT configurator” on page 3815*

EtherCAT module

Tab 'EtherCAT Module - Startup Parameters'

Object: EtherCAT Module

The SDOs (Service Data Objects) or IDNs that transmit specified parameters to the device at the system start are defined on this tab for the current module.

The object directory with the required data objects is described in the EtherCAT XML description file or in an EDS file that is referenced in the XML file.

Requirement: The device supports *CAN over EtherCAT* or *Servodrive over EtherCAT*

Some modules have their own start parameters which are displayed on the tab. The parameters can be modified there. Likewise, the parameters are also displayed in the slave, but they are blocked there.

Table 710: SDO table

List of SDOs or IDNs The order (from top to bottom) in the SDO table specifies the order in which the SDOs are transferred to the module.	
“Line”	Line number
“Idn”	
“Bit length”	Bit length of the SDO
“Abort on Error”	☑: In case of error, the transfer is aborted with an error status.
“Jump to Line on Error ”	☑: The transfer is resumed with the SDO at the specified *“Line”* in case of error.
“Next Line ”	☑: The transfer is resumed with the SDO at the next line.
“Comment”	Input field for comments
“Move Up”	Moves the selected line upwards by one line

"Move Down"	Moves the selected line downwards by one line
"Add"	Opens the *"Select Item from Object Directory"* dialog. In this dialog you can change the parameters of the SDO before the SDO is added to the configuration. By specifying new index/subindex entries, a new object can be added to the SDO that is not yet described in the EDS file. This is useful if only an incomplete object directory or none at all exists.
"Delete"	Removes the selected entry.
"Modify"	Opens the *"Select Item from Object Directory"* dialog for changing the parameters of the selected SDO or IDN in the table

Servodrive over EtherCAT

Table 711: "Select Item from Object Directory"

List of available object directory entries as defined in the XML file.	
Column *"Idn"*	
Column *"Base Value"*	Base value of the IDN. Editable (double-click to open)
Input fields	
"IDN"	• *"S"* • *"P"*
"PSet"	By specifying new *"PSeT"*/*"Offset"* entries, a new object can be added to the IDN that is not yet described in the XML file. This is useful if only an incomplete object directory or none at all exists.
"Offset"	By specifying new PSet/Offset-entries, a new object can be added to the IDN that is not yet described in the XML file. This is useful if only an incomplete object directory or none at all exists.
"Bit length"	List box for selecting the bit length
"Value"	List box for selecting the value
"Channel"	If the object has multiple subobjects, then this list box is displayed automatically.
"As list"	Parameters are loaded as a list. The first four bytes indicate the length. ☑: The length is calculated automatically.

"CAN over EtherCAT "

Table 712: "Select Item from Object Directory"

List of available object directory entries as defined in the EDS file.	
Column *"Flags"*	Display of access flags: RW (read/write), RO (read only), WO (write only)
Column *"Base Value"*	Editable (double-click to open)
Input fields	
"Name"	Input field for displaying and changing the name
"Index: 16#"	By specifying new index/subindex entries, a new object can be added to the SDO that is not yet described in the EDS file.
"Subindex: 16#"	
"Bit length"	Range of values of the object

"Value"	Each value may be max. one byte (0-255). It can also be a hexadecimal in IEC syntax (for example, 16#ad). If the *"Byte array"* option is enabled, then the values have to be specified as a comma-separated list (for example, 1,2,3,4).
"Full access"	The complete object is written with one access and all subindexes are set at the same time. The time needed for the transfer is reduced because not every subindex has to be transferred individually.
"Byte array"	☑ Values can be specified as a comma-separated byte array.

See also

- *Chapter 1.6.6.2.17.2.1 "Tab 'EtherCAT Slave - General' " on page 3819*
- *Chapter 1.6.6.2.17 "EtherCAT configurator" on page 3815*

Bus Cycle Task - EtherCAT

General information

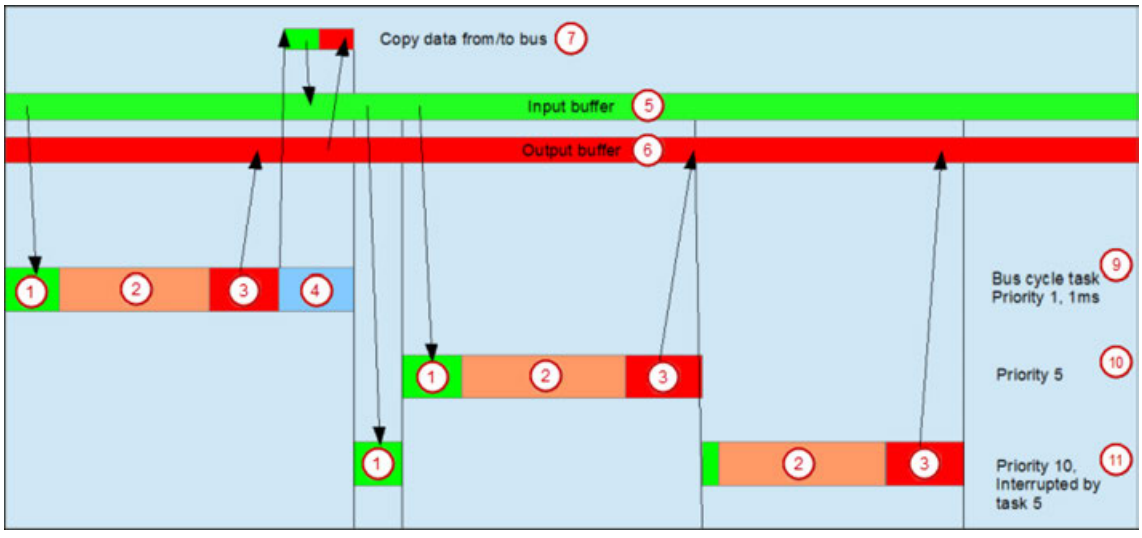
By "bus" it means all fieldbuses including I/O bus. There is no bus cycle task for Modbus because it is controlled by POUs. Modbus does not provide IO mapping.

It's recommended to define a dedicated bus cycle task for each fieldbus configured in the project. It's strongly recommended not to use "unspecified" in the "*"PLC Settings"*" to avoid unexpected behavior. The task defined in *"PLC Settings"* determines the bus cycle task of I/O bus and, depending on the configuration, of the additional fieldbuses (the setting is by default inherited).

Especially in case of EtherCAT, a dedicated bus cycle task should be used which is not shared with other fieldbuses. If *[unspecified]* is set in *"PLC Settings"*, the EtherCAT task might be automatically used by other fieldbuses, potentially causing EtherCAT task processing to fail. This should be avoided by specifying a task different to the EtherCAT task in *"PLC Settings"*.

As a rule, for each IEC task the used input data is read at the start of each task and the written output data is transferred to the I/O driver at the end of the task . The implementation in the I/O driver is decisive for further transfer of the I/O data. The implementation is therefore responsible for the timeframe and the specific time when the actual transmission occurs on the respective bus system.

Other tasks copy only the I/O data from an internal buffer that is exchanged only with the physical hardware in the bus cycle task.



```
(1) Read inputs from input buffer        (2) IEC task
(3) Write outputs to output buffer       (4) Bus cycle
(5) Input buffer                         (6) Output buffer
(7) Copy data to/from bus
(9) Bus cycle task, priority 1, 1 ms
(10) Bus cycle task, priority 5
(11) Bus cycle task, priority 10, interrupted by task 5
```

Using tasks

The *"Task Deployment"* provides an overview of used I/O channels, the set bus cycle task, and the usage of channels.

WARNING!

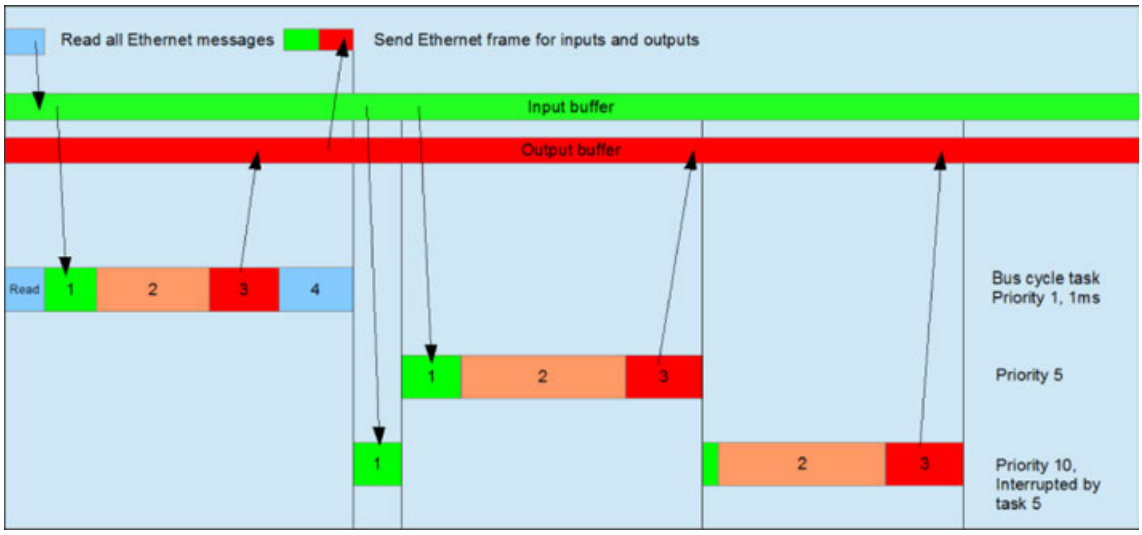
If an output is written in various tasks, then the status is undefined, as this can be overwritten in each case.

When the same inputs are used in various tasks, the input could change when a task is processed. This happens if the task is interrupted by a task with a higher priority and causes the process map to be read again. Solution: At the beginning of the IEC task, copy the input variables to variables and then work only with the local variables in the rest of the code.

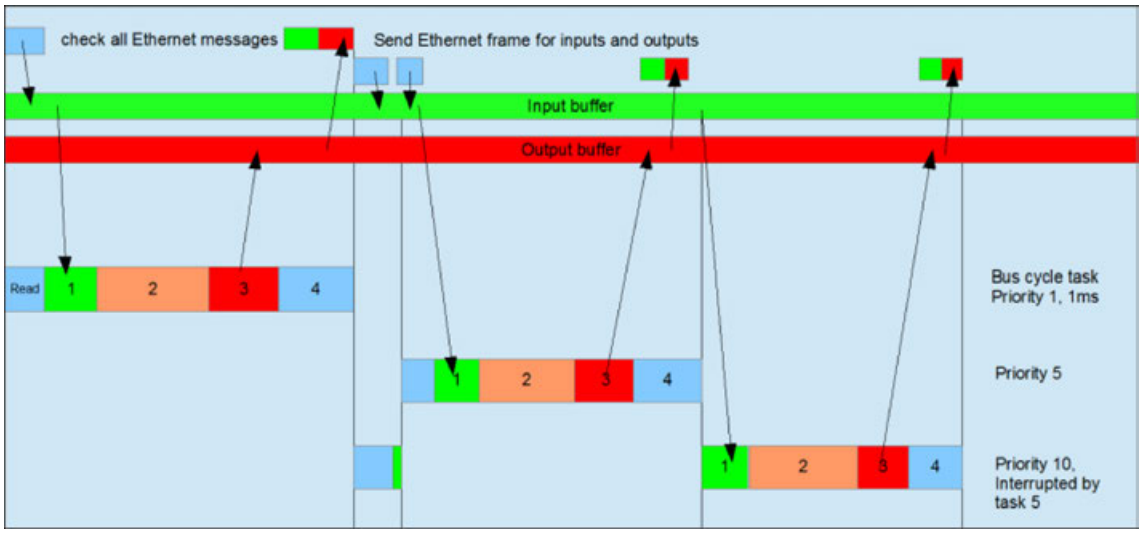
Conclusion: Using the same inputs and outputs in several tasks does not make any sense and can lead to unexpected reactions in some cases.

Behavior of the bus cycle for EtherCAT

Before the IEC inputs are copied, the pending network messages of the last cycle are read.



When the *"Messages per task"* option is enabled in the settings of the EtherCAT Master, additional telegrams are transmitted to the devices employed per task and input or output employed. Channels that are used in a slow task are also transmitted less frequently. As a result, the bus load can be reduced.



See also

- *Chapter 1.6.6.2.17.1.1 “Tab 'EtherCAT Master - General' ” on page 3816*

1.6.6.2.18 PROFINET IO Configurator

PROFINET IO (Process Field Network) is an industrial Ethernet standard widely used in the field of manufacturing and process automation. It is managed by the user organization PI (PROFIBUS&PROFINET International) and is considered the successor of PROFIBUS (see *https://www.profibus.com/*).

PROFINET IO controller

Controller – General

Object: PROFINET IO Controller

The PROFINET IO controller, like the slaves, is identified by the station name. For AC500 Communication Modules, you can also configure the IP settings here. Otherwise the settings apply from the superordinate Ethernet node.

“Station name”	The station name of the device. It is used for unique identification of the device in the network.

Table 713: IP Parameters

“IP address”	Note: Available for AC500 Communication Module only. If you insert the controller below an Ethernet adapter, then you have to define the IP parameters in the dialog of the Ethernet adapter.
“Subnet mask”	
“Default gateway”	

Table 714: Default Slave IP Parameter

“First IP address”	Range of IP addresses that CODESYS uses by default when inserting PROFINET IO devices into the device tree. If you use the *“Auto-IP”* function in the scan dialog, then IP addresses are also used from this range. The next free IP address is selected here.
“Last IP address”	
“Subnet mask”	
“Default gateway”	

Table 715: IO Provider / Consumer Status

"Application stop --> Substitute values"	When the user stops the application, the provider state is set to "BAD". Then the slaves set the inputs and outputs to predefined substitute values. For more information, see "CODESYS default values – PROFINET IO substitute values" at the end of this chapter.
"Add to I/O mapping"	☑: The incoming status information is added to the I/O mapping for all modules; provider state for the input data and consumer state for the output data.
"Substitute input data"	● *"Zero"* ● *"Last valid value"*

Table 716: Port Data

"Peer station/port"	Neighboring device with port that is connected to this port. You can accept this setting in the *"PROFINET IO Controller Topology"* tab.
"Check cable length"	Length of the network cable (in meters) ● < 10 ● < 25 ● < 50 ● < 100 When the cable length is specified, it is checked when the controller is powered up. An incorrect cable length causes an error message.
"Check MAU type"	Type of network cable

Table 717: Watchdog

"Activate"	Note: Available for AC500 Communication Module only. ☑: If the AC500 Communication Module firmware is not set within the given time (for example, in the case of an exception error in the application), then it is reset. The connection is terminated and the slaves switch to their substitute values. The defaults for the watchdog originate from the device description.

See also

- *Chapter 1.6.6.2.18 "PROFINET IO Configurator" on page 3832*

CODESYS default values – PROFINET IO substitute values

PROFINET IO devices set their inputs and outputs to predefined substitute values when there is an interruption. These values are defined in the field device in contrast with default values. These values are usually zero, but specific substitute values can also be configured depending on the device.

The substitute values are set in the following cases:

- The connection is interrupted.
- The controller sets the provider state for the incoming data to "BAD".
- Other interruptions occur (for example, exception in host application, incorrect parameterization)

If the *"Application stop --> Substitute values"* option is enabled, then the controller sets the provider states to "BAD" at application stop. In this case, the slaves set their substitute values. All incoming data from the controller is ignored (including default values).

If the default values defined in the application should be set for an application stop, then you have to disabled this option. Moreover, you should select the *"Update IO while stop"* option (in the *"PLC Settings"*). Otherwise, the CODESYS PROFINET IO controller is stopped.

PROFINET IO Controller - Bus Cycle Task

General information

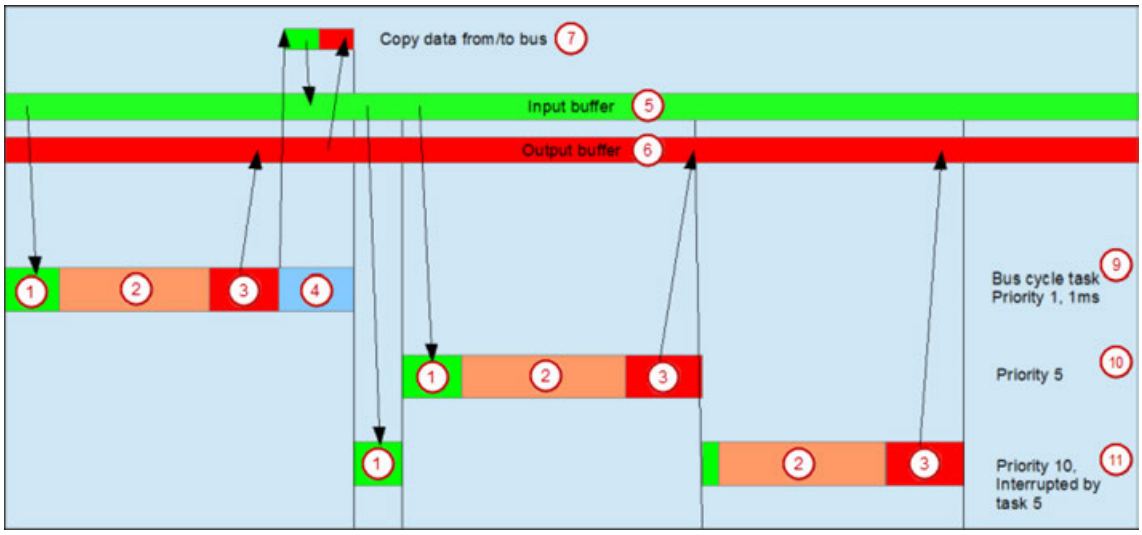
By "bus" it means all fieldbuses including I/O bus. There is no bus cycle task for Modbus because it is controlled by POUs. Modbus does not provide IO mapping.

It's recommended to define a dedicated bus cycle task for each fieldbus configured in the project. It's strongly recommended not to use "unspecified" in the "*"PLC Settings"*" to avoid unexpected behavior. The task defined in *"PLC Settings"* determines the bus cycle task of I/O bus and, depending on the configuration, of the additional fieldbuses (the setting is by default inherited).

Especially in case of EtherCAT, a dedicated bus cycle task should be used which is not shared with other fieldbuses. If *[unspecified]* is set in *"PLC Settings"*, the EtherCAT task might be automatically used by other fieldbuses, potentially causing EtherCAT task processing to fail. This should be avoided by specifying a task different to the EtherCAT task in *"PLC Settings"*.

As a rule, for each IEC task the used input data is read at the start of each task and the written output data is transferred to the I/O driver at the end of the task . The implementation in the I/O driver is decisive for further transfer of the I/O data. The implementation is therefore responsible for the timeframe and the specific time when the actual transmission occurs on the respective bus system.

Other tasks copy only the I/O data from an internal buffer that is exchanged only with the physical hardware in the bus cycle task.



(1) Read inputs from input buffer
(2) IEC task
(3) Write outputs to output buffer
(4) Bus cycle
(5) Input buffer
(6) Output buffer
(7) Copy data to/from bus
(9) Bus cycle task, priority 1, 1 ms
(10) Bus cycle task, priority 5
(11) Bus cycle task, priority 10, interrupted by task 5

Using tasks

The *"Task Deployment"* provides an overview of used I/O channels, the set bus cycle task, and the usage of channels.

WARNING!

If an output is written in various tasks, then the status is undefined, as this can be overwritten in each case.

When the same inputs are used in various tasks, the input could change when a task is processed. This happens if the task is interrupted by a task with a higher priority and causes the process map to be read again. Solution: At the beginning of the IEC task, copy the input variables to variables and then work only with the local variables in the rest of the code.

Conclusion: Using the same inputs and outputs in several tasks does not make any sense and can lead to unexpected reactions in some cases.

PROFINET IO bus cycle behavior

PROFINET IO does not provide any additional settings. Its functionality corresponds to the general description.

PROFINET IO device

Device – General

Object: PROFINET IO Device

In this dialog, you configure a communication link (PROFINET IO: application relation) to a PROFINET IO Field Device.

For all settings in the present dialog, the device description determines if the values here are editable and the values that are predefined or possible.

“Station name”	The station name of the device. It is used for unique identification of the device in the network.
“Station status”	32-bit error code compliant with the PROFINET IO specification. In case of error, the status is provided here, for example, when establishing a connection fails or a link is interrupted. A description is also displayed.
“IP Parameters”	
“IP address”	The IP settings of the device. Set when establishing the connection to the controller.
“Subnet mask”	
“Default gateway”	
“Communication Settings”	
“Send clock (ms)”	Send clock (in milliseconds).
“Reduction ratio”	Scaling factor The send cycle is defined by *“Send clock”* * *“Reduction ratio”*. Therefore, a *“Send clock”* of 1ms and a *“Reduction ratio”* of 4 means that I/O data is sent every 4ms.
“Phase”	With a *“Reduction ratio”* of `n`, the send cycle is divided into phases 1 to n (where data is sent in one phase only). You can determine in which phase the data is sent for the purpose of load distribution. If *“Send clock”* = 1 and *“Reduction ratio”* = 4 (as in the example above), then you could configure phases 1–4. For four slaves with this send clock and reduction ratio settings, you could assign one of the four phases to each of the four slaves. In this way, only one data packet is sent in each of the four phases of the send cycle and the load is distributed equally.

“Watchdog (ms)”	Monitoring time. A multiple of the send cycle (send cycle = *“Send Clock”* * *“Reduction Ratio”*). Possible values: 3 ms – 1920 ms. A connection is terminated when the controller or the PROFINET IO Device does not receive I/O data from the communication peer within this time period. The device enters failure mode and switches the outputs to substitute values.
“VLAN ID”	VLAN identifier: Number between 0 and 4095 for VLAN type 802.1Q. Note: For newer devices compliant with PROFINET IOspecification V2.3, only "0" is still permitted.
“RT class”	If available, you can select the required RT class from the list (real-time communication).
“User Parameters”	
“Set All Default Values”	CODESYS resets all settings to default values (see default value column) from the GSDML file.
“Read All Values”	CODESYS reads the current values from the device and updates them in the editor.
“Write All Values”	CODESYS writes the current values from the editor to the IO device. Not all IO devices support parameter updates in run mode. If not, then an error message is displayed.

See also

- *Chapter 1.6.6.2.18 “PROFINET IO Configurator” on page 3832*

PROFINET IO - Module

Module – General

Object: PROFINET IO Module

Table 718: “Module Information”

“ID number”	Identification of the module (from the device description).
“Slot number”	Position of the I/O module below the I/O device, starting at "1" for the first module and incremented for each additional module. This results automatically from the current structure in the device tree.

Table 719: “User Parameters”

“Set All Default Values”	CODESYS resets all settings to default values (see default value column) from the GSDML file.
“Read All Values”	CODESYS reads the current values from the device and updates them in the editor.
“Write All Values”	CODESYS writes the current values from the editor to the I/O module. Not all I/O modules support parameter updates in run mode. If not, then an error message is displayed.

See also

- *Chapter 1.6.6.2.18 “PROFINET IO Configurator” on page 3832*

PROFINET IO - Field Device

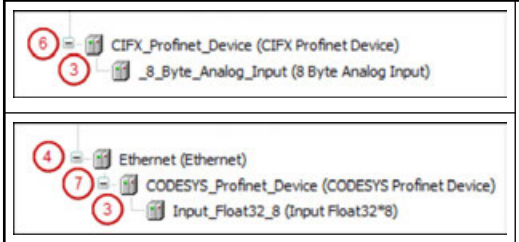
The configuration of the PROFINET IO field device consists of the device itself as well as the modules inserted below.

CODESYS provides two different PROFINET IO field devices:

- A variant especially for Communication Module CM579-PNIO
- A variant which is hardware-independent, the CODESYS PROFINET IO field device This variant runs on any number of Ethernet adapters and is also available in a purely programmatically configurable variant.

When inserting the Ethernet-based CODESYS field device, two tasks are created implicitly that are required by the PROFINET IO communication stack.

- *"Profinet_CommunicationTask"*: This task includes the acyclic communication services, such as establishing connections and diagnostics. These services are not time-critical due to very weak real-time demands. Therefore the task is low priority.
- *"Profinet_IOTask"*: This is where the actual PROFINET IO real-time data exchange takes place. Pending I/O data packets are processed in each cycle (see Slave Configurator: *"Send clock"*). Therefore, a cycle time of 1ms is required (for 1ms send clock).

	(6) ↳ *Chapter 1.6.6.2.18.2.2 "Field Device NetX – General" on page 3838* (3) ↳ *Chapter 1.6.6.2.18.1.4.1 "Module – General" on page 3836* (4) Ethernet adapter (7) ↳ *Chapter 1.6.6.2.18.2.1 "Field Device – General" on page 3837*

For maximum IO performance with minimum delay when reading/writing, I/O data can be updated in this task (→ insert own POU that updates IOs in this task). No blocking or persisting operations should be executed in the IO task, such as visualization or file access. If the task is blocked too long, then the watchdog cancels the connection for communicating with the slave (see Slave Configurator: *"Watchdog"*).

NOTICE!
We recommend that you activate the *"Refresh I/Os in Stop"* option in the PLC settings. Otherwise the communication is canceled when the application stops at a breakpoint.

See also

- *Device Editor Options*
- ↳ *Chapter 1.6.6.2.18.2.1 "Field Device – General" on page 3837*
- ↳ *Chapter 1.6.6.2.18.2.2 "Field Device NetX – General" on page 3838*

Field Device – General

Object: PROFINET IO Field Device

The tab displays the basic communication settings.

According to the PROFINET IO standard, the PROFINET IO device is responsible for the IP settings of the used adapter. It has to save remanent IP settings and be able to reset or modify at the request of the controller (IP=0.0.0.0). Initial state (Reset to factory) is with deactivated IP suite (IP=0.0.0.0).

To allow this full reset of the IP configuration, some settings have to be done on most systems (see chapter ↳ *Chapter 1.6.6.2.18 "PROFINET IO Configurator" on page 3832*).

However, if the PROFINET IO device is a programmable logic controller and connected with the (CODESYS) programming environment via one and the same Ethernet adapter, then modifying and resetting the IP address is interruptive (connection termination between IDE and PLC). Therefore, one of the modes is provided that deviates from the standard (*"Use project parameters"*).

Table 720: "IP and Name Assignment"

"Use remanent data"	The IP settings and the station name of the file are used. The file is stored in the file system. The data is set by the controller and saved to a file by the device.
"Use project parameters"	The IP settings and the station name of the project are used (settings of the Ethernet adapter). This option must be selected for Windows, VxWorks, and WinCE, because changing the IP address is not possible for these systems.
"Station name"	Station name of the PROFINET IO Device

Table 721: "IO Provider / Consumer Status"

"Use incoming"	☑: The I/O data for the provider and consumer states is generated which is received by the controller.
"Use outgoing"	☑: The I/O data for the provider and consumer states is generated which is sent to the controller.
"Substitute values"	The substitute values for the output data become active when the corresponding provider status (Output Data PS) is set to `Bad`. The output data is sent by the controller and copied to the %I area of the Profinet modules. The following options are available for the substitute values: ● *"Inactive"*: The outputs are set to "inactive" (example: 0). ● *"Last value"*: The output data retains the last valid value (provider status = `GOOD`). The value is retained even if the connection to the controller has been interrupted.

In online mode, the station name and the IP settings are displayed "Status" tab.

See also

- *Chapter 1.6.6.2.18 "PROFINET IO Configurator" on page 3832*

Field Device NetX – General

Object: PROFINET IO Field Device

The tab displays the basic communication settings.

"Use remanent data"	The IP settings and the station name of the file are used. Initially, the IP address is `0.0.0.0` according to the standard and the station name is blank. When a controller sets these values with the "store remanent" option, then they are stored here.
"Use project parameters"	When starting the device, the values defined in the project for IP configuration and station name are always used initially.

“Station name”	The name of the device in the network. Note: The station name and the IP settings can deviate from the (default) settings configured in the project. They can be set by the controller in runtime mode and in some cases stored persistently (this means when this is specified for the controller in the `Set IP` or `Set station name` commands). After a restart, the device is configured with these values as long as the *“Use remanent data”* option is set.
“IP address”	Initial IP settings Caution: This data can be modified by the PROFINET IO Controller. The remanent data is stored on the file system of the controller. After the controller is restarted, this stored data goes into effect. The settings here are then ignored.
“Subnet mask”	
“Default gateway”	

In online mode, the station name and the IP settings are displayed “Status” tab.

See also

- *Chapter 1.6.6.2.18 “PROFINET IO Configurator” on page 3832*

1.6.6.3 Protocols and special servers

1.6.6.3.1 IEC60870-5-104 (Telecontrol)

General information IEC60870

Introduction

The implemented IEC60870-5-104 protocol allows link-ups between AC500 CPUs with onboard Ethernet and external systems. The link-up takes place via the onboard Ethernet interface of the CPU. The telecontrol protocol according to IEC60870-5 is used.

The CPU can work as both control station and substation. In control direction, setpoints and commands can be set; in monitoring direction the control station sends status values, real values and discrete values to the substation. Via general inquiry, the substation requests the control station to send all status values, real values and discrete values. Otherwise, these values are sent by the control station on a change-driven basis, cyclically or when triggered by an application. Status values, real values and discrete values may contain timestamps. These are filled in with the time of the process station when sent. The CPU can time-synchronize the telecontrol link.

A module accepts the configuration of the physical interface (link layer) and the general protocol parts (application layer).

Send and receive blocks are available for data exchange. These blocks exist for the IEC60870-5 data types setpoint value, command value, double command value, status value, double status value, real value and discrete value. The inputs/outputs of the send and receive blocks are combined with the signals to be communicated. See documentation of IEC60870 library for more information.

For a better understanding on how events are processed on a AC500 V3 PLC, refer to the *application example*.

Limits of supported devices

AC500 V3 (Standard):

- PM5630: Support of 5 control stations and/or substations with 1.000 information objects overall on ETH1 and ETH2.
- PM5650: Support of 10 control stations and/or substations with 5.000 information objects overall on ETH1 and ETH2.
- PM5670: Support of 20 control stations and/or substations with 10.000 information objects overall on ETH1 and ETH2.

Limits of supported devices

AC500-eCo V3:

-
- PM5012/ PM5052: no support of IEC60870-5-104 protocol.
- PM5072: Support of 5 substations with up to 1.000 information objects overall on ETH1 and ETH2. IEC60870 control stations are not supported.

Data flow control

Each send or receive block can only process one data message. Ideally, new data are available at each user task run-through or new data can be sent.

If the output OV (send block only) indicates TRUE, the function block computes more quickly than the data can be sent. This can happen if the receive block is not computed quickly enough and has thus not collected all the data.

Alternatively, this block sends either cyclically or if the input value is changed. Ideally, the topical data can be sent via the telecontrol link in connection with every user task run-through.

Data integrity

With IEC60870-5 protocol, a distinction is made between data transmission in the monitoring direction (status values, real values, discrete values) and in the control direction (commands and setpoints).

All data transmissions are acknowledged from the link communication level by the receiver. This acknowledgement is not sent to the sender of the data in every telecontrol link.

For data transmission in control direction, additional acknowledgement (e.g. ACTTERM) is possible. These acknowledgements are not sent by every telecontrol link either. For safe data transmission, it is necessary, in such cases, to configure data readback. The receiver then sends the data received back to the sender via the corresponding send blocks.

Information in the monitoring direction is acknowledged by the receiver on the lowest communication level (link level) when received. This acknowledgement is generated by the telecontrol head itself with some telecontrol heads. In the event of overload/overrun, a data message may be lost. For data in the control direction, so-called ACTTERM acknowledgement can be used. This additional acknowledgement is sent back to the sender when the data have been executed in the process. If data are to be sent in the monitoring direction with guaranteed transmission, it is necessary to read back the sent value via another variable and, after observing a monitoring time, resend in the event of an error.

Data transmission

Send blocks

On the basis of the communication protocol, it is sensible to restrict the data types at one send block to one type. Therefore, there are 5 types of send blocks: send of status values, commands, real values, setpoints and discrete values. These types are mapped to the IEC1131 data types BOOL, REAL and DINT. See documentation of IEC60870 Library for more information.

Operating modes of the send blocks

The send blocks know three operating modes to send their data:

- Caused by request pin (SEND)
- Send in connection with a change of data (AUTO)
- Cyclic send of data (CYCLE)

Send via request pin

The SEND signal is evaluated on the rising edge, the RDY signal remains applied for one computation cycle. If a rising edge is generated again at the SEND signal although no acknowledgement has yet been received from the receiver, the OV pin is set in order to indicate that an overrun has happened. The evaluation of the receive acknowledgement is carried out before the evaluation of whether transmission is to take place. This means, assuming that there is an appropriately fast telecontrol link, that in connection with change-driven and cyclic transmission, a transmission job can be sent in connection with every computation of the block. In connection with send via the request pin it is possible to send only in connection with every second computation (send takes place only with a rising edge).

Change-driven send of data

Data are always sent when the value of the input variables changes. When changes take place, there is an internal simulation that the SEND pin changed from 0 to 1.

In order to prevent unnecessarily frequent send in the event of mild fluctuations in the input value, a threshold value can be configured for real values and setpoints. The input value is not sent until it differs positively or negatively from the value last sent by more than the threshold value.

If the input value changes again although no acknowledgement has yet been received from the receiver, the OV pin is set in exactly the same way as in connection with send via the request pin. If an error occurs during send, the job is automatically retried until the value has been sent without error.

Cyclic send

The data are automatically sent after expiration of a configurable cycle time (SCANDOWN). This cycle time is indicated in multiples of the task cycle time in which the block is computed. In this operating mode, an overrun error can occur if the transmission is faster than the response time of the receiver. For setpoints, it is necessary to ensure that an acknowledgement is generated by the receiver which is not sent until the setpoint is accepted. The send block is not ready for transmission again until after this acknowledgement has been received.

Receive blocks

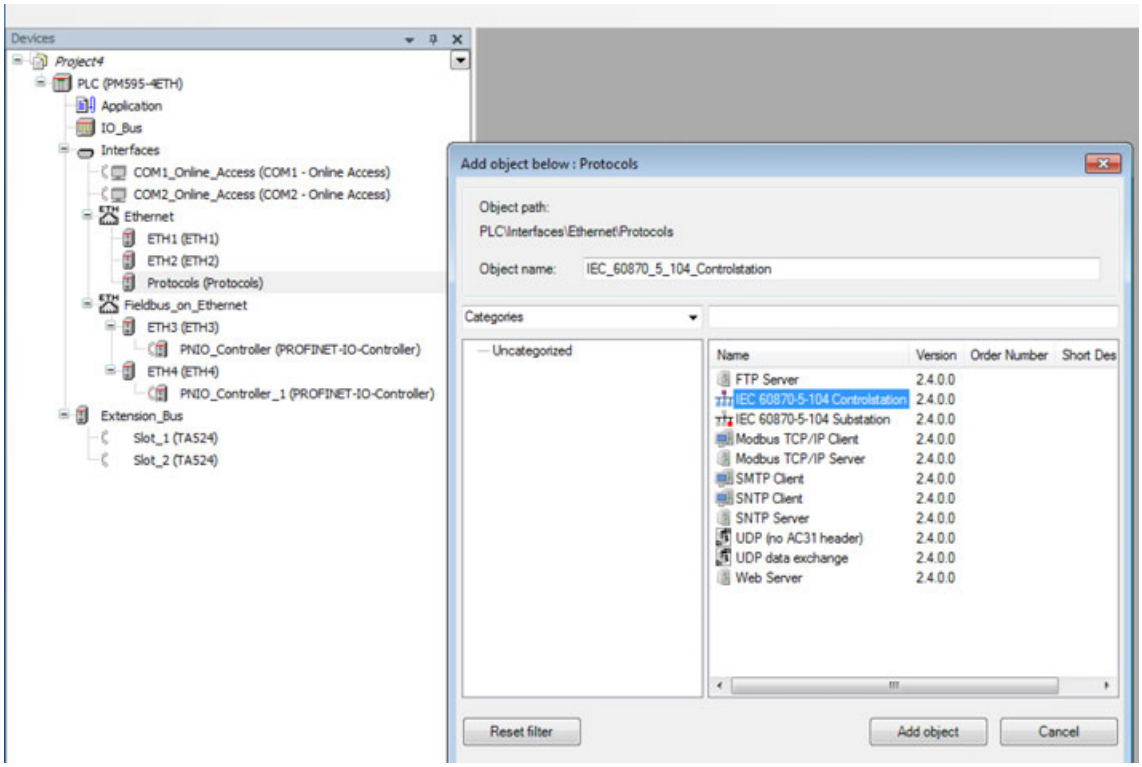
In receive direction, the jobs enter the device module via the interface. The device module selects the correct receive block using the telecontrol address. To this end, during installation the receive blocks pass their parameterized telecontrol addresses to the device module. The device module stores the data received and the receive blocks make the data available at their output pins in connection with the next computation of the user task.

Configuration

Configuration changes >= Automation Builder 1.1/CBP 2.4

The IEC 60870 protocol allows link-ups between AC500 CPUs with onboard Ethernet (e.g. PM595-4ETH and PM591-2ETH) and external systems.

The link-up takes place via the onboard Ethernet interface of the CPU. As of Automation Builder Version 1.1 telecontrol is also supported for CPUs that provide more than one Ethernet interface (e.g. PM595-4ETH and PM591-2ETH). This allows to use different Ethernet interfaces for IEC 60870 connections, hence, telecontrol configuration is changed. Further, as of this version terminology is aligned with IEC 60870 standard and provides additional features that are described in this chapter. For a description on principle telecontrol configuration.



For further information on configuration changes, see the following chapters:

Control and Substations ≥ CBP 2.4 ↬ *Chapter 1.6.6.3.1.2.2 “Control station and substation configuration” on page 3842*

Import Export ≥ CBP 2.4 ↬ *Chapter 1.6.6.3.1.2.1 “Configuration changes >= Automation Builder 1.1/CBP 2.4” on page 3841*

Validity Check of Configuration ≥ CBP 2.4 ↬ *Chapter 1.6.6.3.1.2.4 “Validity check of configuration” on page 3862*

Control station and substation configuration

The CPU can work as both, control station and substation.

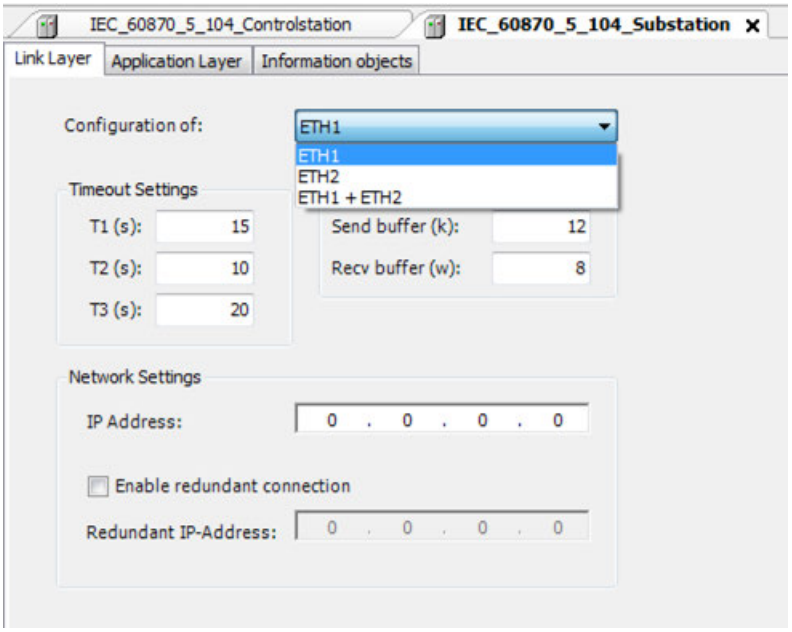
Control station	Client, master, controlling station: Synonyms for a higher-level station (central station, monitors other stations)
Control direction	Data transfer direction from the control station to the substation
Substation	Server, slave, controlled station: synonyms for a subordinate IEC 60870-5-104 telecontrol station (which is monitored)
Monitoring direction	Data transfer direction from the substation to the controlling station

Configure a control station in the device tree *PLC -> Interfaces -> Ethernet -> ETHx*:

1. Right-click *"ETHx ➔ Add objects"*.
2. Select the control station from the list and click *"Add object"*. Configure substations and further control stations in the same way. As of Automation Builder 1.1 any combination of control stations and substations can be configured, in due consideration of a total number of 10 stations.
3. Double-click the new control station node to open parameter configuration. In the Link Layer tab access to the Ethernet interface is configured.

> *As of Automation Builder 2.2.1 the V3 PLC telecontrol station objects from GVL IEC60870_5_104_Connection_GVL can only be used in Structured Text by adding the namespace of the GVL as prefix.*
>
> *Example:*
>
> *byteValue := IEC60870_5_104_Connection_GVL.IEC_60870_5_104_Control-station.Con;*

Tab link layer



In order to provide flexible usage of control stations and substations as of Automation Builder 1.1 configuration of substations has been changed. As several substations can be operated with several Ethernet interfaces, select the Ethernet interface to be used from the pull-down menu. Enter the IP address to the control station and if required to another control station (redundant connection). If no IP address is defined, the substation accepts connection to any control station.

> *This field is not available in the Link Layer tab of control stations. Selection of ETH interface is only possible for substations. The control station is always configured on both interfaces by default.*

Timeout settings

T1, T2, T3: The values for the connection control and message replication; timeout1/2/3.

Buffer settings

This parameter gives the maximum number of outstanding messages and acknowledgement behavior.

Send buffer (k): Maximum difference receive sequence number to send state variable.

Rec buffer (w): Latest acknowledge after receiving w I format APDUs.

Network settings

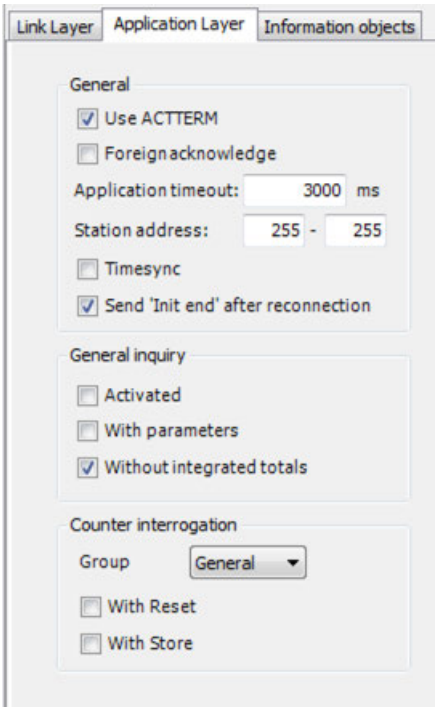
Network settings are available for control stations and for substations. The IP address of the control station and if available the IP address of another control station (redundant IP address) can be selected by the user.

For an overview on the configured Ethernet interfaces for the control stations and substations, double-click the *“Protocols”* node.

Tab application layer

Settings

The application layer is the communication layer with which the send and receive blocks work.



Use ACTTERM

This parameter concerns only setpoints and commands. If this parameter is checked, an acknowledgement with set ‘actterm’ is generated as reason for transmission at the time at which the receive block is computed and outputs its telecontrol data at its output pins. On transmission side, the data block awaits the reception of this ACTTERM acknowledgement and reacts with its corresponding output to the reception of this acknowledgement. For commands with execution time, the acknowledgement is generated when the command is terminated, for commands with continuous execution time and for setpoints, the acknowledgement is generated when the data are output to the output pins.

ForeignAck-nowledge

If this option is not enabled (default), a message that was sent is considered as ok as soon as transmission was successful. If you enable this option, a message that was sent is not considered as ok until a success message (foreign acknowledge) is returned from the receiver.

Application timeout This time indicates how long an acknowledgement will be awaited on the application level. An acknowledgement is generated only for commands and setpoints on the application level.

Station address The station address defines which station will be subject to a count query. The values define the 2 bytes for the common telecontrol address (Common addr.). The values concerned are as follows:

0: The station address is not used.

1...254: The count is queried on the station defined by the station address.

255: The count is queried on all accessible stations.

Timesync After each new establishment of a link and once per hour, a ‘coarse time synchronisation’ message is generated. This time synchronisation is only supported from AC500 to external systems. Time synchronisation from an external system to AC500 is not provided! Incoming time synchronisation messages are confirmed by the process station but not executed. Greenwich Mean Time (GMT) is used as the time for the synchronisation.

Send 'Init end' after reconnection After each establishment of a link or only in connection with the first establishment of a link and after reconfiguration, an init end message is generated. After the init end message, there is a general inquiry, if configured.

General inquiry

Activated This parameter concerns only real values, discrete values and status values. The device module generates a general inquiry message after each new establishment of a link. The other side then generates a message with the reason for transmission ‘general inquiry’ for every data point and subsequently an init end message. This procedure ensures that, in the event of a new establishment of a link, all data are available on the reception side in topical form.

With parameters If general inquiry is activated the parameter values are sent.

Without integrated totals With a general inquiry no integrated total values are sent.

Counter interrogation

Group General, 1 .. 4: The count inquiry is executed for a specific group of counters (1 .. 4). The count inquiry is executed for all groups of counters.

With reset The reset quality bit is sent along with the count inquiry.

With relocate The relocate quality bit is sent along with the count inquiry.

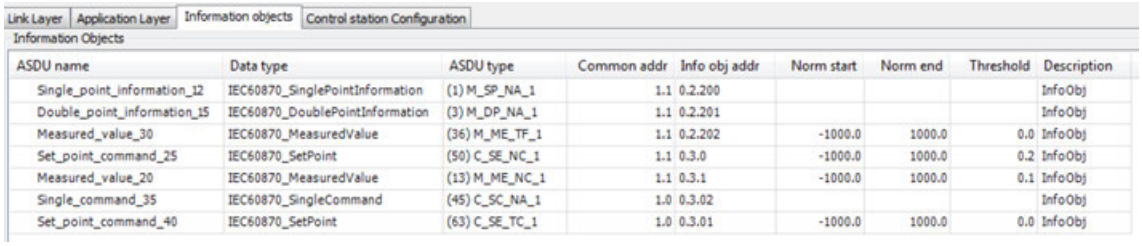
Tab information objects

Open the *“Information object”* tab to configure so called information objects and a common address (known as 'data points' and 'Global address' in former Automation Builder versions). In this tab different information objects and their services for transmission are defined. A data point or information object is identified via a system-wide unambiguous address containing a maximum 5 bytes.

1. Right-click in the empty view and select *“Add Information Object with ASDU”* to add a data group. Select the desired object from the list (e.g. M_SP_NA_1).

   ⇨ An information object with a corresponding ASDU (Application Service Data Unit) is created.

2. Configure the settings in the *“Information Object”* tab to your convenience.



ASDU name	Data type	ASDU type	Common addr	Info obj addr	Norm start	Norm end	Threshold	Description
Single_point_information_12	IEC60870_SinglePointInformation	(1) M_SP_NA_1	1.1	0.2.200				InfoObj
Double_point_information_15	IEC60870_DoublePointInformation	(3) M_DP_NA_1	1.1	0.2.201				InfoObj
Measured_value_30	IEC60870_MeasuredValue	(36) M_ME_TF_1	1.1	0.2.202	-1000.0	1000.0	0.0	InfoObj
Set_point_command_25	IEC60870_SetPoint	(50) C_SE_NC_1	1.1	0.3.0	-1000.0	1000.0	0.2	InfoObj
Measured_value_20	IEC60870_MeasuredValue	(13) M_ME_NC_1	1.1	0.3.1	-1000.0	1000.0	0.1	InfoObj
Single_command_35	IEC60870_SingleCommand	(45) C_SC_NA_1	1.0	0.3.02				InfoObj
Set_point_command_40	IEC60870_SetPoint	(63) C_SE_TC_1	1.0	0.3.01	-1000.0	1000.0	0.0	InfoObj

3. Double-click a table cell to modify pre-set values. For some ASDUs additional sub information objects can be configured. For this, right-click the already existing ASDU and select *“Add Information Object”* to selected ASDU option. This allows configuration of 16 data points at the most (depending on the ASDU type). With *“Remove Information Object”* the selected ASDU is deleted.

Description of the columns

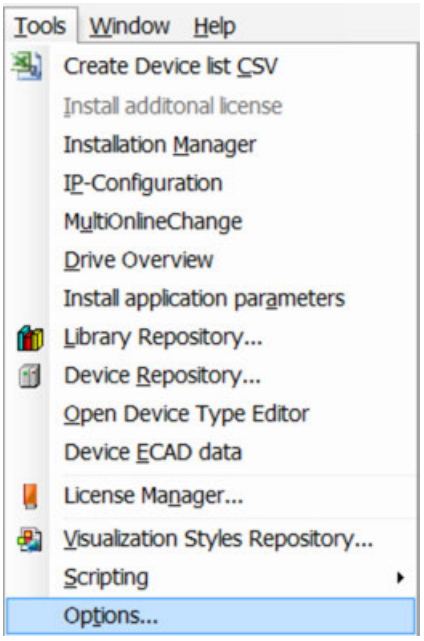
- ASDU name: node name of the information object (name of the ASDU).
- Data type: Data type of the ASDU.
- ASDU type: Type of ASDU.
- Common addr: Common address of the ASDU (known as 'Global Address' in former AB Versions). Byte 1/2 of the common telecontrol address of the block (range: 0...255).
- Info obj addr: Together with common address Info obj addr defines the endpoint (range: 0...255).
- Norm start: Low limit (0 %) of the normalized range for real values and setpoints.
- Norm end: High limit (100 %) of the normalized range for real values and setpoints.
- Threshold: Threshold limit beyond which a change of the input value referred to.
- Description: Table cell for free text. Use this field to describe your configuration settings e.g. differences between configuration variants.

Format of common addr and info obj addr

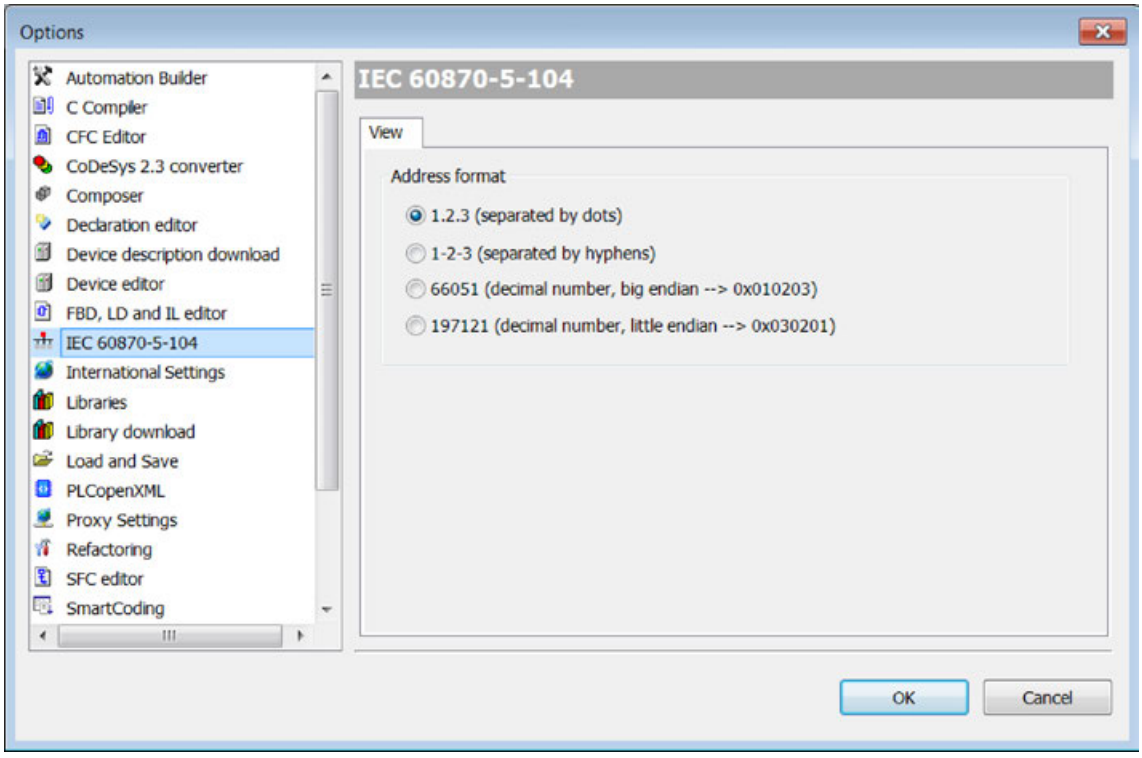
The following adress formats of your entries in the columns *Common addr* and *Info obj addr* of the Tab *Information Objects* are possible:

- 1.2 and 3.4.5 (Default format)
- 1-2 and 3-4-5
- 258 or hex 0x102 and 197637 or hex 0x30405
- 513 or hex 0x201 and 328707 or hex 0x50403

Previously you have to choose your preferred adress format:



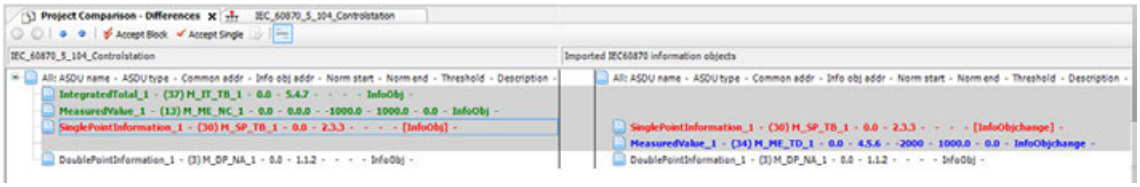
1. Click *"Tools"*and then *"Options..."*
   ⇨ The Window *Options* appears



2. Select *IEC 60870-5-104*, make your choice and click *OK*.

Import options of information objects

The User can accept the imported IEC60870 information objects as single change or change as block.



IEC60870-5-104 Multiple connections

An AC500 with more than one substation connection must be able to identify the corresponding control station clearly. This identification takes place exclusively via the control station's IP address. In order to make it possible for a non-redundant control station to have redundant access to a substation with 2 Ethernet connections. The local substation address is ignored during connection establishment.

In the following descriptions, the term station must not be confused with the individual connection. One station can have several connections. An IEC60870-5-104 communication always takes place between a control station and a substation. A control station can manage several substations and also simultaneously be a substation for one or several control stations. However, these must then be realized using different stations.

A PLC may **not** be configured for another PLC repeatedly as a substation or a control station unless a disjunctive Ethernet infrastructure is used for this.

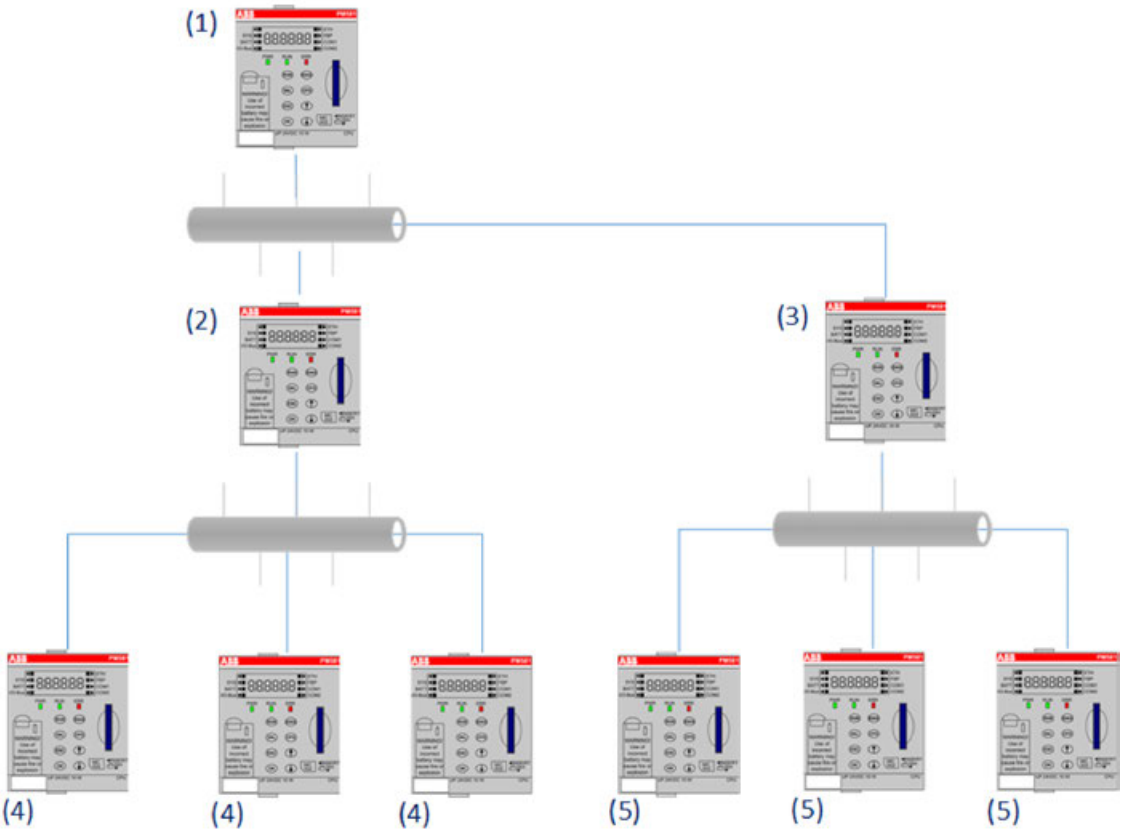
Redundant connections must be specified as such in the configuration.

An AC500 can be used only once as control station for another AC500, it makes no sense to use the same AC500 repeatedly as a control station for the same substation. Such a structure is configured as a redundant control station as long as only one AC500 exists as a control station per substation. However, this control station may have 2 IP addresses. Therefore, this configuration must either have the IP address 0.0.0.0 entered on the substation for the control station, meaning that all IP addresses are accepted and no other control station can access this AC500 or alternatively the possible control station addresses must be specified (ETH1 and ETH2).

Tree constellation

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

If you plan to control several substations with the AC500, they can be cascaded. This results in a tree structure.



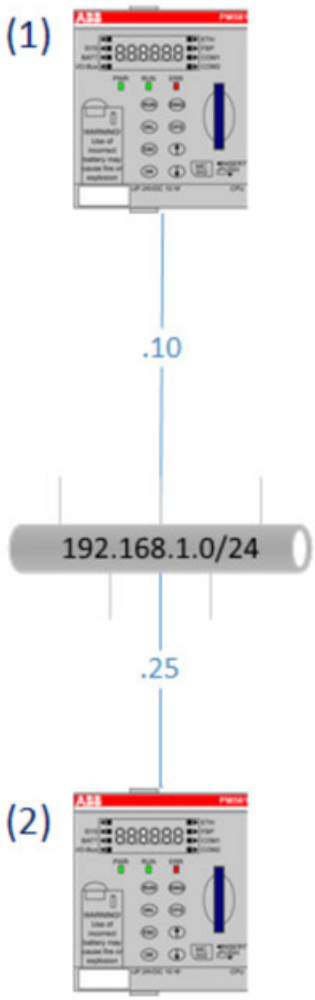
(1) 2 control stations
(2) Substation and 3 control stations
(3) Substation and 3 control stations
(4) Substation
(5) Substation

Structures of connections

In the following, the notation 192.168.1.0/24 is used for TCP/IP networks. Here, the figure /24 specifies the network mask with 255.255.255.0 and 192.168.1.0 describes the network. The valid addresses for this Class C network are 192.168.1.1 to 192.168.1.254! Only the last byte of the address is provided on the respective devices, with e.g. .10. This means that the respective device has the address 192.168.1.10.

Minimal structure

A control station with an Ethernet interface is connected to a substation with an Ethernet interface.



(1) Control station
(2) Substation

Configuration at control station

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

The respective substation IP address must be specified at the control station. For this, in the network settings of the control station (1) enter the IP address of the substation (in the example: 192.168.1.25). Option *“Enable redundant connection”* must be disabled.

Configuration at substation

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

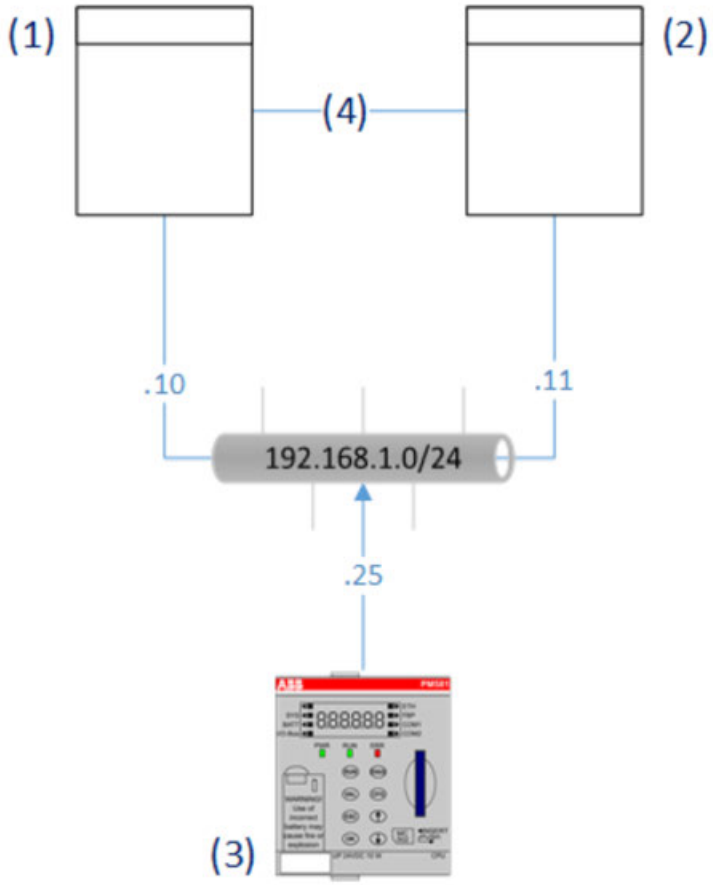
Either the control station IP address or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP address of the control station (in the example: 192.168.1.10). Option *“Enable redundant connection”* must be disabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station can be configured on this controller for a further substation.*

Minimal redundancy structure

The most simple redundant structure with an AC500 consists of a redundant control station (not AC500) which is connected to the AC500 substation with 2 different IP addresses. These redundant control stations must synchronize which control station is active.

Only one control station can be active at any given time.



(1) Control station 1A (Not AC500)
(2) Control station 1B (Not AC500)
(3) Substation
(4) Redundancy link

Configuration at control stations

The respective substation IP address must be specified at the control stations 1 and 2 (not AC500). For this, in the network settings of both control stations enter the IP address of the substation (in the example: 192.168.1.25).

Configuration at substation

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

Either the control station IP addresses or the general address 0.0.0.0 must be specified at the substation (3). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.1.11). Option *“Enable redundant connection”* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station can be configured on this controller for a further substation.*

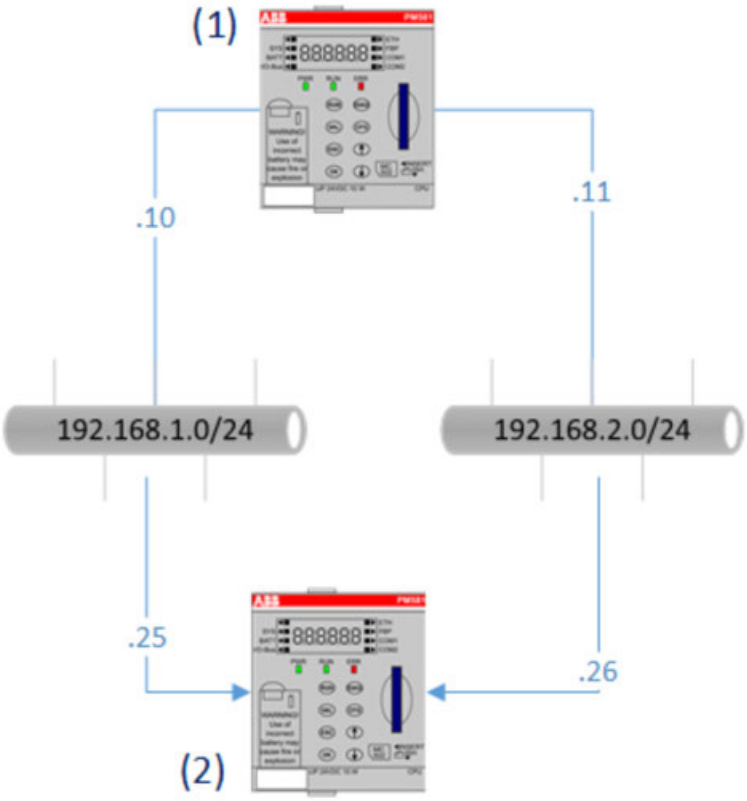
Network redundancy

For network redundancy a control station can reach a substation via 2 paths.

Both the control station and the substation can have 2 different IP addresses. Without special network routing, 2 separate networks should exist, within which both the substation and the control station each have 2 interfaces.

Possible variants of network redundancy are described in the following.

Network redundancy with 2 separate networks



(1) Control station with 2 redundant paths
(2) 1 Substation with 2 Ethernet interfaces

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.2.26). Option *"Enable redundant connection"* must be enabled.

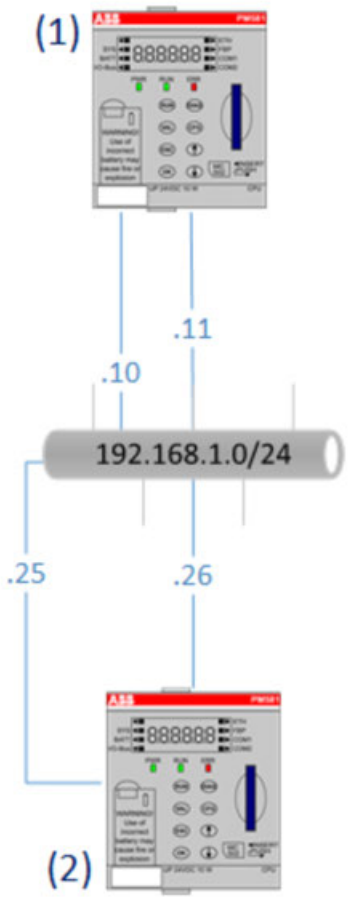
Configuration at substation

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Network redundancy with 1 network and 2 Ethernet ports in substation



(1) Control station with 2 paths to reach substation
(2) 1 Substation with 2 Ethernet interfaces

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.1.26). Option *"Enable redundant connection"* must be enabled.

Configuration at substation

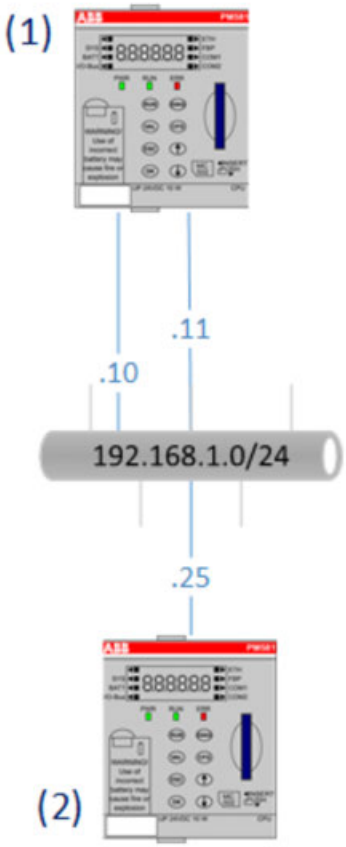
PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Network redundancy with 1 network and 1 Ethernet port in substation

> *No online redundancy.*
>
> *Only one connection will be established.*



(1) Control station with 2 paths to reach substation
(2) 1 Substation with 1 Ethernet interface

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 0.0.0.0). Option *"Enable redundant connection"* must be disabled.

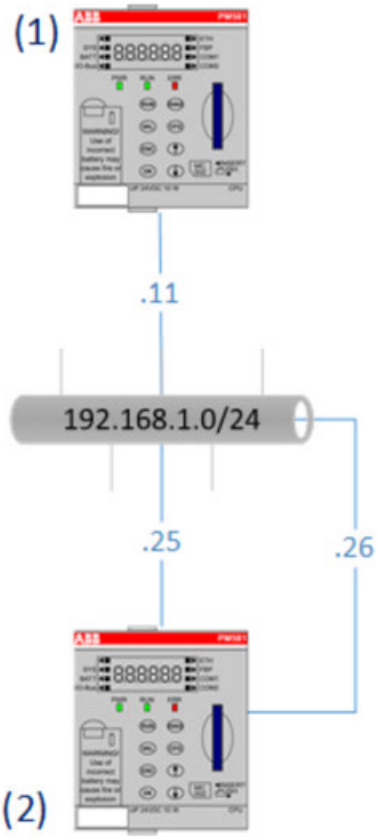
Configuration at substation

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Network redundancy with 2 Ethernet ports in substation



(1) Control station with 2 paths to reach substation
(2) 1 Substation with 2 Ethernet interfaces

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.1.26). Option *"Enable redundant connection"* must be enabled.

Configuration at substation

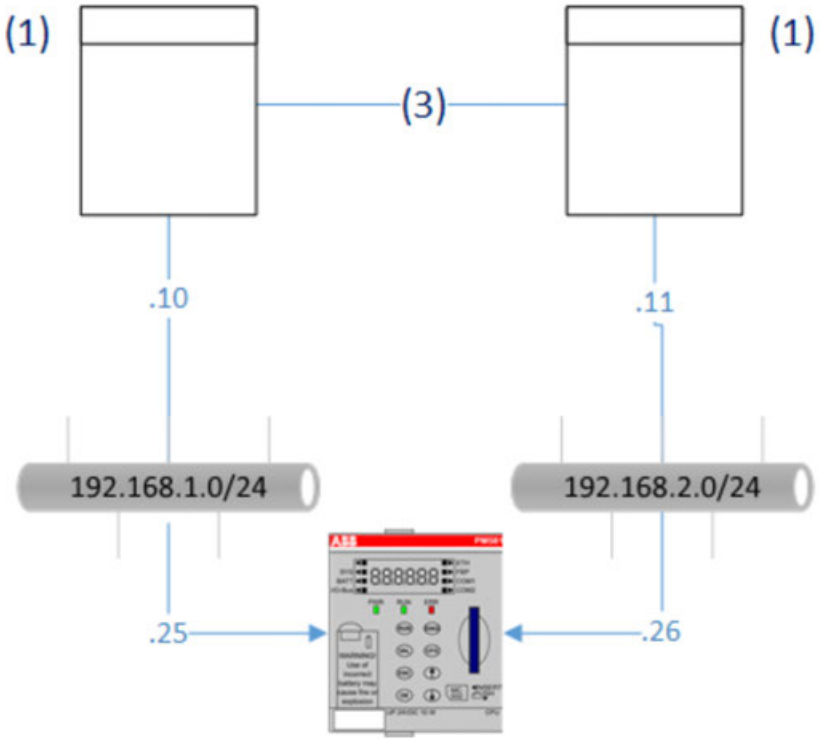
PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.11 and 0.0.0.0). Option *"Enable redundant connection"* must be disabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Full control station redundancy

A control stationcan consist of two fully redundant units (not AC500s), which are connected via a redundancy link. These control stations must ensure that only one of them at a time is actively connected to the substation and communicates with it. The inactive control station, however, can establish non-active connection with a substation and monitor it with keep alive packages.



(1) 2 redundant Control stations (Not AC500)
(2) 1 Substation with redundant Control station and 2 Ethernet interfaces (2nd port)
(3) Redundancy link

Configuration at control stations

The substation's IP address must be specified at the control stations (1) (not AC500). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.2.26).

Configuration at substation

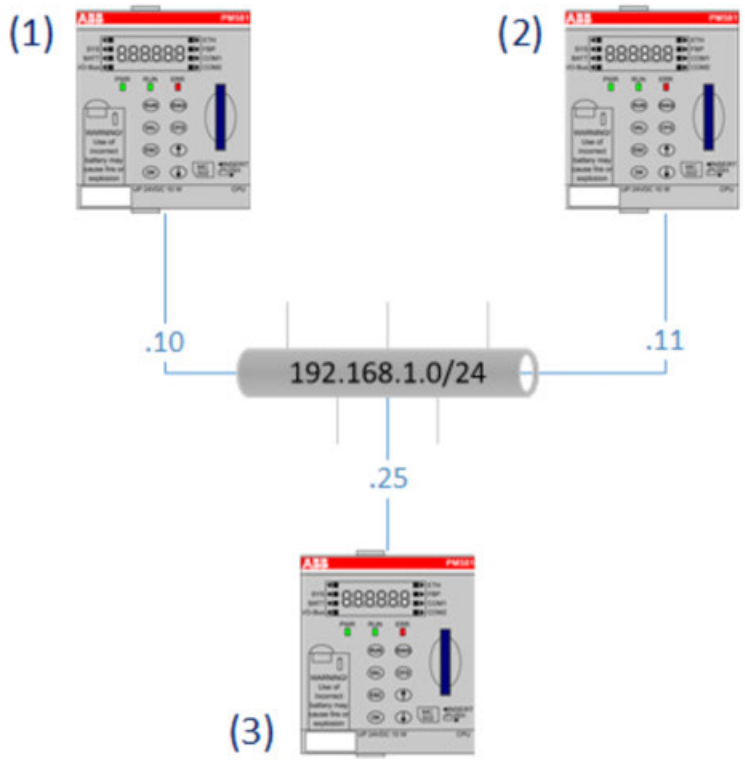
PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Multiple control stations on the same network

As of firmware version 2.4, an AC500 can be used as a substation for several control stations. For this, the control stations must be distinguished by their IP addresses. Should a control station have more than one IP address (redundancy), both possible IP addresses should also be entered for the allocated substation connection. As a result, even despite being equipped with several Ethernet interfaces, a device can only be one allocated control station at a time for a determined substation. Thus, several substations can be configured for different control stations on a AC500.



(1) Control station 1
(2) Control station 2
(3) 2 Substations (IEC60870-5-104 2nd Connection)

Configuration at control stations

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

The substation’s IP address must be specified at the control stations. For this, in the network settings of the control station (1 and 2) enter the IP addresses of the substation (in the example: 192.168.1.25). Option *“Enable redundant connection”* must be disabled.

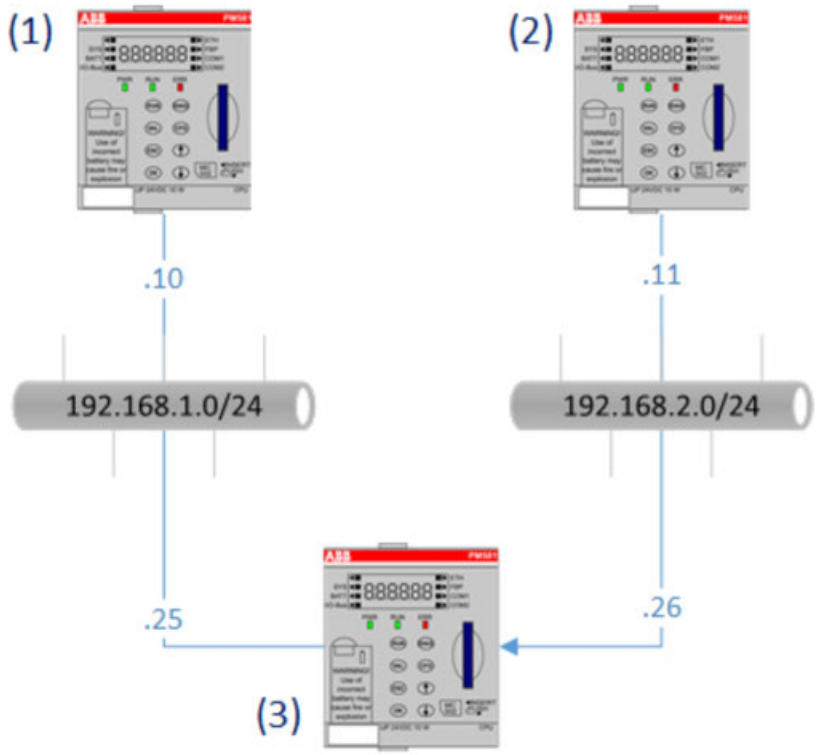
Configuration at substations

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

Both control station's IP addresses must be specified at the substation (3). For this, in the network settings of the substation enter the IP addresses of the control stations (in the example: 192.168.1.10 and 192.168.1.11). Option *“Enable redundant connection”* must be disabled.

Multiple control stations on different networks

As of firmware version 2.4, an AC500 can have several local Ethernet interfaces which can be used for separate control station connections. For this, a control station must be identified via its IP address. The substation address used locally is not used to distinguish a connection in order to enable a network and therefore route redundancy. On AC500, the acception of IEC60870-5-104 connections on an interface can only be prevented.



(1) Control station 1
(2) Control station 2
(3) 2 Substations with 2 Ethernet interfaces (2nd port and 2nd connection)

Configuration at control stations

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1 and 2). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.2.26). Option *"Enable redundant connection"* must be disabled.

Configuration at substations

PM591-ETH, PM595-ETH, PM5650-2ETH:

Both control station's IP addresses must be specified at the substation (3) under both substation connections. For this, in the network settings of the substation enter the IP addresses of the control stations (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be disabled.

Double connection

*This configuration **does** work.*

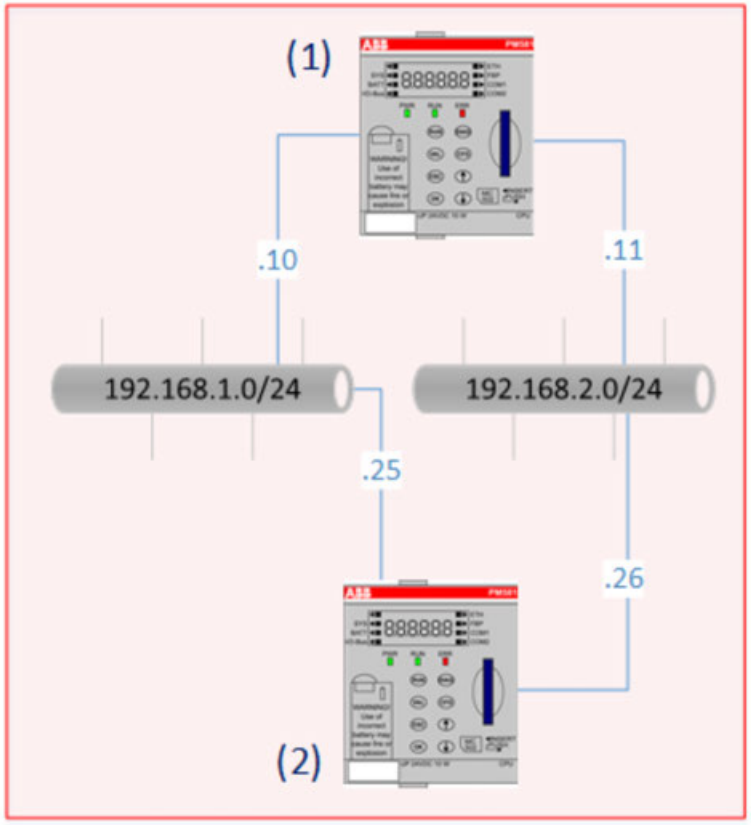
*But it is **senseless!***

It is possible to configure a double connection between 2 stations using 2 separate networks (at least logically separated sub-networks).

However, such a setup has no advantages via-à-vis the minimal structure right at the start ↳ *Chapter 1.6.6.3.1.2.2.6.1.1 “Minimal structure” on page 3850*.

For this setup, connection data must be double configured and double resources are also required at the stations, not providing any advantages whatsoever.

Rather the opposite is true, because such configurations are highly prone to errors.



(1) 2 Control stations with 2 Ethernet interfaces
(2) 2 Substations with 2 Ethernet interfaces

Faulty configuration

*This configuration **does not** work!*

If an AC500 is configured as a control station, the interface which is used to reach the substation is not defined.

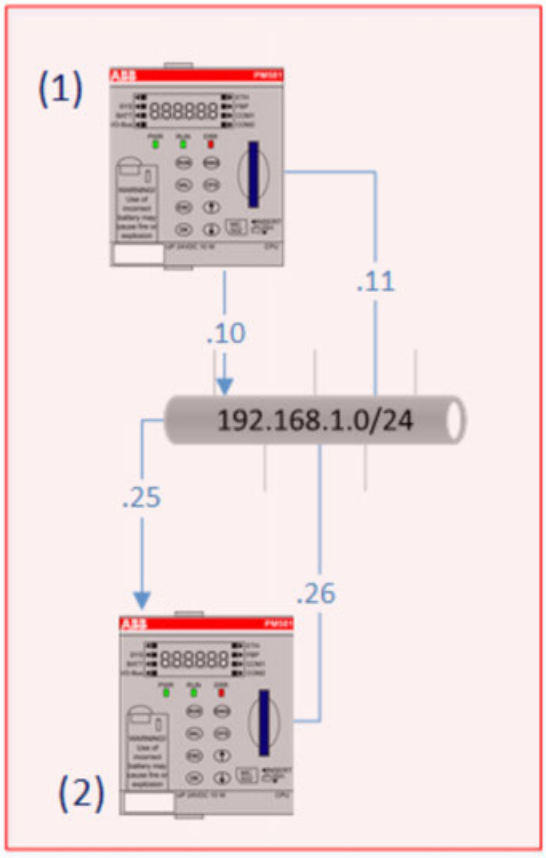
The decision as to which interface is used for this is taken by TCP/IP when running.

It is also dependent on the current network configuration.

Here, the current link status and the order of link recognition may be decisive for the interface to be used.

Such a scenario would not result in stable communication as both substations cannot clearly distinguish the control stations.

Instead, the connection management for a substation will assume that the control station has lost the connection and then establishes a connection.



(1) 2 Control stations with 2 Ethernet interfaces
(2) 2 Substations with 2 Ethernet interfaces

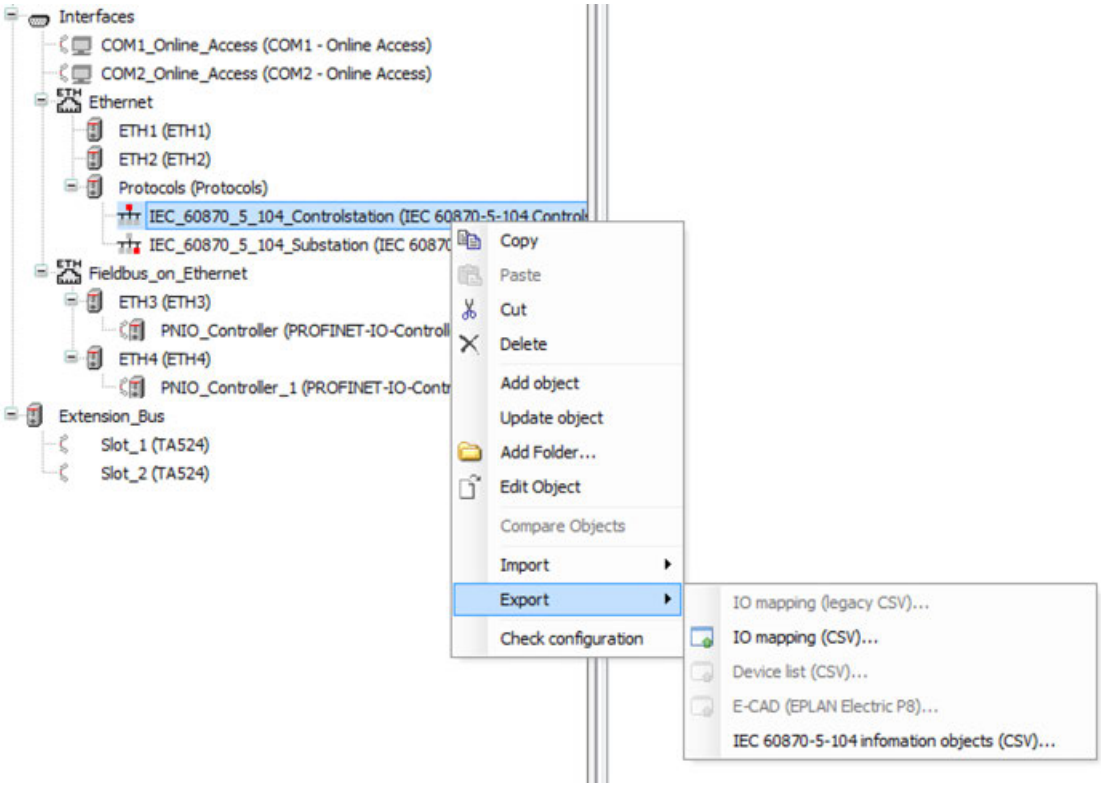
Export a CSV file

As an alternative many values can be modified at a time by exporting the configuration to a CSV file. After modifying the file data, import the CSV file *Chapter 1.6.6.3.1.2.3 “Import/Export functionality ” on page 3861*.

Import/Export functionality

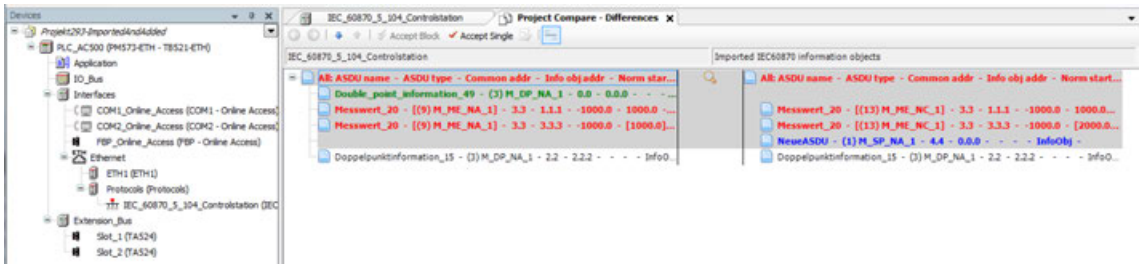
As of Automation Builder 1.1 (CBP >= 2.4) configuration of control stations and substations can be exported/imported via CSV file. Open the CSV file with a spreadsheet software (e.g. Microsoft Excel) and modify the values within the file to your convenience:

1. Export configuration data: right-click the node of the control station or substation to be exported.



2. Click *“Export ➔ IEC 60870-5-104 information objects (CSV)”* and store the CSV file to a desired directory.
3. Open the CSV file with a spreadsheet software (e.g. Microsoft Excel) and change the values to your convenience. Added table columns are only accepted after the last column.
4. Import configuration data: right-click the node of the control station or substation that has been exported previously.
5. Click *“Import ➔ IEC 60870-5-104 information objects (CSV)”* and select the CSV file from the file system. Configuration data is imported.

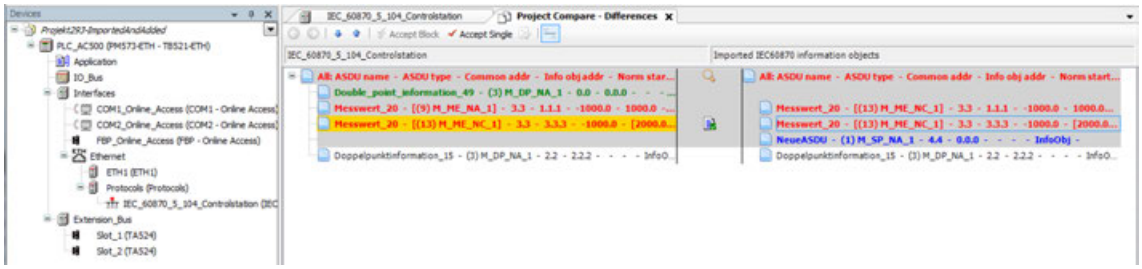
As of Automation Builder 1.1.1 during file import the project data is compared with the project data that is already available. In order to prevent data from being overwritten inadvertently, you can select the data that shall be imported in the *“Project Compare - Differences”* window:



Data on the left side of the window refers to already available project data. This data is displayed under *“Control station ➔ Information objects”* tab. Data on the right side of the window refers to new data that can be imported after your confirmation. Decide whether to import (and overwrite) the data or not.

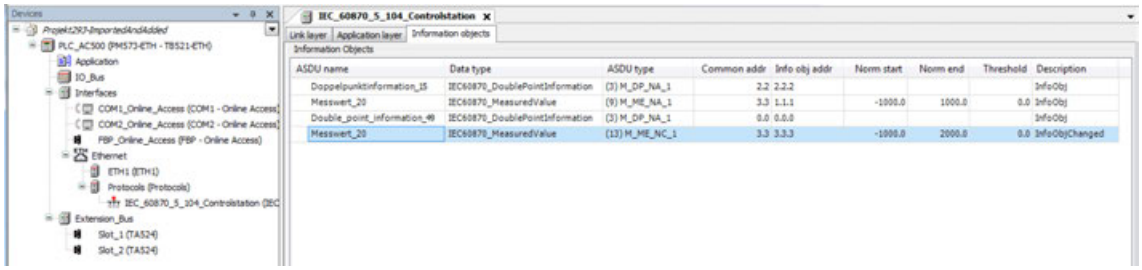
- Data in black color means the existing data and the data to be imported is identical.
- Data in red color means the existing data and the data to be imported differ. Decide whether to import the new data (and to overwrite the existing data) or not.
- Data in blue color means, the data to be imported is new and will be added to the existing data.
- Data that has been confirmed for the import already is displayed in green color (after clicking the *[Accept Single]* button).

In order to move data from one side of the window to another, select the data and click the *[Accept Single]* button. Data is highlighted in yellow.



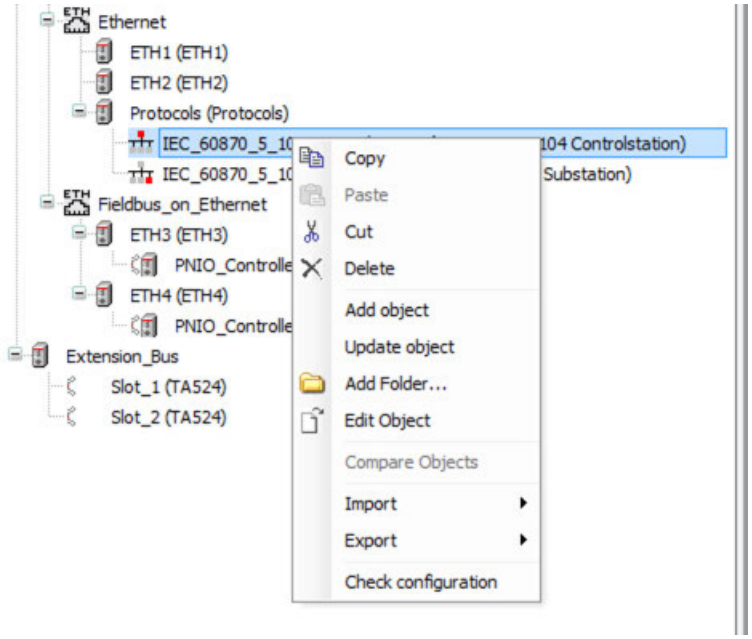
To confirm the import of all new data, click the top entry (here: All: ASDU name - ASDU tpye - Common addr - ...). Then, click the *[Accept Single]* button.

Close the *"Project Compare - Differences"* tab, save your project and confirm the message. The changes are displayed in the *"Information objects"* tab.



Validity check of configuration

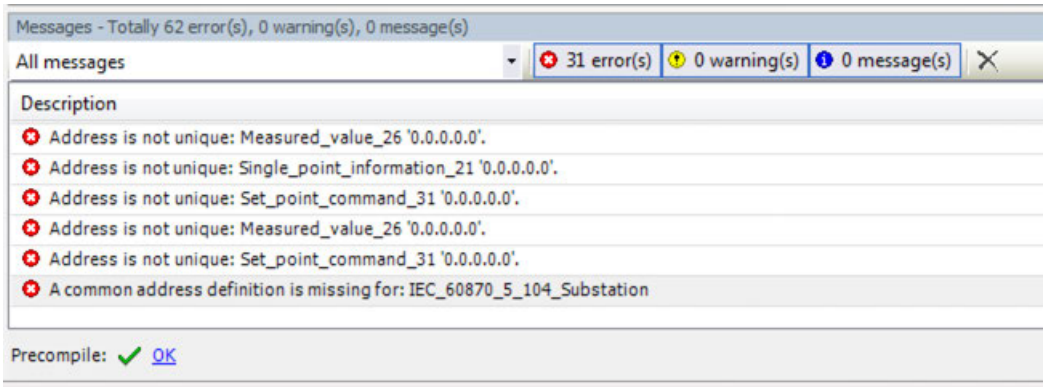
We recommend you to verify the IEC configuration of control stations and substations: Right-click a control station or substation -> Check configuration.



The check will look for the following topics:

- Duplicate addresses.
- Stations without any Information objects.
- ASDU names, which are not unique.

When a check finds errors or incompatibilities this will be reported in a separate messages view at the buttom of the window:



With a double-click on the error line, the part of the configuration with the violation will be opened. Now, you can correct the error.

IEC60870 compatibility list

AC500 V2.4 IEC60870-5-104 Compatibility List

9 Interoperability

This companion standard presents sets of parameters and alternatives from which subsets must be selected to implement particular telecontrol systems. Certain parameter values, such as the choice of "structured" or "unstructured" fields of the INFORMATION OBJECT ADDRESS of ASDUs represent mutually exclusive alternatives. This means that only one value of the defined parameters is admitted per system. Other parameters, such as the listed set of different process information in command and in monitor direction allow the specification of the complete set or subsets, as appropriate for given applications. This clause summarizes the parameters of the previous clauses to facilitate a suitable selection for a specific application. If a system is composed of equipment stemming from different manufacturers, it is necessary that all partners agree on the selected parameters.

The interoperability list is defined as in IEC 60870-5-101 and extended with parameters used in this standard. The text descriptions of parameters which are not applicable to this companion standard are strike-through (corresponding check box is marked black).

NOTE In addition, the full specification of a system may require individual selection of certain parameters for certain parts of the system, such as the individual selection of scaling factors for individually addressable measured values.

The selected parameters should be marked in the white boxes as follows:

☐ Function or ASDU is not used

☒ Function or ASDU is used as standardized (default)

☒ Function or ASDU is used in reverse mode

☒ Function or ASDU is used in standard and reverse mode

The possible selection (blank, X, R, or B) is specified for each specific clause or parameter.

A black check box indicates that the option cannot be selected in this companion standard.

9.1 System or device

(system-specific parameter, indicate definition of a system or a device by marking one of the following with "**X**")

☐ System definition

☐ Controlling station definition (Master)

☐ Controlled station definition (Slave)

9.2 Network configuration

(network-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~Point-to-point~~ ■ ~~Multipoint~~

■ ~~Multiple-point-to-point~~ ■ ~~Multipoint-star~~

AC500 V2.4 IEC60870-5-104 Compatibility List

9.3 Physical layer

(network-specific parameter, all interfaces and data rates that are used are to be marked "**X**")

Transmission speed (control direction)

Unbalanced interchange Circuit V.24/V.28 Standard	Unbalanced interchange Circuit V.24/V.28 Recommended if >1 200 bit/s	Balanced interchange Circuit X.24/X.27	
~~100 bit/s~~	~~2 400 bit/s~~	~~2 400 bit/s~~	~~56 000 bit/s~~
~~200 bit/s~~	~~4 800 bit/s~~	~~4 800 bit/s~~	~~64 000 bit/s~~
~~300 bit/s~~	~~9 600 bit/s~~	~~9 600 bit/s~~	
~~600 bit/s~~		~~19 200 bit/s~~	
~~1 200 bit/s~~		~~38 400 bit/s~~	

Transmission speed (monitor direction)

Unbalanced interchange Circuit V.24/V.28 Standard	Unbalanced interchange Circuit V.24/V.28 Recommended if >1 200 bit/s	Balanced interchange Circuit X.24/X.27	
~~100 bit/s~~	~~2 400 bit/s~~	~~2 400 bit/s~~	~~56 000 bit/s~~
~~200 bit/s~~	~~4 800 bit/s~~	~~4 800 bit/s~~	~~64 000 bit/s~~
~~300 bit/s~~	~~9 600 bit/s~~	~~9 600 bit/s~~	
~~600 bit/s~~		~~19 200 bit/s~~	
~~1 200 bit/s~~		~~38 400 bit/s~~	

9.4 Link layer

(network-specific parameter, all options that are used are to be marked "X". Specify the maximum frame length. If a non-standard assignment of class 2 messages is implemented for unbalanced transmission, indicate the Type ID and COT of all messages assigned to class 2.)

~~Frame format FT 1.2, single character 1 and the fixed time out interval are used exclusively in this companion standard.~~

Link transmission

- ~~Balanced transmission~~
- ~~Unbalanced transmission~~

Frame length

- ~~Maximum length L (number of octets)~~

Address field of the link

- ~~not present (balanced transmission only)~~
- ~~One octet~~
- ~~Two octets~~
- ~~Structured~~
- ~~Unstructured~~

AC500 V2.4 IEC60870-5-104 Compatibility List

When using an unbalanced link layer, the following ASDU types are returned in class 2 messages (low priority) with the indicated causes of transmission:

■ ~~The standard assignment of ASDUs to class 2 messages is used as follows:~~

Type identification	Cause of transmission
9, 11, 13, 21	<1>

■ ~~A special assignment of ASDUs to class 2 messages is used as follows:~~

Type identification	Cause of transmission

~~Note: (In response to a class 2 poll, a controlled station may respond with class 1 data when there is no class 2 data available).~~

9.5 Application layer

Transmission mode for application data

Mode 1 (Least significant octet first), as defined in 4.10 of IEC 60870-5-4, is used exclusively in this companion standard.

Common address of ASDU
(system-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~One octet~~ [X] Two octets

Information object address
(system-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~One octet~~ [ ] Structured
■ ~~Two octets~~ [ ] Unstructured
[X] Three octets

Cause of transmission
(system-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~One octet~~ [X] Two octets (with originator address). Originator address is set to zero if not used

Length of APDU
(system-specific parameter, specify the maximum length of the APDU per system)

The maximum length of APDU for both directions is 253. It is a fixed system parameter.

■ Maximum length of APDU per system in control direction

AC500 V2.4 IEC60870-5-104 Compatibility List

■ Maximum length of APDU per system in monitor direction

Selection of standard ASDUs

Process information in monitor direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<1>	:= Single-point information	M_SP_NA_1
■	~~<2>~~	~~:= Single-point information with time tag~~	~~M_SP_TA_1~~
X	<3>	:= Double-point information	M_DP_NA_1
■	~~<4>~~	~~:= Double-point information with time tag~~	~~M_DP_TA_1~~
	<5>	:= Step position information	M_ST_NA_1
■	~~<6>~~	~~:= Step position information with time tag~~	~~M_ST_TA_1~~
	<7>	:= Bitstring of 32 bit	M_BO_NA_1
■	~~<8>~~	~~:= Bitstring of 32 bit with time tag~~	~~M_BO_TA_1~~
X	<9>	:= Measured value, normalized value	M_ME_NA_1
■	~~<10>~~	~~:= Measured value, normalized value with time tag~~	~~M_ME_TA_1~~
X	<11>	:= Measured value, scaled value	M_ME_NB_1
■	~~<12>~~	~~:= Measured value, scaled value with time tag~~	~~M_ME_TB_1~~
X	<13>	:= Measured value, short floating point value	M_ME_NC_1
■	~~<14>~~	~~:= Measured value, short floating point value with time tag~~	~~M_ME_TC_1~~
X	<15>	:= Integrated totals	M_IT_NA_1
■	~~<16>~~	~~:= Integrated totals with time tag~~	~~M_IT_TA_1~~
■	~~<17>~~	~~:= Event of protection equipment with time tag~~	~~M_EP_TA_1~~
■	~~<18>~~	~~:= Packed start events of protection equipment with time tag~~	~~M_EP_TB_1~~
■	~~<19>~~	~~:= Packed output circuit information of protection equipment with time tag~~	~~M_EP_TC_1~~
■	<20>	:= Packed single-point information with status change detection	M_SP_NA_1
■	<21>	:= Measured value, normalized value without quality descriptor	M_ME_ND_1

X	<30>	:= Single-point information with time tag CP56Time2a	M_SP_TB_1
X	<31>	:= Double-point information with time tag CP56Time2a	M_DP_TB_1
	<32>	:= Step position information with time tag CP56Time2a	M_ST_TB_1
	<33>	:= Bitstring of 32 bit with time tag CP56Time2a	M_BO_TB_1
X	<34>	:= Measured value, normalized value with time tag CP56Time2a	M_ME_TD_1
	<35>	:= Measured value, scaled value with time tag CP56Time2a	M_ME_TE_1
X	<36>	:= Measured value, short floating point value with time tag CP56Time2a	M_ME_TF_1
X	<37>	:= Integrated totals with time tag CP56Time2a	M_IT_TB_1
■	<38>	:= Event of protection equipment with time tag CP56Time2a	M_EP_TD_1
■	<39>	:= Packed start events of protection equipment with time tag CP56Time2a	M_EP_TE_1
■	<40>	:= Packed output circuit information of protection equipment with time tag CP56Time2a	M_EP_TF_1

In this companion standard only the use of the set <30> – <40> for ASDUs with time tag is permitted.

AC500 V2.4 IEC60870-5-104 Compatibility List

Process information in control direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<45> :=	Single command	C_SC_NA_1
X	<46> :=	Double command	C_DC_NA_1
	<47> :=	Regulating step command	C_RC_NA_1
X	<48> :=	Set point command, normalized value	C_SE_NA_1
	<49> :=	Set point command, scaled value	C_SE_NB_1
X	<50> :=	Set point command, short floating point value	C_SE_NC_1
	<51> :=	Bitstring of 32 bit	C_BO_NA_1
X	<58> :=	Single command with time tag CP56Time2a	C_SC_TA_1
X	<59> :=	Double command with time tag CP56Time2a	C_DC_TA_1
	<60> :=	Regulating step command with time tag CP56Time2a	C_RC_TA_1
X	<61> :=	Set point command, normalized value with time tag CP56Time2a	C_SE_TA_1
	<62> :=	Set point command, scaled value with time tag CP56Time2a	C_SE_TB_1
X	<63> :=	Set point command, short floating point value with time tag CP56Time2a	C_SE_TC_1
	<64> :=	Bitstring of 32 bit with time tag CP56Time2a	C_BO_TA_1

Either the ASDUs of the set <45> – <51> or of the set <58> – <64> are used.

System information in monitor direction
(station-specific parameter, mark with an "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<70> :=	End of initialization	M_EI_NA_1

System information in control direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<100>:=	Interrogation command	C_IC_NA_1
X	<101>:=	Counter interrogation command	C_CI_NA_1
X	<102>:=	Read command	C_RD_NA_1
X	<103>:=	Clock synchronization command (option see 7.6)	C_CS_NA_1
	<104>:=	~~Test command~~	~~C_TS_NA_1~~
X	<105>:=	Reset process command	C_RP_NA_1
	<106>:=	~~Delay acquisition command~~	~~C_CD_NA_1~~
X	<107>:=	Test command with time tag CP56Time2a	C_TS_TA_1

AC500 V2.4 IEC60870-5-104 Compatibility List

Parameter in control direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<110>:= Parameter of measured value, normalized value	P_ME_NA_1
	<111>:= Parameter of measured value, scaled value	P_ME_NB_1
X	<112>:= Parameter of measured value, short floating point value	P_ME_NC_1
	<113>:= Parameter activation	P_AC_NA_1

File transfer
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

	<120>:= File ready	F_FR_NA_1
	<121>:= Section ready	F_SR_NA_1
	<122>:= Call directory, select file, call file, call section	F_SC_NA_1
	<123>:= Last section, last segment	F_LS_NA_1
	<124>:= Ack file, ack section	F_AF_NA_1
	<125>:= Segment	F_SG_NA_1
	<126>:= Directory {blank or X, only available in monitor (standard) direction}	F_DR_TA_1
	<127>:= Query Log – Request archive file	F_SC_NB_1

Type identifier and cause of transmission assignments
(station-specific parameters)

Shaded boxes: option not required.

Black boxes: option not permitted in this companion standard

Blank: functions or ASDU not used.

Mark Type Identification/Cause of transmission combinations:

"**X**" if only used in the standard direction;

"**R**" if only used in the reverse direction;

"**B**" if used in both directions.

Type identification		**Cause of transmission**																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	20 to 36	37 to 41	44	45	46	47
<1>	M_SP_NA_1		X	X		X									X					
~~<2>~~	~~M_SP_TA_1~~																			
<3>	M_DP_NA_1		X	X		X									X					
~~<4>~~	~~M_DP_TA_1~~																			
<5>	M_ST_NA_1																			
~~<6>~~	~~M_ST_TA_1~~																			
<7>	M_BO_NA_1																			
~~<8>~~	~~M_BO_TA_1~~																			
<9>	M_ME_NA_1	X	X	X		X									X					
~~<10>~~	~~M_ME_TA_1~~																			
<11>	M_ME_NB_1																			
~~<12>~~	~~M_ME_TB_1~~																			

AC500 V2.4 IEC60870-5-104 Compatibility List

Type identification		Cause of transmission																		
		1	2	3	4	5	6	7	8	9	10	11	12	13	20 to 36	37 to 41	44	45	46	47
<13>	M_ME_NC_1	x	x	x		x									x					
~~<14>~~	~~M_ME_TC_1~~			■		■														
<15>	M_IT_NA_1			x												x				
~~<16>~~	~~M_IT_TA_1~~			■												■				
~~<17>~~	~~M_EP_TA_1~~			■																
~~<18>~~	~~M_EP_TB_1~~			■																
~~<19>~~	~~M_EP_TC_1~~			■																
<20>	M_PS_NA_1																			
<21>	M_ME_ND_1																			
<30>	M_SP_TB_1			x		x														
<31>	M_DP_TB_1			x		x														
<32>	M_ST_TB_1																			
<33>	M_BO_TB_1																			
<34>	M_ME_TD_1			x		x														
<35>	M_ME_TE_1																			
<36>	M_ME_TF_1			x		x														
<37>	M_IT_TB_1			x												x				
<38>	M_EP_TD_1																			
<39>	M_EP_TE_1																			
<40>	M_EP_TF_1																			
<45>	C_SC_NA_1						x	x			x									
<46>	C_DC_NA_1						x	x			x									
<47>	C_RC_NA_1																			
<48>	C_SE_NA_1						x	x			x									
<49>	C_SE_NB_1																			
<50>	C_SE_NC_1						x	x			x									
<51>	C_BO_NA_1																			
<58>	C_SC_TA_1						x	x			x									
<59>	C_DC_TA_1						x	x			x									
<60>	C_RC_TA_1																			
<61>	C_SE_TA_1						x	x			x									
<62>	C_SE_TB_1																			
<63>	C_SE_TC_1						x	x			x									
<64>	C_BO_TA_1																			
<70>	M_EI_NA_1*				x															
<100>	C_IC_NA_1						x	x			x									
<101>	C_CI_NA_1						x	x			x									
<102>	C_RD_NA_1					x														
<103>	C_CS_NA_1			■			x	x												
~~<104>~~	~~C_TS_NA_1~~						■	■									■	■	■	■
<105>	C_RP_NA_1						x													
~~<106>~~	~~C_CD_NA_1~~			■			■	■									■	■	■	■
<107>	C_TS_TA_1																			
<110>	P_ME_NA_1						x	x							x					
<111>	P_ME_NB_1																			
<112>	P_ME_NC_1						x	x							x					
<113>	P_AC_NA_1																			
<120>	F_FR_NA_1																			
<121>	F_SR_NA_1																			
<122>	F_SC_NA_1																			
<123>	F_LS_NA_1																			
<124>	F_AF_NA_1																			
<125>	F_SG_NA_1																			
<126>	F_DR_TA_1*																			

AC500 V2.4 IEC60870-5-104 Compatibility List

Type identification		**Cause of transmission**																
		1	2	3	4	5	6	7	8	9	10	11	12	13	20 to 36	37 to 41	44	45	46	47
<127>	F_SC_NB_1*																			

***** Blank or X only

AC500 V2.4 IEC60870-5-104 Compatibility List

9.6 Basic application functions

Station initialization
(station-specific parameter, mark "**X**" if function is used)

[X] Remote initialization

Cyclic data transmission
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions)

[X] Cyclic data transmission

Read procedure
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions)

[X] Read procedure

Spontaneous transmission
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions)

[X] Spontaneous transmission

Double transmission of information objects with cause of transmission spontaneous
(station-specific parameter, mark each information type "**X**" where both a Type ID without time and corresponding Type ID with time are issued in response to a single spontaneous change of a monitored object)

The following type identifications may be transmitted in succession caused by a single status change of an information object. The particular information object addresses for which double transmission is enabled are defined in a project-specific list.

- [ ] Single-point information M_SP_NA_1, M_SP_TA_1, M_SP_TB_1 and M_PS_NA_1
- [ ] Double-point information M_DP_NA_1, M_DP_TA_1 and M_DP_TB_1
- [ ] Step position information M_ST_NA_1, M_ST_TA_1 and M_ST_TB_1
- [ ] Bitstring of 32 bit M_BO_NA_1, M_BO_TA_1 and M_BO_TB_1 (if defined for a specific project)
- [ ] Measured value, normalized value M_ME_NA_1, M_ME_TA_1, M_ME_ND_1 and M_ME_TD_1
- [ ] Measured value, scaled value M_ME_NB_1, M_ME_TB_1 and M_ME_TE_1
- [ ] Measured value, short floating point number M_ME_NC_1, M_ME_TC_1 and M_ME_TF_1

AC500 V2.4 IEC60870-5-104 Compatibility List

Station interrogation
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] global
- [ ] group 1
- [ ] group 2
- [ ] group 3
- [ ] group 4
- [ ] group 5
- [ ] group 6
- [ ] group 7
- [ ] group 8
- [ ] group 9
- [ ] group 10
- [ ] group 11
- [ ] group 12
- [ ] group 13
- [ ] group 14
- [ ] group 15
- [ ] group 16

Information object addresses assigned to each group must be shown in a separate table.

Clock synchronization
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Clock synchronization
- [ ] Day of week used
- [ ] RES1, GEN (time tag substituted/ not substituted) used
- [ ] SU-bit (summertime) used

optional, see 7.6

Command transmission
(object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Direct command transmission
- [X] Direct set point command transmission
- [ ] Select and execute command
- [ ] Select and execute set point command
- [ ] C_SE ACTTERM used

- [ ] No additional definition
- [X] Short-pulse duration (duration determined by a system parameter in the outstation)
- [X] Long-pulse duration (duration determined by a system parameter in the outstation)
- [X] Persistent output

- [ ] Supervision of maximum delay in command direction of commands and set point commands
- [ ] Maximum allowable delay of commands and set point commands

AC500 V2.4 IEC60870-5-104 Compatibility List

Transmission of integrated totals
(station- or object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [ ] Mode A: Local freeze with spontaneous transmission
- [ ] Mode B: Local freeze with counter interrogation
- [ ] Mode C: Freeze and transmit by counter-interrogation commands
- [ ] Mode D: Freeze by counter-interrogation command, frozen values reported

- [X] Counter read
- [X] Counter freeze without reset
- [X] Counter freeze with reset
- [X] Counter reset

- [X] General request
- [X] Request counter group 1
- [X] Request counter group
- [X] Request counter group 3
- [X] Request counter group 4

Parameter loading
(object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Threshold value
- [ ] Smoothing factor
- [ ] Low limit for transmission of measured values
- [ ] High limit for transmission of measured values

Parameter activation
(object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [ ] Act/deact of persistent cyclic or periodic transmission of the addressed object

Test procedure
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Test procedure

AC500 V2.4 IEC60870-5-104 Compatibility List

File transfer
(station-specific parameter, mark "**X**" if function is used).

File transfer in monitor direction

☐ Transparent file
☐ Transmission of disturbance data of protection equipment
☐ Transmission of sequences of events
☐ Transmission of sequences of recorded analogue values

File transfer in control direction

☐ Transparent file

Background scan
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

☒ Background scan

Acquisition of transmission delay
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

■ ~~Acquisition of transmission delay~~

Definition of time outs

Parameter	Default value	Remarks	Selected value
t_0	30 s	Time-out of connection establishment	
t_1	15 s	Time-out of send or test APDUs	
t_2	10 s	Time-out for acknowledges in case of no data messages $t_2 < t_1$	
t_3	20 s	Time-out for sending test frames in case of a long idle state	

Maximum range for timeouts t_0 to t_2: 1 s to 255 s, accuracy 1 s.
Recommended range for timeout t_3: 1 s to 48 h, resolution 1 s.
Long timeouts for t_3 may be needed in special cases where satellite links or dialup connections are used (for instance to establish connection and collect values only once per day or week).

Maximum number of outstanding I format APDUs *k* and latest acknowledge APDUs (*w*)

Parameter	Default value	Remarks	Selected value
k	12 APDUs	Maximum difference receive sequence number to send state variable	
w	8 APDUs	Latest acknowledge after receiving *w* I format APDUs	

Maximum range of values *k*: 1 to 32767 (2^{15}–1) APDUs, accuracy 1 APDU

AC500 V2.4 IEC60870-5-104 Compatibility List

Maximum range of values *w*: 1 to 32767 APDUs, accuracy 1 APDU (Recommendation: *w* should not exceed two-thirds of *k*).

Portnumber

Parameter	Value	Remarks
Portnumber	2404	In all cases

Redundant connections

2 Number N of redundancy group connections used

RFC 2200 suite

RFC 2200 is an official Internet Standard which describes the state of standardization of protocols used in the Internet as determined by the Internet Architecture Board (IAB). It offers a broad spectrum of actual standards used in the Internet. The suitable selection of documents from RFC 2200 defined in this standard for given projects has to be chosen by the user of this standard.

☐ Ethernet 802.3

☐ Serial X.21 interface

☐ Other selection from RFC 2200:

List of valid documents from RFC 2200

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. etc.

1.6.6.3.2 IEC 61850 Server

IEC 61850 Server

The package CODESYS IEC 61850 Server is a configurator for creating an IEC 61850 Server according to IEC 61850. The IEC 61850 is the communication standard for switchgear automation of medium and high voltage technology.

The essential features of this package are::

- Configuration of data models for IEDs (Intelligent Electronic Device) with logical devices, logical nodes, data objects and data attributes
- Generation of the corresponding IEC 61131-3 code
- Mapping of data attributes to IEC 61131-3 variables
- Configuration of dynamic data sets
- Buffered and unbuffered Reports
- Protocols implemented: MMS and GOOSE
- Import /Export of different SCL formats

Inserting the IEC 61850 Server into the device tree

The IEC 61850 Server is inserted below an Ethernet Adapter in the device tree. For this select the Ethernet Adapter in the device tree and activate the context menu command *“Add device...”*. In the opened dialog select the IEC 61850 Server in the *“Miscellaneous ”* category and activate the *“Add device ”* button .

The configuration of the IEC 61850 Server takes place in the *Chapter 1.6.6.3.2.3.1 “IEC 61850 Editor” on page 3885*.

Quickstart

Here, a project with an IEC 61850 Server is created as an example. After the configuration of the Server, a data set is created and assigned to a Report. Subsequently the code is generated for the IEC 61850 Server and the project is loaded to the PLC. On the PLC the project can be connected with an d IEC 61850 client.

Step 1: Create a new project and insert the IEC 61850 Server

First create a new project. Select the *“Standard project ”* template.

Subsequently the dialog opens for selecting the PLC and the implementation language. Select the CODESYS Control Win V3 PLC and the *“Structured Text (ST)”* implementation language.

Now the project is created displayed with its objects in the device tree.

In order to add the IEC 61850 Server to the PLC, first add an Ethernet Adapter:

1. Mark the PLC in the device tree and activate the context menu command *“Add Device...”*
2. In the *“Add Device”* dialog select the adapter *“Ethernet ”* of the *“Fieldbusses ➔ Ethernet Adapter”* category and confirm your selection by activating the *“Add Device ”* button.

Subsequently, add the IEC 61850 Server to the Ethernet Adapter as follows:

1. Select the Ethernet Adapter in the device tree and activate the context menu command *“Add Device...”*.
2. In the *“Add Device”* dialog select the *“IEC 61850 Server”* of the *“Miscellaneous ”* category and confirm your selection with the *“Add Device ”* button.

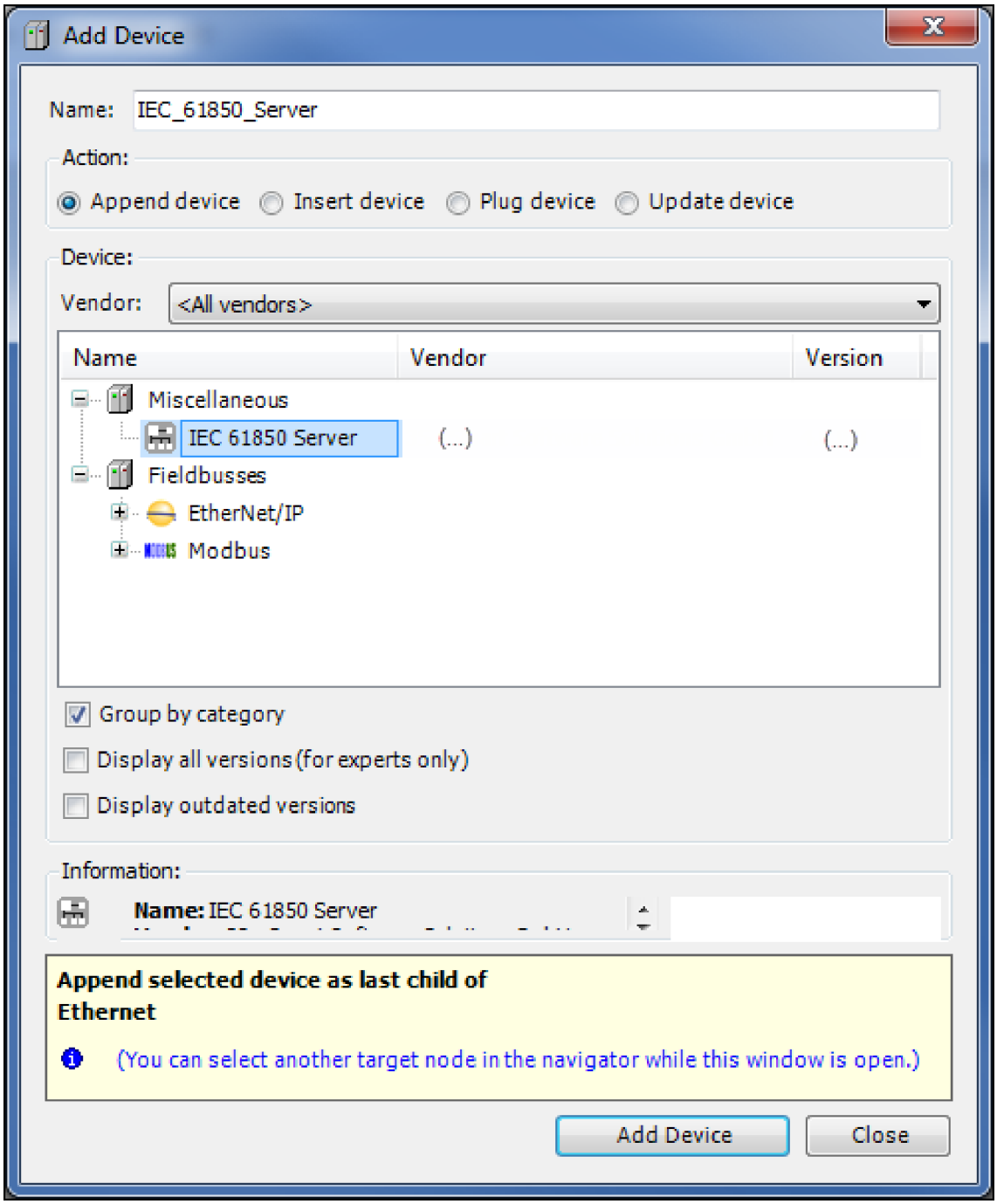


Fig. 311: 'Add Device' dialog

Now the IEC 61850 Server is inserted in the device tree.

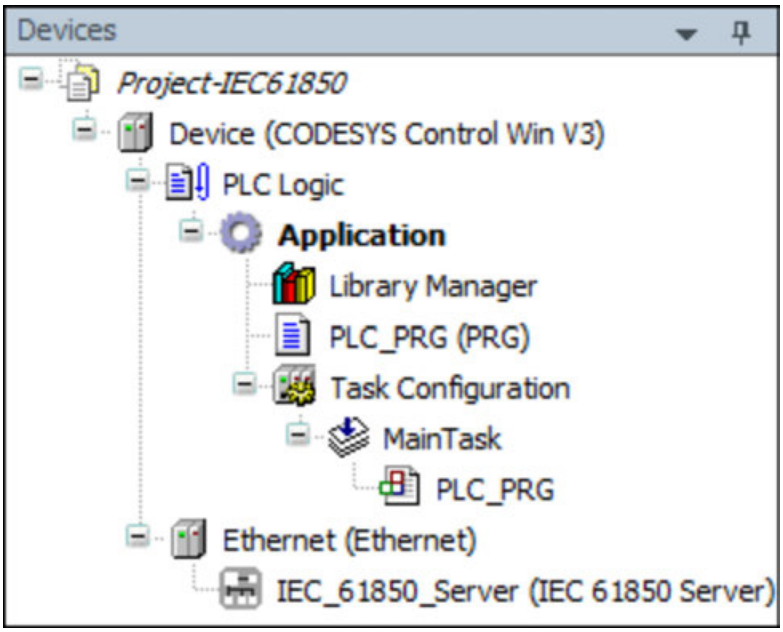


Fig. 312: Device tree with IEC 61850 Server

Step 2: Add the Logical Device to the server

Open the editor for the configuration of the server via a double-click on the IEC 61850 Server in the device tree.

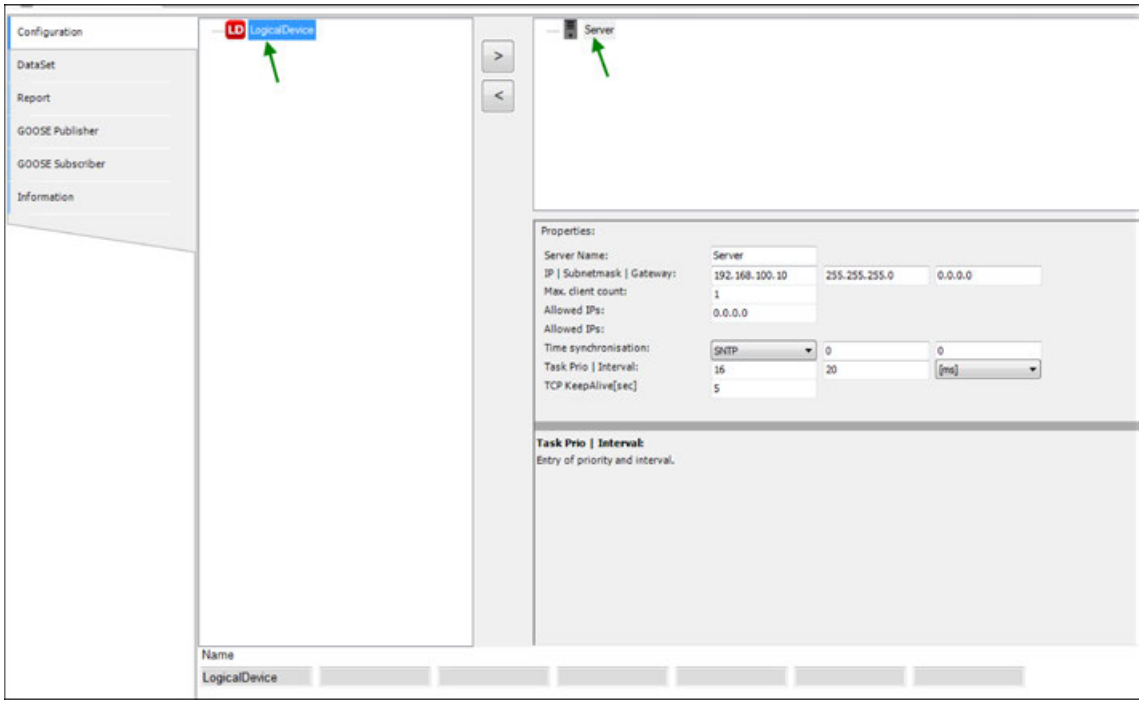


Fig. 313: editor of the IEC 61850 Server

First a *"Logical Device"* is added to the server. The *"Logical Device "* is the instance of an IED.

1. Select the *"Logical Device"*
2. Activate the *">"* button

Together with the *"Logical Device"* the two LNC instances () *"LLN0"* and *"LPHD1"* are added. These two information objects are elements of every IED and can not be removed.

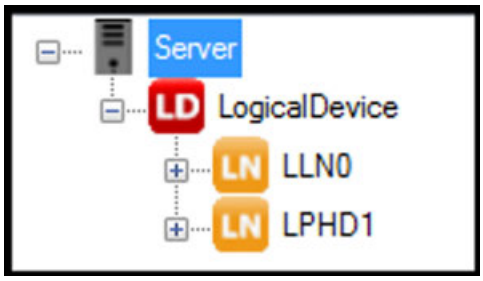


Fig. 314: Server with Logical Device, LLN0 and LPHD1

Step 3: Add another LNC instance to the Logical Device

1. Select the *"Logical Device"* below the server
2. On the left-hand side select the *"XCBR"* LNC instance below *"LN [Xxxx]-Switchgear"*
3. Activate the *">"* button

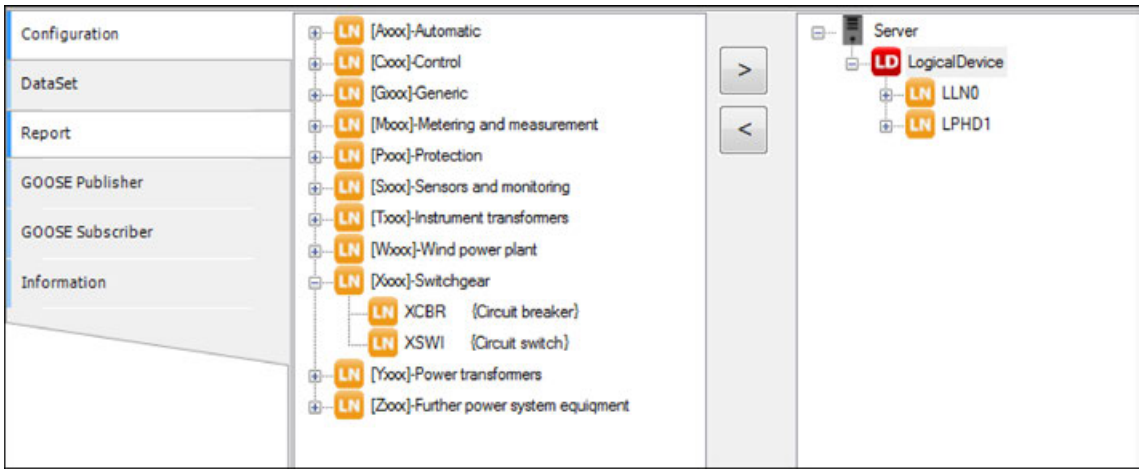


Fig. 315: Adding the LNC instance 'XCBR'

Step 4: Expand the *"XCBR"* LNC instance with the optional *"MaxOpCap"* CDC instance

If you select the LNC instance on the right-hand side, all of the optional and obligatory CDCs (data objects DC) will be displayed on the left-hand side. .

1. Select the *"XCBR"* LNC instance on the right-hand side
2. Select the *"MaxOpCap"* CDC instance on the left-hand side
3. Activate the *">"* button

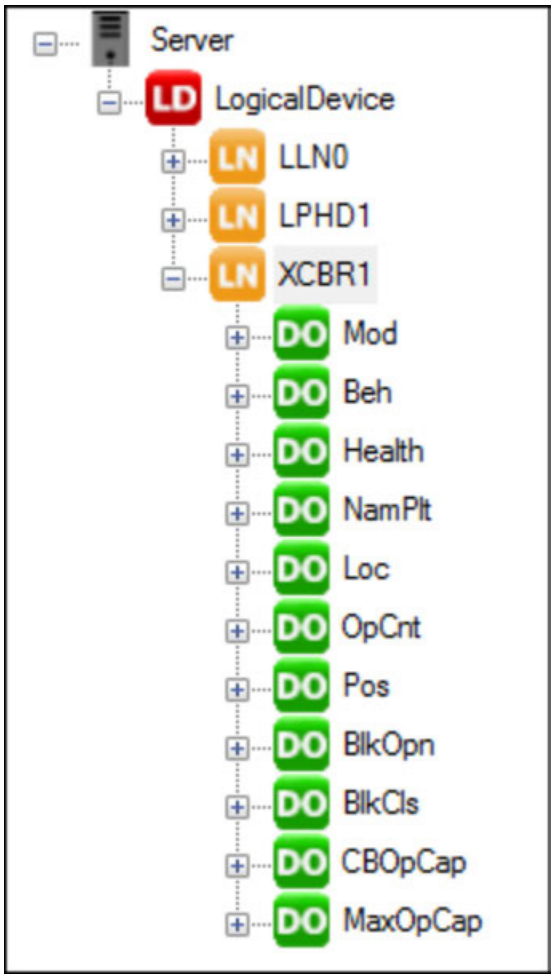


Fig. 316: XCBR1 with 'MaxOpCap' CDC

Step 5: Link an attribute (DA) of the IEC 61850 Server with a CODESYS variable

1. Select the desired attribute (in the example: *"Server ➔ LogicalDevice ➔ XCBR1 ➔ MaxOpCap ➔ DA (ST I INT32) StVal "*)
2. Edit the CODESYS variable name in the input field *"Monitoring Var"* (in the example Var_stVal) in the *"Properties"* section

The *"Autom. declare"* option must be activated, thus the variable is declared automatically as global variable by theIEC 61850 Server. You can edit the initial value in the input field next to the *"Monitoring Var"* field.

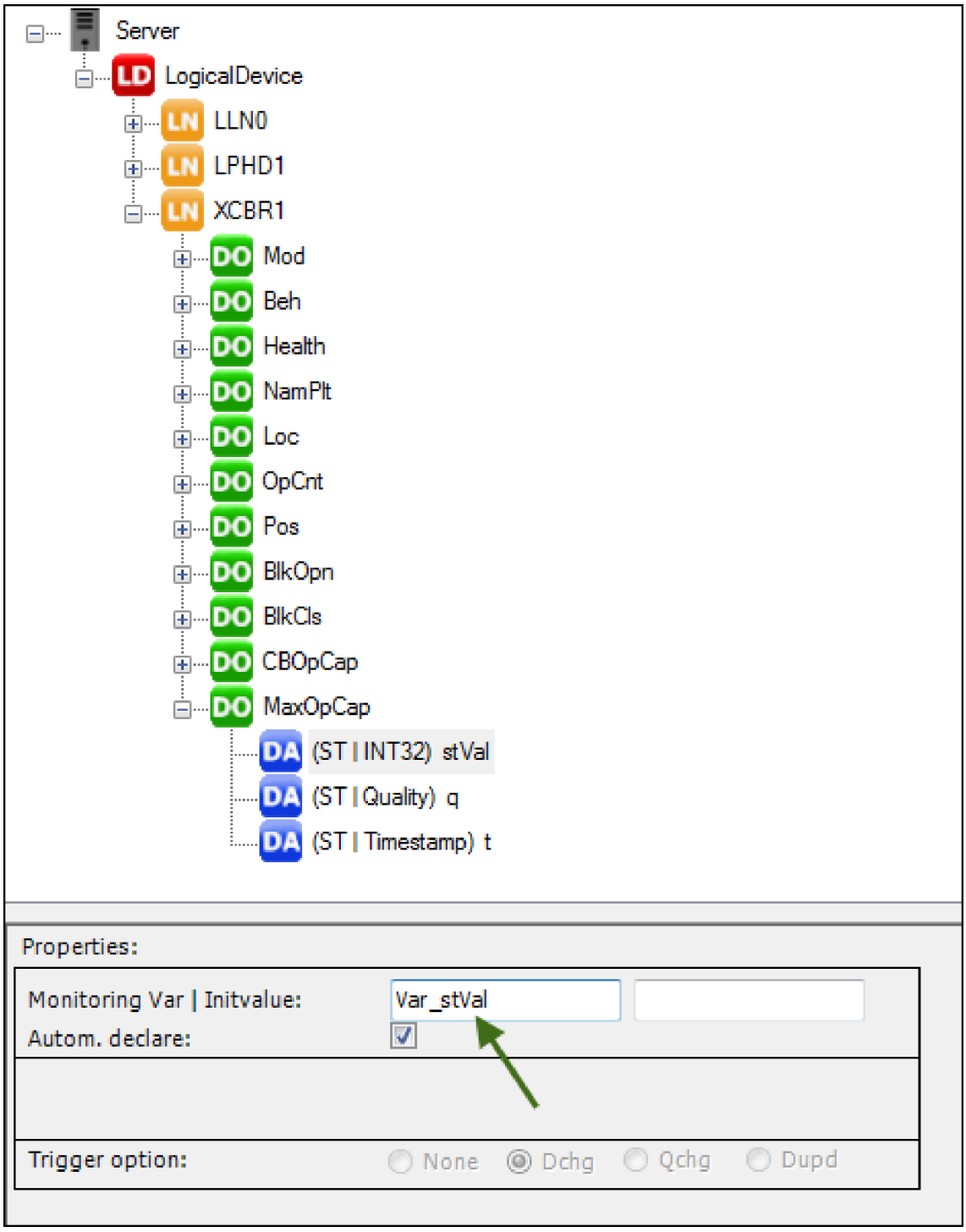


Fig. 317: Link of the attribute 'stVal' to the monitoring variable 'Var_stVal'

Step 6: Create a data set

In this step you create a data set (Compilation of data) for the IEC 61850 configuration created in the previous steps.

1. Open the *"DataSet"* tab
2. Activate the *"New"* button. The created *"LLN0.dataSet_0"* data set is displayed in the *"DataSets"* section.
3. Select the *"LLN0.dataSet_0"* DataSet
4. Select the *"MaxOpCap"* data object on the left-hand side (*"Server ➔ LogicalDevice ➔ LN XCBR1 ➔ FC ST ➔ DO MaxOpCap"*)
5. Activate the *">"* button.

Now the data set contains the data object *“LogicalDevice/XCBR1.ST.MaxOpCap”*

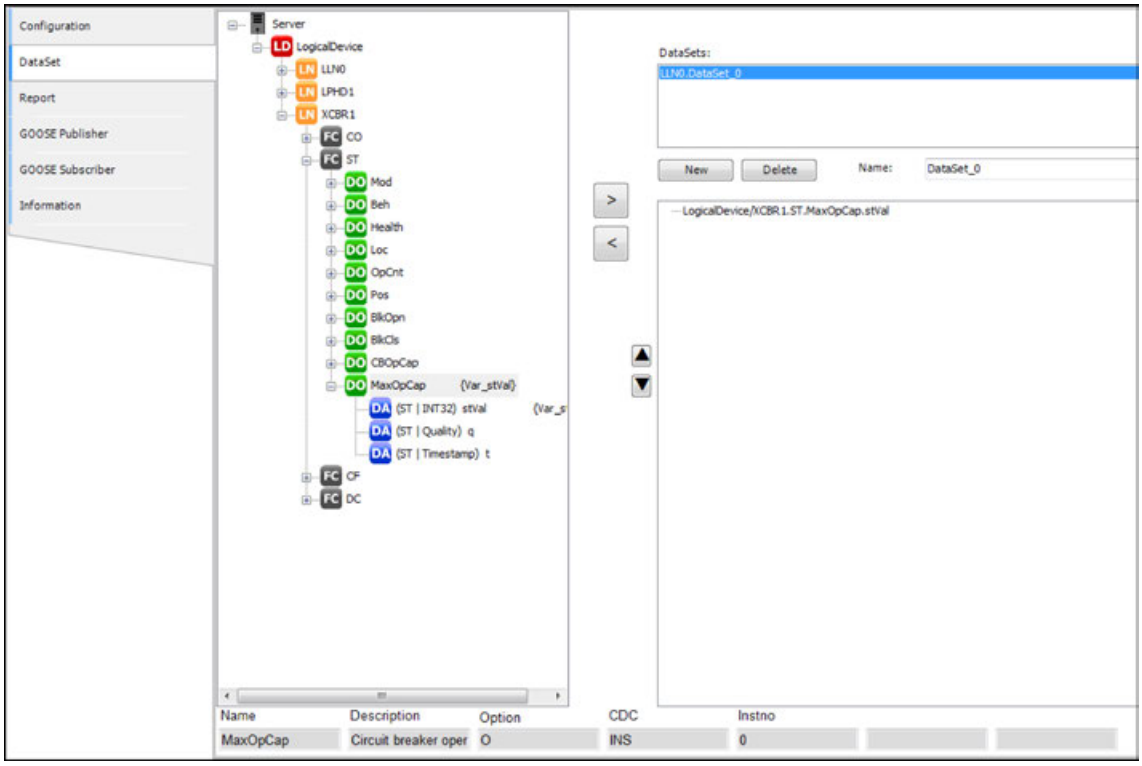


Fig. 318: 'DataSet' tab

Step 7: Create a Report

In this step you assign a report to the defined data set. A report transports the data assigned via a data set to a connected client in the event of a trigger (see *Trigger Options*).

1. Open the *“Report”* tab
2. Activate the *“New ”* button. The *“RCB_1”* is displayed in the *“Reports”* section. In the *“Name:”* field you can change the name of the report.
3. Select the *“LLN0.DataSet_0”* data set in the *“DataSet”* selection list

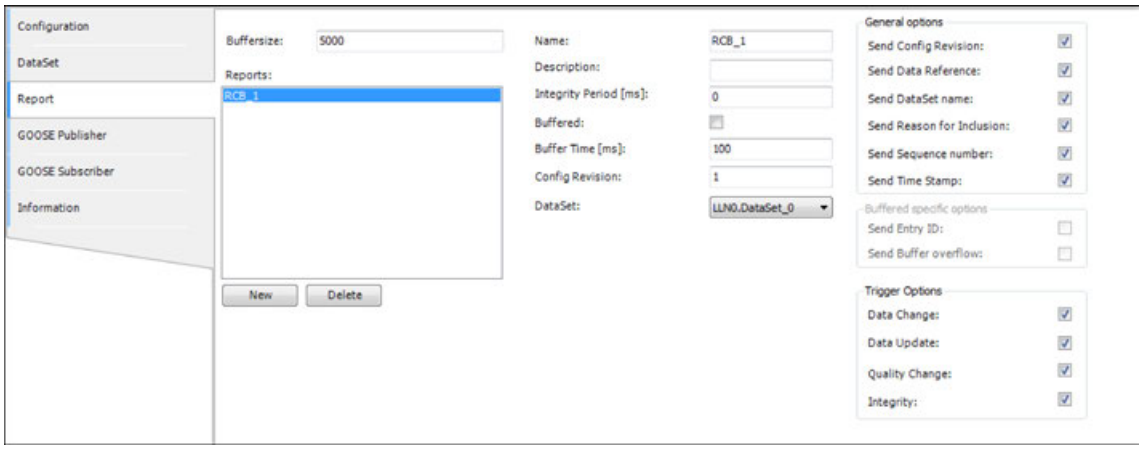


Fig. 319: 'Report' tab with created 'RCB' report

You set options about the reporting behavior in the *“General options”* section, you select the events that trigger a report in the *“Trigger Options”* section (for more information about these options see *Trigger Options*).

Step 8: Generate code and load the application to the PLC

The *"Generate code"* command of the menu *"IEC61850"* generates code from the created configuration and puts it into the *"IEC61850 Generated POUs"* folder of the device tree.

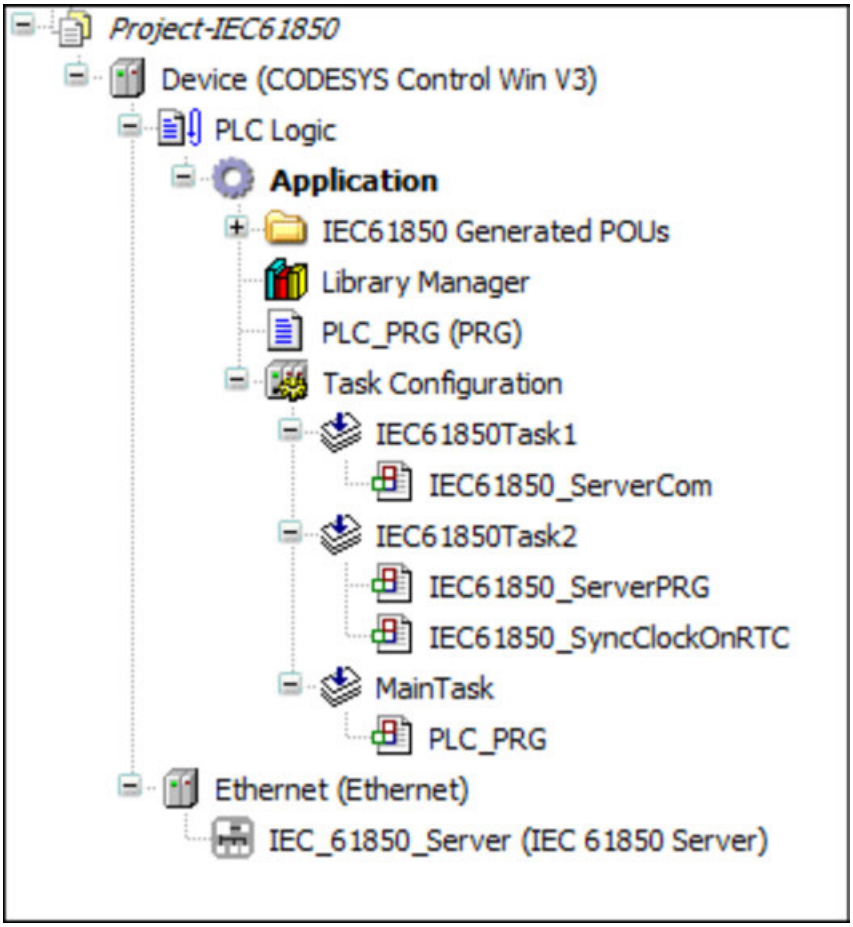


Fig. 320: device tree with 'IEC61850 Generated POUs'

The global variable *"Var_stVal"* created in step 5 is listed in the *"IEC61850_Generated_GVL"* global variables list.

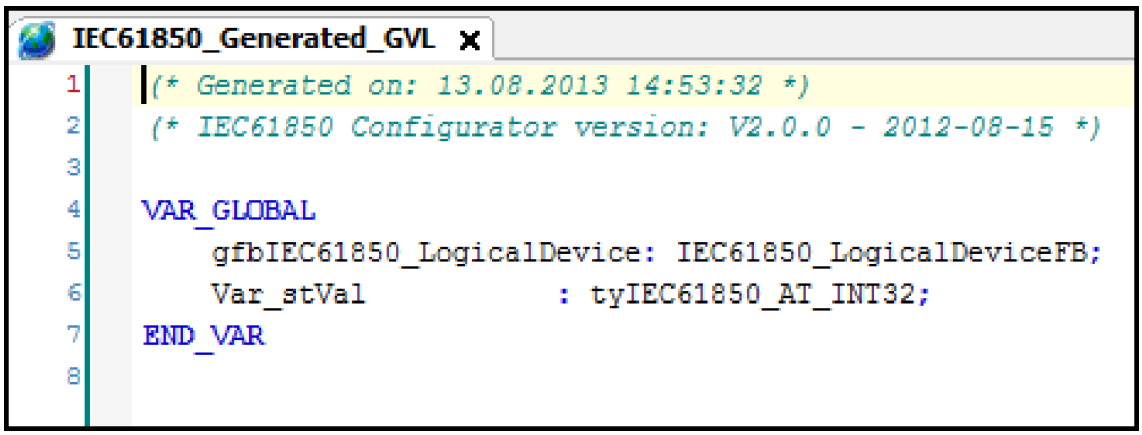


Fig. 321: IEC61850_Generated_GVL with 'Var_stVal'

Subsequent compile the application via the *"Build ➔ Build"* command.

Step 9: Connecting with an IEC 61850 Client

If the application was finished successfully you create a connection to an IEC 61850 client in this step. For this, login to the PLC and start the application via the *"Start"* command of the *"Debug"* menu. Now you connect an IEC 61850 client with the IEC 61850 Server. By the client you can read out your IEC 61850 Server configuration and the configured data sets und reports, as well as you can receive GOOSE messages and send GOOSE messages to the server.

Editor of the IEC 61850 Server

IEC 61850 Editor

You open the editor of the IEC 61850 Server with the *“Edit Object ”* command of the *“File ”* category or with a double-click on the device in the device tree.

> *If you move the mouse pointer over buttons, options or names of input fields in this editor more information about the element is displayed by the tooltip.*

The tabs of the editor::

- *Chapter 1.6.6.3.2.3.2.1 “Configuration” on page 3885*
- *Chapter 1.6.6.3.2.3.3 “DataSet” on page 3895*
- *Chapter 1.6.6.3.2.3.4 “Report” on page 3896*
- *Chapter 1.6.6.3.2.3.5 “GOOSE Publisher” on page 3898*
- *Chapter 1.6.6.3.2.3.6 “GOOSE Subscriber” on page 3900*
- *Chapter 1.6.6.3.2.3.7 “Information” on page 3902*

Configuration

Configuration

In the *“Configuration”* tab of the IEC 61850 editor you create, configure an parametrize the IED from the pool of the existing LNC and CDC types.

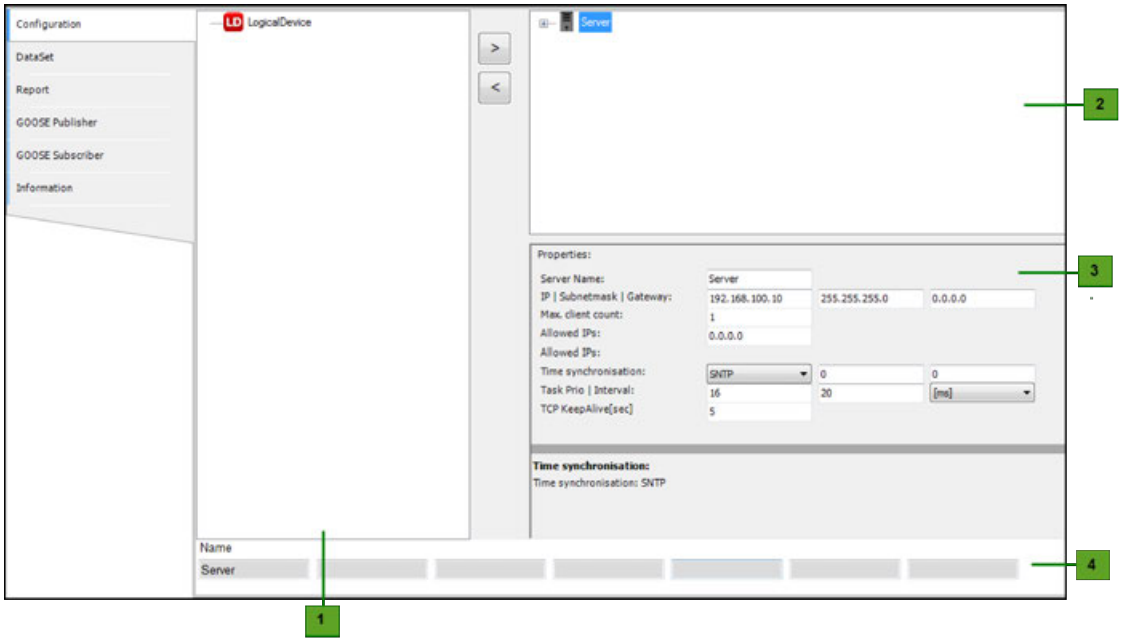


Fig. 322: 'Configuration' tab

“Configuration” is split in 4 sections:

- Section 1 and section 2 are for creating the IEC 61850 Server. In section 1 all of the choices are displayed, which can be added to the instance currently focused or to the object of the sever currently focused in section 2. In the default setting there is the Logical Device (LD) in section 1 and the server ( ) in section 2. For more information see *Chapter 1.6.6.3.2.3.2.2 "Creation of the IEC 61850 Server" on page 3886*
- In the *"Properties"* section 3 the following activities can be performed, depending on the selected objects:
  - *"Parameterization of the IEC 61850 Server" on page 3889*
  - *"Entry of a device name for the Logical Device" on page 3890*
  - *"Connecting an attribute (DA) with a CODESYS variable" on page 3890*
  - *"Entry of a node prefix for LNC instances" on page 3893*
- 4 is the status bar. For a more information see *Chapter 1.6.6.3.2.3.2.4 "Status bar" on page 3893*.

Creation of the IEC 61850 Server

Adding instances

You add an element to the server or to the marked instance below the server by selecting the element in section 1 and activating the *">"* button or by a double-click on the element

Removing instances

To delete instances below the server, mark the instance an activate the*"<"* button..

Configure the server

If you create a new configuration, the default settings in the *Chapter 1.6.6.3.2.3.2.1 "Configuration" on page 3885* tab are: the Logical Device in section 1 and the server in , section 2 .

First the *"Logical Device"* (LD) is added to the server. The Logical Device is an instance of an IED (intelligent field device). Together with the Logical Device the objects having the option 'mandatory' are added automatically. These objects added automatically can not be removed from the Logical Device.

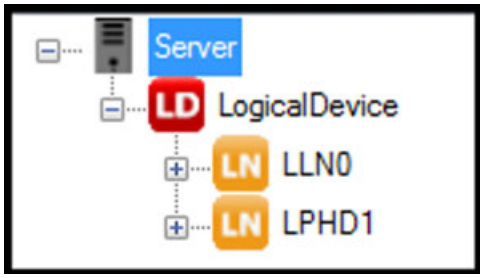


Fig. 323: Server with Logical Device an d objects 'LLN0' und ' LPHD'

Any number of logical nodes (LN) can be added to the Logical Device.

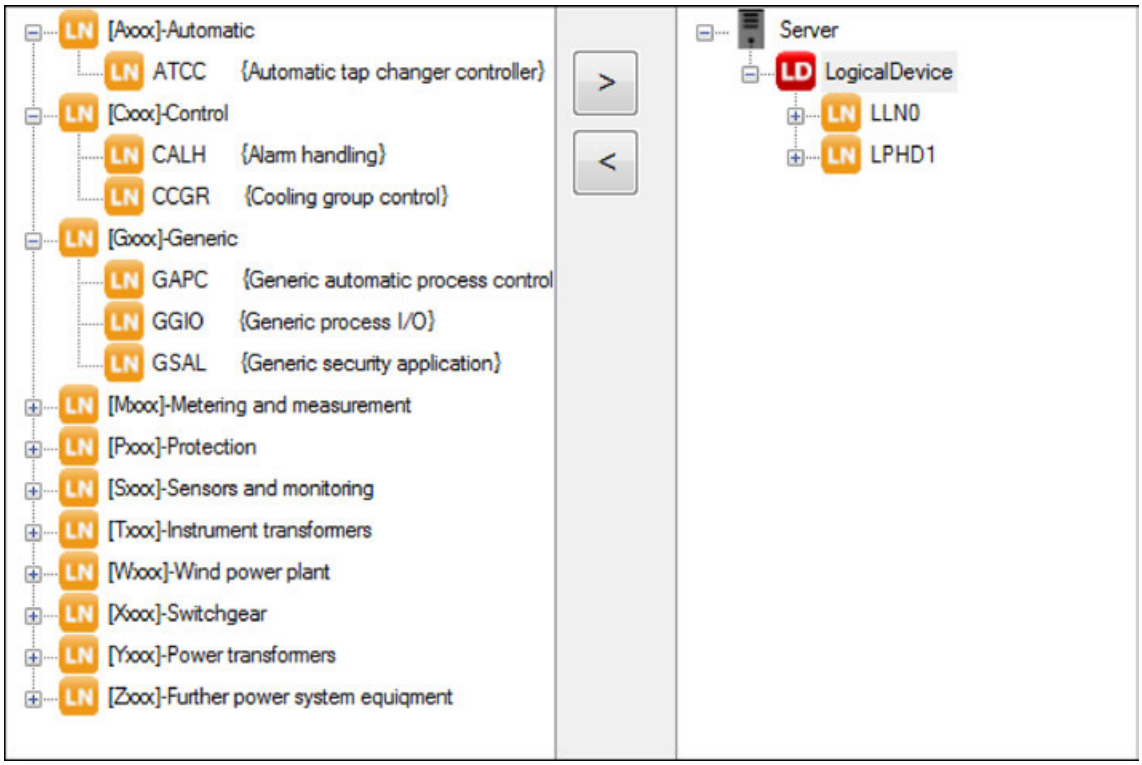


Fig. 324: 'Configuration': List of the available LACS

If you mark a LNC instance in section 2 all mandatory and all optional CDC types (data object DO) will be listed in section 1. The mandatory CDC types (Mod, Beh, Health, NamPlt, in the example) are already contained in the LNC instance (LN GGIO1) and can not be removed.

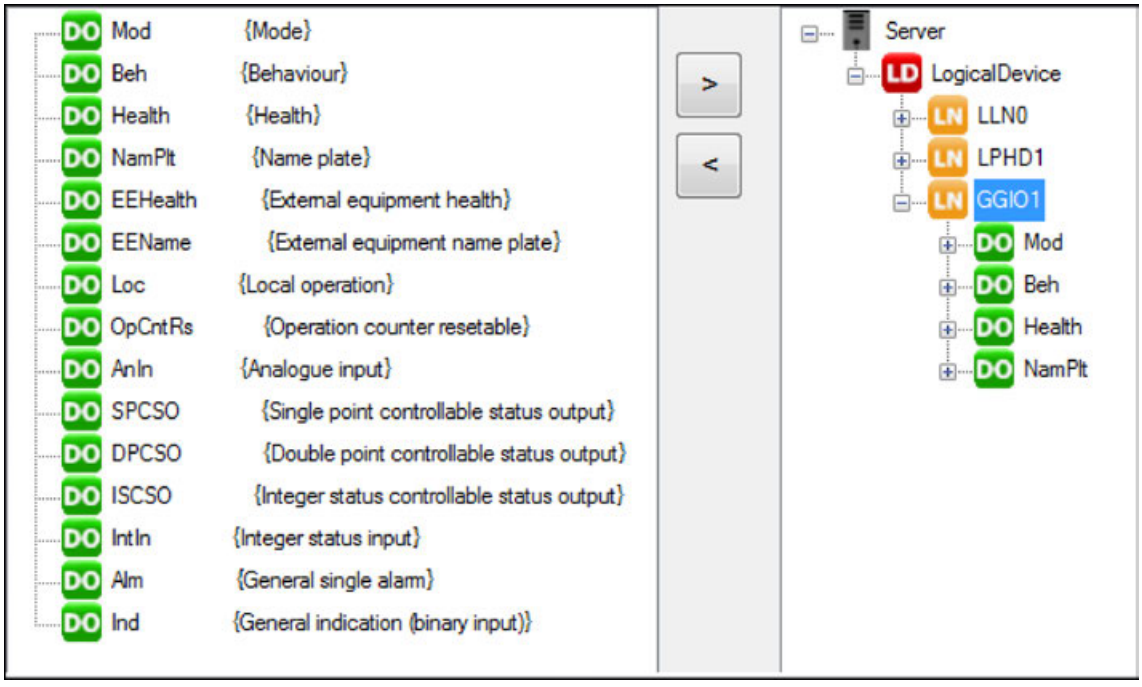


Fig. 325: 'Configuration' list of the CDCs (DO) available for the GGIO1

The DOs include the attributes (DAs)

Fig. 326: 'Configuration': list of attributes (DA) of the added to the CDC 'DO AnIn'

The attributes of the server can be connected with CODESYS variables (see *"Connecting an attribute (DA) with a CODESYS variable" on page 3890*).

Properties

3: In the *"Properties "* section of the *Chapter 1.6.6.3.2.3.2.1 "Configuration" on page 3885* tab the following functions can be performed dependent on the marked object or the marked instance:

- *"Parameterization of the IEC 61850 Server" on page 3889*
- *"Entry of a device name for the Logical Device" on page 3890*
- *"Connecting an attribute (DA) with a CODESYS variable" on page 3890*
- *"Entry of a node prefix for LNC instances" on page 3893*

> *If you move the mouse pointer over an input field or the name of an input field, you get a tooltip with a description in the window below the "Properties " (see the following figure).*

Parameterization of the IEC 61850 Server

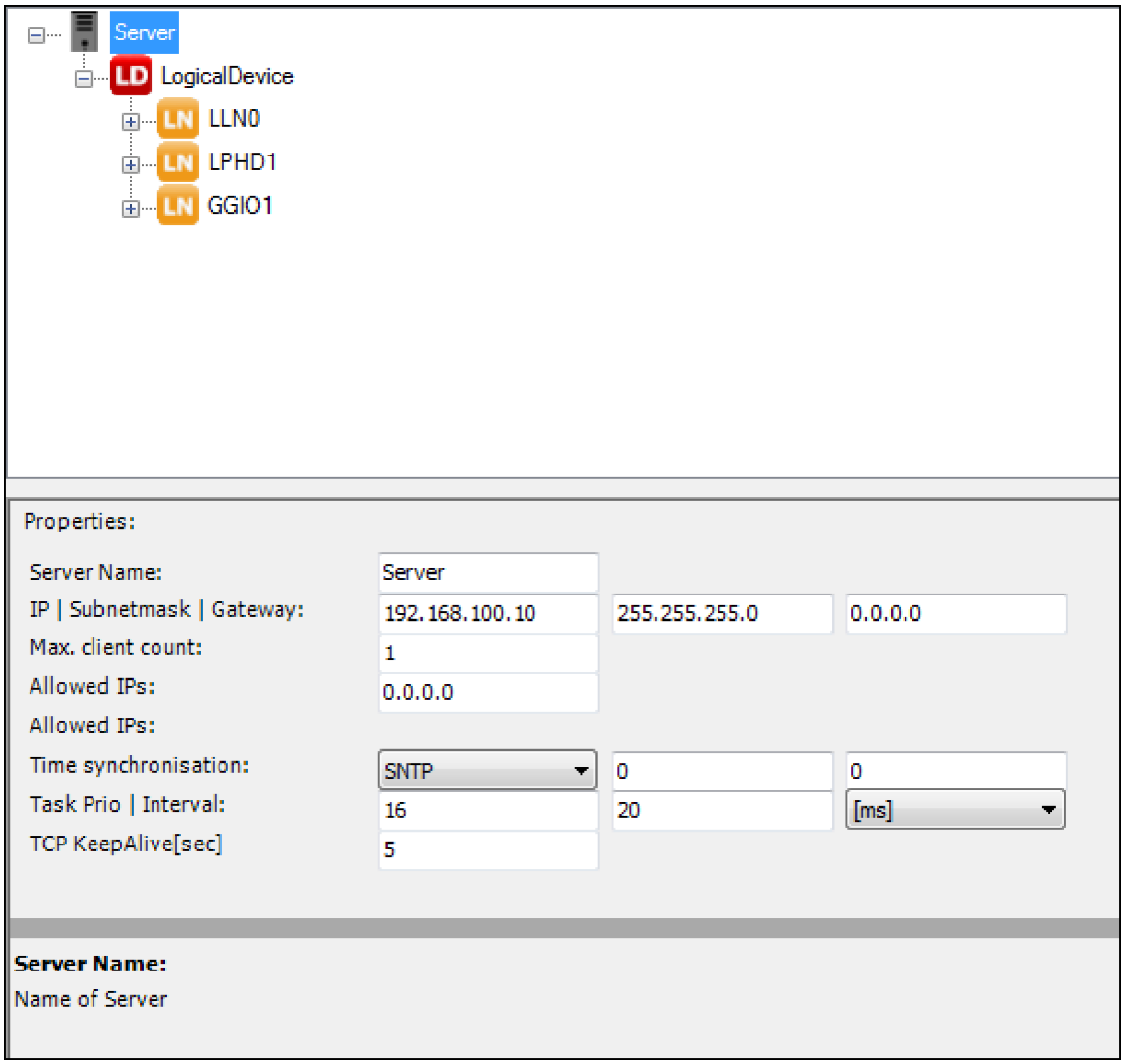


Fig. 327: 'Properties' with tooltip of 'Max. client count'

Property	Description
Server Name	Name of the server,
IP I Subnetmask I Gateway	Own IED IP address, I Subnetmask I own gateway address
Max.client count	The maximum number of clients that can connect to the IED, possible values: 1, 2, 3, 4, 5
Allowed IPs	Allowed IPs for clients 1...5 default is 0.0.0.0, whereby an IP address that equals 0.0.0.0 means that no IP address validity test will take place. If more than one client connection was selected above, additional IPs must be configured for each one. As soon as an IP address is parameterized with 0.0.0.0, testing for all connected clients is deactivated.

Property	Description
Time synchronisation	Selection: SNTP SNTP (default): SNTP time synchronization. In addition to activation, functionality must be parameterized in the device's web-based management. Currently, the SNTP time telegram does not use milliseconds, which means accuracy is measured in 1 second increments. 1. Input field **Time zone**: Offset between Greenwich (GMT)- and the local time (for Germany 1 h, for example). The value is limited between -12 an d+ 14 2. Input field: **DLS Mode**: Ratio for the mode summer/winter time changeover. Possible values: ● 0 = No automatic summertime/wintertime changeover ● 1 = Timeover from wintertime to summertime on last the Sunday in March, changeover from summertime to wintertime on the last Sunday in October
Task Prio I Interval	Task 1.input field: entry of the priority, 2.input field: entry of the interval in ms
TCP KeepAlive[sec]	The KeepAlive is to check the connection to the client.

Entry of a device name for the Logical Device

If the Logical Device LD is focused in the configured server you can entry a device name for the Logical Device in the 'Properties' section.

Connecting an attribute (DA) with a CODESYS variable

1. Select the attribute of a CDC instance below the server.
2. Enter the desired CODESYS variable name into the input field *“Monitoring Var”* in the properties section.

   Entry optionally an initial value into the input field right-side hand of the variable name.

In case of an attribute with RW-access, a *“Control Variable”* (writing access) can be entered in addition to the *“Monitoring Var”* variable (reading access) . For a more detailed description about reading and writing of variables at the IEC 61850 Server see *Chapter 1.6.6.3.2.4 “Reading and Writing from CODESYS Variables ” on page 3902*. The monitoring and control variables declared in the *“Properties”* section are displayed next to the respective attribute and at the superordinated node *“DO”* of the server tree.

By activating the *“Autom. declare ”* checkbox the variable is declared by the IEC 61850 configurator and stored in the *“IEC61850_Generated_GVL”* (of the *“IEC61850 Generated POUs”* folder) after *Chapter 1.6.6.3.2.5.1.1 “Generate code” on page 3903* of the IEC 61850 Server.

If you do not activate the “Autom. declare” checkbox you select the variable via the input assistance ([F2])) or you declare the variable yourself.

“Trigger option:” With the trigger options you set the attributes to select the events which might trigger a report. The selected trigger option is displayed in the status bar. For a description of the options see *status bar*.

The trigger option determines whether the ↳ Chapter 1.6.6.3.2.3.4 “Report” on page 3896, assigned to the ↳ Chapter 1.6.6.3.2.3.3 “DataSet” on page 3895, is sent when the value of this attribute changes.

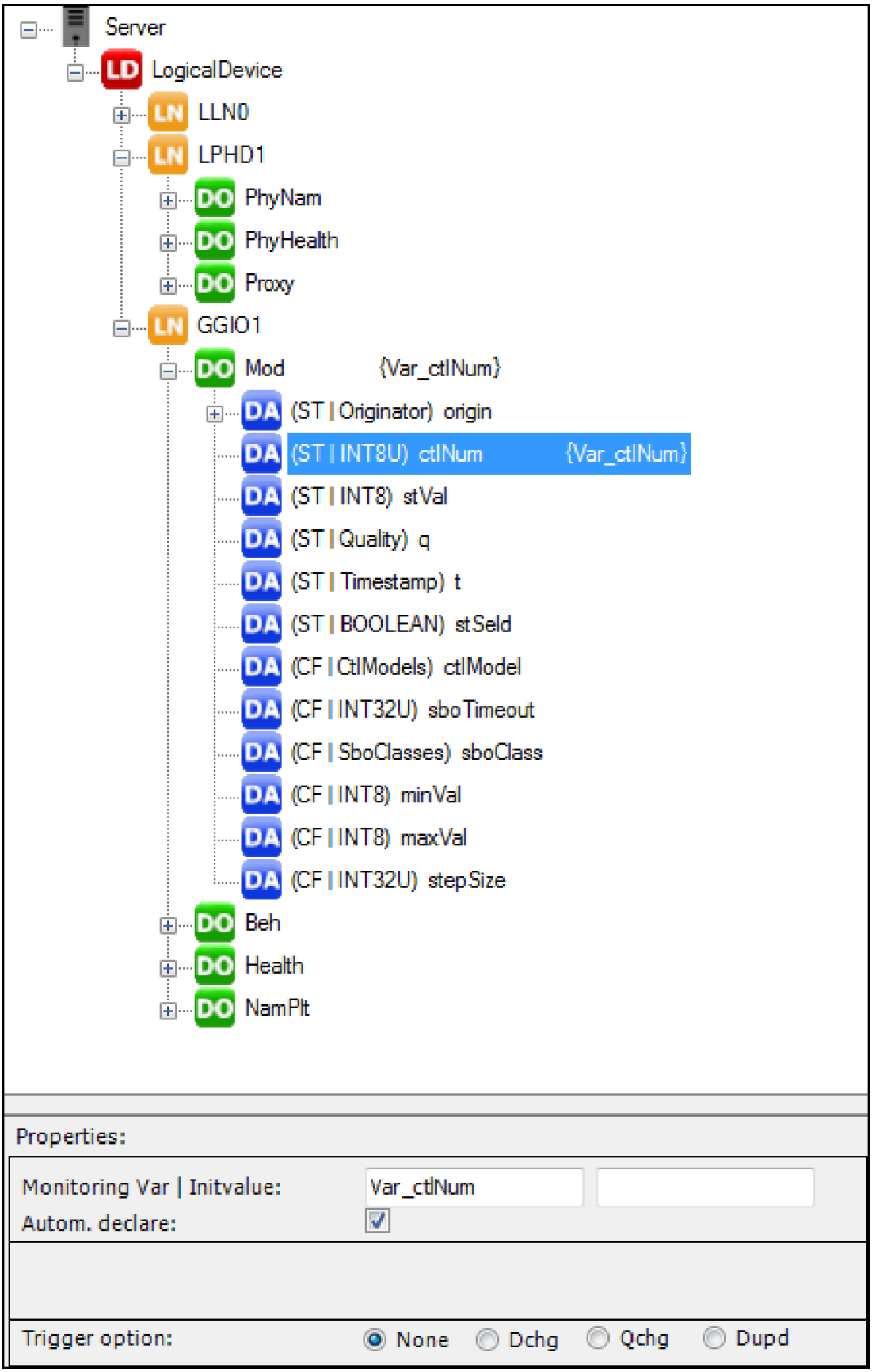


Fig. 328: 'Properties' for the attribute(DA) 'ctlNum', input fields: 'Monitoring Var' with 'Initvalue'

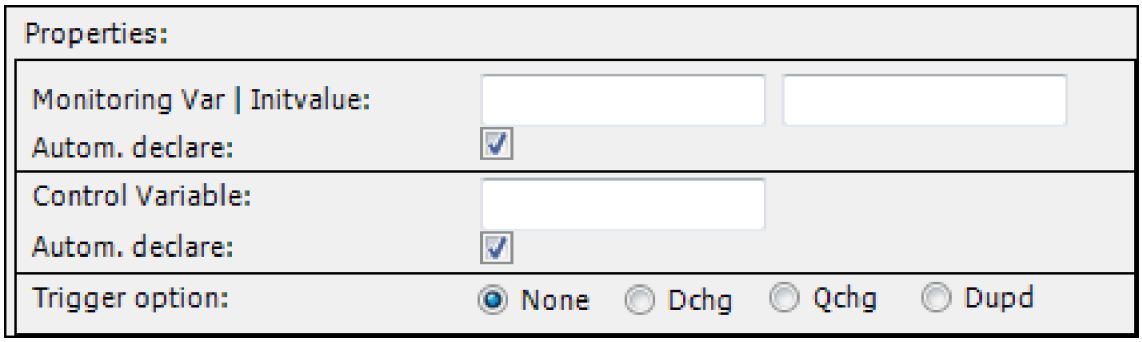


Fig. 329: 'Properties' of an attribute with RW-access: in addition input : 'Control Variable'

Entry of a node prefix for LNC instances

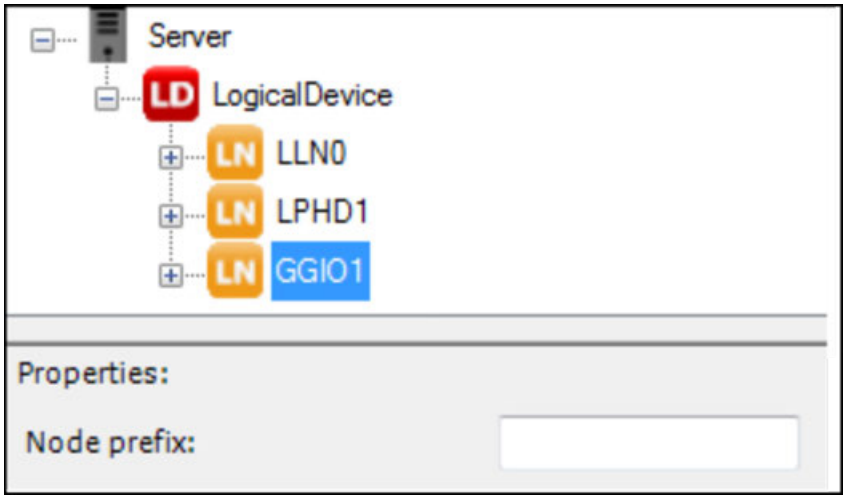


Fig. 330: 'Properties' of the LNC instance 'GGIO1' with 'Node prefix' input field

Here you enter a prefix for the selected LNC instance. The prefix is put in front of the LN name in the server tree. The prefix is displayed in the ↳ *Chapter 1.6.6.3.2.3.2.4 “Status bar” on page 3893*, too.

Status bar

In the status bar (4 of the ↳ *Further information on page 3885*) of the IEC 61850 editor you find object-specific detail information about the selected object.

Object information: Server and Logical Device

Fig. 331: Status bar for the Server

Only the information *“Name”* is displayed for the server and the Logical Device.

Object information: LN

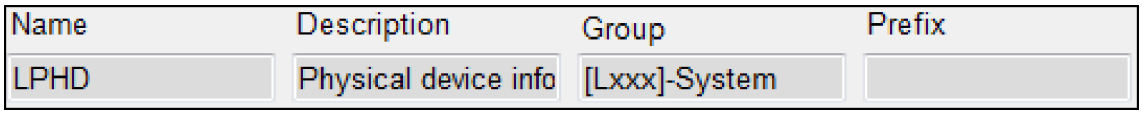


Fig. 332: Status bar for the LN instance 'LPHD'

Status	Description
"Name"	Name of the selected LN instance
"Description"	Description of the selected LN instance
"Group"	Associated group of the LN instance Examples: [Axxx]-Automatic [Cxxx]-Control [Gxxxx]-Generic ...
"Prefix"	Prefix of the LNC instance, entered by the user

Object information: Common Data Class Object (CDC Object)

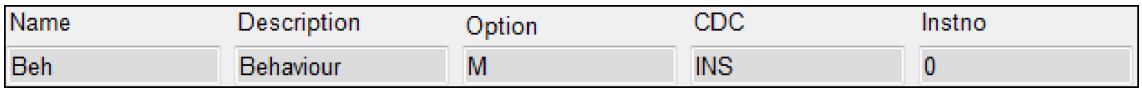


Fig. 333: Status bar of the CDC instance 'Beh'

Status	Description
"Name"	Name of the attribute
"Description"	Description of the selected CDC instance
"Option"	Option of the selected CDC- instance M = mandatory O = optional
"CDC"	Type of the selected CDC instance
"Instno"	Instance number of the CDC instance. Only optional CDCs can have instance numbers. If there is only one optional CDC, it has no instance number. Otherwise 1 to n.

Objects with the 'mandatory' options are inserted automatically when adding the Logical Device.

Object information: Attribute (DA)

Fig. 334: Status bar of the attribute 'q'

Status	Description
"Name"	Name of the attribute
"FC"	Functional Constraint of the selected attribute
"Option"	Option of the selcted attribute: M = mandatory O = optional

Status	Description
"Type"	Data type of the attribute
"Trigger Option"	Trigger option of the attribute dchg = data change dupd = data update qchg = quality change <empty> = no trigger option
"Value"	Associated variable
"Writeable"	Access for the attribute R = Read W = Write RW = Read and Write

DataSet

In this tab of the IEC 61850 editor you create and delete data sets, you assign attributes (DA) and data objects (DO) to a data set and you delete existing assignments.

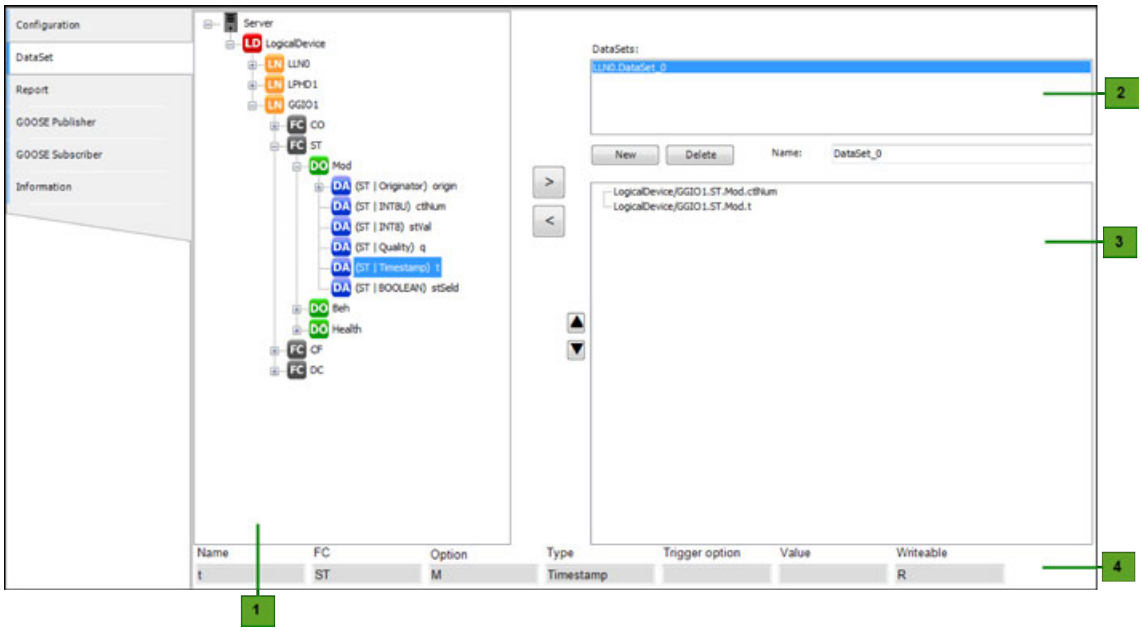


Fig. 335: 'DataSet' tab with 'LLN0.DataSet_0' data set and DAs 't' and 'ctlNum'

Structure of the tab

Section 1 displays the IEC 61850 Server created in the ↳ *Chapter 1.6.6.3.2.3.2.1 "Configuration" on page 3885* tab.

The sections 2 and 3 are for creating, editing and deleting of data sets. In section 2 the data sets are listed. In section 3 the attributes and data objects of the data set are listed, which is marked in section 2.

For more information about section 4 see ↳ *Chapter 1.6.6.3.2.3.2.4 "Status bar" on page 3893*.

> **NOTICE!**
> The order of the attributes of a data set is important for the receiving and the sending of GOOSE messages. Type and order of the entries from sender and recipient must be identical for GOOSE communication. For more information about GOOSE communication see *Chapter 1.6.6.3.2.3.5 "GOOSE Publisher" on page 3898* and *Chapter 1.6.6.3.2.3.6 "GOOSE Subscriber" on page 3900*.

Buttons:

- *"New"*: Create a new data set. This is displayed in the *"DataSets"* section and is named *"LLN0.DataSet_Suffix"*. The suffix is incremented beginning with 0 (1. DataSet: LLN0.DataSet_0 ...)
- *"Delete"*: Delete a data set: Select the desired data set in the *"DataSets"* section and activate the *"Delete"*.button.
- *">"*: Assign an attribute or a data object to the selected data set. First select the data set in section 2 then select the attribute or the data object in section 1 and activate the *">"* button.
- *"<"*: Deletes an element from a data set. First select the data set in section 2 then select the attribute or the data object in section 1 and activate the *"<"* button.
- ▼: Moves the selected entry one row up
- ▲: Moves the selected entry one row down.

Input fields

- *"Name: "* Name of the data set can be edited. The name gets the prefix *"LLN0."*.

Report

In this tab of the IEC 61850 editor you create and parameterize buffered und unbuffered reports. A report transports the data, that are assigned to it, to the connected client in the event of a trigger. Each report a data set must be assigned to.

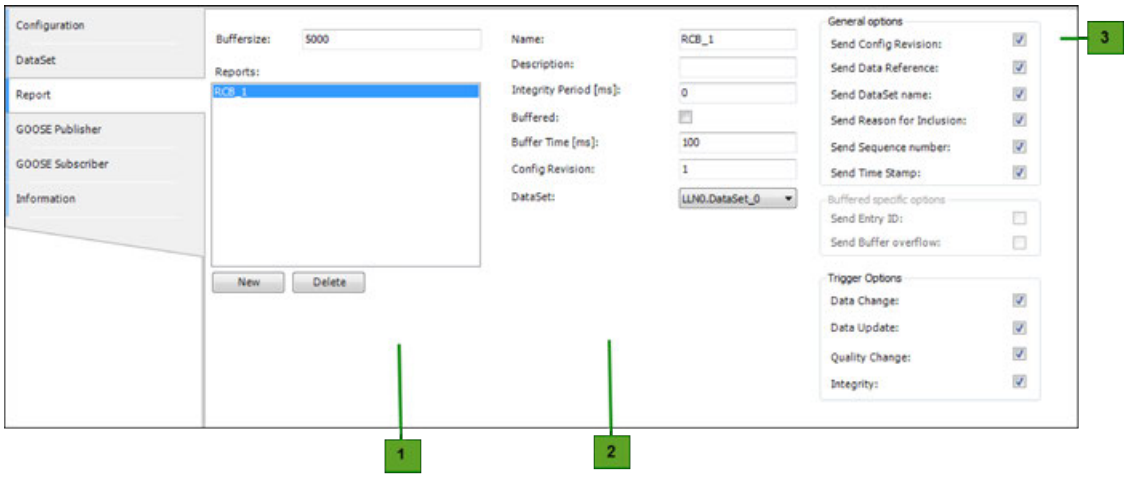


Fig. 336: 'Report' tab

In section 1 the created report control blocks (RCB) are listed. The following buttons are available:

- *"New"*: Create a new report control block.
- *"Delete"*: 'Delete the selected report control block

In section 2 you make general settings for the reporting configuration.

Table 722: General settings

Setting possibility	Description
"Buffersize"	Buffer size of buffered reports (in bytes).
"Name"	Unique Report Block name within the logical node.
"Description"	Description of the report block
"Integrity Period [ms]"	Stealthy general interrogation. After this time the referenced data set will be actuated. Time (ms) between two messages The messages are transferred cyclic, independent from other events.
"Buffered"	Enable / disable the report buffering. A buffered report stores the data, even if there's no connection to the client. In the case of an unbuffered report, the messages will get lost, if there is no connection to the client.
"Buffer Time [ms]"	Buffer time is the amount of time that the server waits to transfer a report after a given event occurs. Events that occur during this time period are collected and then transferred as a batch. If the buffer time is 0, the telegram will be sent immediately. For example, if the buffer time 10s the telegram will be sent after this time period or when the value changes the second time.
"Config Revision"	Versioning is used to identify whether or not a member was deleted from a data set or whether member order has changed. Such changes cause values to not be transferred, or cause values to be in a different location within the report. Such an event is communicated to the client with a new version number. Since all data sets are firmly defined, this identifier does not apply to the solution described here.
"DataSet"	Data set reference

Section 3 is for the setting of the following options:

- *"General options"*: Control of the reporting behavior. An activated checkbox means, that the information is transferred by the message
- *"Buffered specific options"*: can be activated, if the option *"Buffered:"* (in section 2) is activated.
- *"Trigger options"*: Determining of attributes to select the events which may trigger a message.

If the checkbox is activated the information will be transferred by the message.

Table 723: General options

Setting Possibility	Description
"Send Config Revision"	'Config Revision' information
"Send Data Reference"	Enable/disable to transfer the complete reference information, for example: LogicalDevice/GGIO1.ST.Mod.ctlNum
"Send DataSet name"	Enable/disable to transfer the data set name
"Send Reason for Inclusion"	Enable/disable to transfer the reason of transmission for each attribute
"Send Sequence Number"	Enable/disable to transfer a unique sequence number for each message
"Send Time Stamp"	Enable/disable to transfer the timestamp of transmission for each message

Table 724: Buffered specific options

Setting Possibility	Description
" Send Entry ID"	Enable/disable to transfer the 'Entry ID'
"Send Buffer overflow"	Enable/disable to transfer the message if a buffer overflow occurs.

Table 725: Trigger options

Setting Possibility	Description
"Data Change "	Enable/disable to trigger the report if a 'data change' event occured
"Data Update"	Enable/disable to trigger the report if a 'data update' event of an attribute occured.
"Quality Chance"	Enable/disable to trigger the report if a 'quality change' event of an attribute occured
"Integrity"	Enable/disable the cyclic transmission of the report independent of any datachanges (Stealthy general interrogation) . The time period has to be defined in the *"Integrity Period"* general setting.
"General Interrogation"	

Create a report control block and assign it to a data set

1. Activate the *"New"* button
2. Select the desired data set from the *"DataSet"* selection list

GOOSE Publisher

In the *"GOOSE Publisher"* tab of the IEC 61850 editor you create, edit and delete GOOSE messages. If a value changes in the selected data set, a GOOSE message is sent.

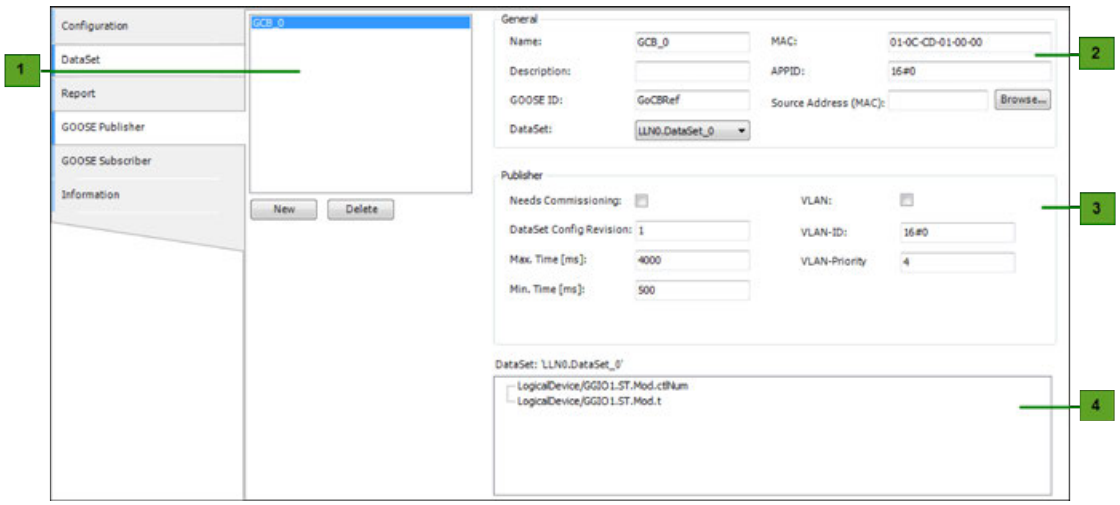


Fig. 337: 'GOOSE Publisher' tab

NOTICE!
The order of the attributes of a DataSet is important for the receiving and the sending of GOOSE messages. Type and order of the entries from sender and recipient must be identical for GOOSE communication.

NOTICE!

To receive a GOOSE message from an IED, sender an recipient must have the identical settings in the following input fields:

- *“APPID”*
- *“GOOSE-ID”*
- *“Dataset structure (with regard to order and data type of the attributes)”*

After the data set is sent, it is sent again after time interval of 500 ms. The repeat time then doubles and the data set is sent again. The data set is sent repeatedly until the value set in the *“Repeat Time”* input field is reached. The data set is then sent again at the Repeat Time interval.

Sections of the tab:

1: List of the GOOSE control blocks (GCB).

A GOOSE control block is a GOOSE message.

Buttons

- *“New”*: Create a new GOOSE cntrol block
- *“Delete”*: Delete the selected GOOSE control block.

2: General settings:

Table 726: General

Setting	Description
“Name”	Name of the GOOSE control block., editable
“Description”	Description of the GOOSE control block
“GOOSE-ID”	Unique character string of the GOOSE control block, editable
“DataSet”	Data set sent as a GOOSE message.
“MAC”	Multicast addressing Multicast addressing is used to send GOOSE messages. Addressing allows a entire group of devices to exchange data with each other. Requirement: unique address allocation of the different device groups. Valid range of values: 01-0C-CD-01-00-00....01-0C-CD-01-01-FF
“APPID”	Application-ID Number for the system-wide unique identification of a GOOSE control block. To exchange GOOSE telegrams, this number must be identical for sender and recipient. Valid range of values: 0 ... 4095
“Source Address (MAC)”	*“Browse...”* button: looks for an Ethernet Port in the network. Requirement: an existing network path to the PLC (see *ms-its:CODESYS.chm::/Communication_Settings.htm*).

3: GOOSE Publisher settings

Table 727: Publisher

Setting	Description
"Needs Commissioning"	Indicates whether the control block must be checked Value is provided from the configurator. Usage of the flag is customer-specific.
"DataSet Config Revision"	Integer value with the version of the GOOSE control block.
"Repeat Time (T0)[ms]"	Time interval during which the GOOSE telegram is valid.
"Max. Time [ms]"	Source supervision time (heartbeat cycle)
"Min. Time [ms]"	Maximum permissible send delay time of a data change
"VLAN"	'Virtual Local Area Network' Logical subnet within a physical network. Multicast messages can be passed through and filtered. The configuration is done in managed ETHERNET switches. If the *"VLAN"* checkbox is activated, values can be entered into the *"VLAN-ID"* and *"VLAN-Priority"* input fields, concerning the passed through of massages via switches.
"VLAN-ID"	A value of 0 is a non-configured VLAN in which the switch performs no filtering. This value is recommended when no logical network should be set up. Valid range of values: 0 ... 4095.
"VLAN-Priority"	Messages within a managed ETHERNET switch can be forwarded depending on the priority Valid range of values: : 0 … 7. Default value for GOOSE: 4.

4: Content of the data set assigned to the GOOSE control block.

Create an GOOSE control block and assign it to a data set

1. Activate the *"New"* button
2. Select the desired data set from the *"DataSet"* selection list.

GOOSE Subscriber

In this tab of the IEC 61850 editor you make settings for the receiving of GOOSE messages.

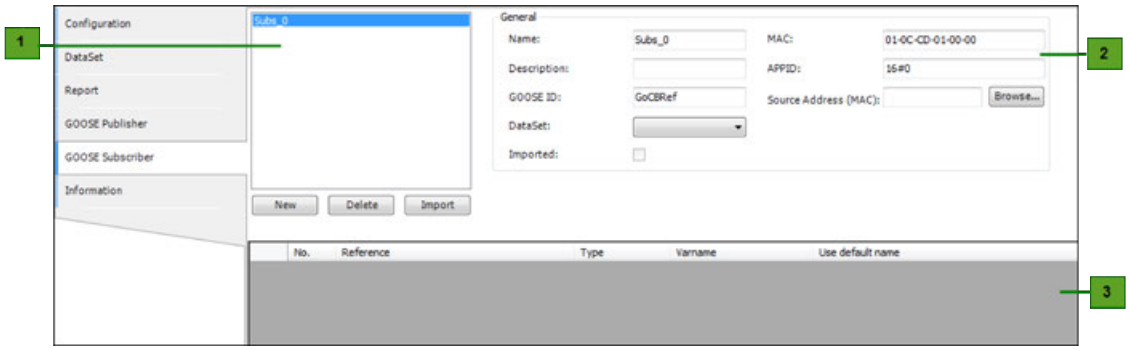


Fig. 338: 'GOOSE Subscriber' tab

NOTICE!
The order of the attributes of a data set is important for the receiving and the sending of GOOSE messages. Type and order of the entries from sender and recipient must be identical for GOOSE communication.

NOTICE!
To receive a GOOSE message from an IED, sender an recipient must have the identical settings in the following input fields:

- *"APPID"*
- *"GOOSE-ID"*
- *"DataSet"* structure (with regard to order and data type of the attributes)

Sections of this tab:

1: List of the GOOSE control blocks (GCB)

Buttons

- *"New"*: Create a new GOOSE control block
- *"Delete"*: Delete the selected GOOSE control block.
- *"Import"*: Import a GOOSE control block in the SCL format

2: General settings:

Table 728: General

Setting	Description
"Name"	Name of the GOOSE control block., editable
"Description"	Description of the GOOSE control block
"GOOSE-ID"	Unique character string of the GOOSE control block, editable
"DataSet"	Data set received as a GOOSE message.
"MAC"	Multicast addressing Multicast addressing is used to send GOOSE messages. Addressing allows a entire group of devices to exchange data with each other. Requirement: unique address allocation of the different device groups Valid range of values: 01-0C-CD-01-00-00....01-0C-CD-01-01-FF
"APPID"	Application-ID Number for the system-wide unique identification of a GOOSE control block. To exchange GOOSE telegrams, this number must be identical for sender and recipient. Valid range of values: 0 ... 4095
"Source Address (MAC)"	*"Browse..."* button: looks for an Ethernet Port in the network. Requirement: an existing network path to the PLC (see *ms-its:CODESYS.chm::/Communication_Settings.htm*).

3: List to assign GOOSE messages to global variables.

All attributes within the selected data set are listed in this list. You can assign incoming GOOSE messages to global CODESYS variables. For this, select the desired attribute in the list and edit the name of a global variable in the *"Varname"* column. If you edit a new variable name a global variable will be created, if you activate the *"Use default name"* checkbox, a variable name is generated automatically. This variable will be written by incoming GOOSE messages.

The variables will be stored *"IEC61850_Generated_GVL"* (of the *"IEC61850 Generated POUs"* folder) after ↳ *generating the code* of the IEC 61850 Server.

No.	Reference	Type	Varname	Use default name
1	LogicalDevice/GGIO1.ST.Mod.ctlNum	INT8U	Var_ctlNum	☐
2	LogicalDevice/GGIO1.ST.Mod.t	Timestamp	Subs_0_Entry1	☑

Fig. 339: Example for the variable list

Create a GOOSE control block and assign it to a data set

1. Activate the *"New"* button
2. Select the desired data set from the *"DataSet"* selection list

Information

This tab of the IEC 61850 editor shows information on the IEC 61850 Server

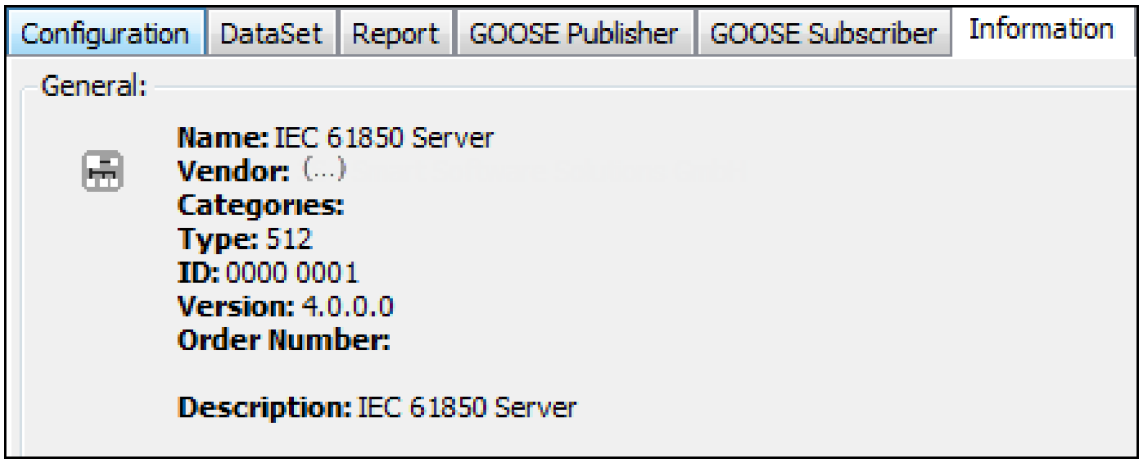


Fig. 340: 'Information' tab'

Reading and Writing from CODESYS Variables

Monitoring direction, reading

For reading in monitoring direction you connect an attribute (DA) with R-access (read) a CODESYS monitoring variable (see *"Parameterization of the IEC 61850 Server" on page 3889*).

The following dataflow variants are possible:

- from the IEC 61850 Server to the connected IEC 61850 Client to read a CODESYS monitoring variable
- from an I/O module to the IEC 61850 Server to the connected IEC 61850 client to read an I/O module pin.

Control direction, writing

For writing in control direction you connect an attribute (DA) with W-access (write) to a CODESYS control variable (see *"Parameterization of the IEC 61850 Server" on page 3889*).

The following dataflow variants are possible:

- from a connected IEC 61850 client to the IEC 61850 Server to write a CODESYS-variable
- form the connected IEC 61850 client to the IEC 61850 Server to an I/O module to write the I/O module pins.

Monitoring direction + control direction, reading an writing

It may be the case that the IEC 61850 client will read the monitoring variable of an attribute and will write the control variable of the same attribute. Monitoring variable and control variable must not be the sameCODESYS variable.

In monitoring direction the data flow takes place from the IEC 61850 Server to the connected IEC 61850 client to read the CODESYS monitoring variable.

In control direction the data flow takes place from the connected IEC 61850 client to the IEC 61850 Server to write the CODESYS control variable.

Menu Command sorted by Categories

IEC61850

Generate code

Symbol: 

On activating the *“Generate code”* command of the *“IEC61850”* category the code generation is started and the generated IEC 61850 code is stored in the folder *“IEC61850 Generated POUs”* in the device tree.

Export Server

This command of the *“IEC61850”* category exports the current configuration. In the *“Safe as ”* dialog select the format filter:

- XML files: for IEC 61850 format with all specific data, variable mapping, for example
- SCL-Files: for IEC 61850 format to export data to other IEC 61850 tools

If you have changed the configuration since the latest code generation, you will be asked whether new code should be generated before export.

Import Server

This command of the *“IEC61850”* category discards the current configuration and imports an new configruation. In the *“Save as ”* dialog select the format filter:

- XML files: for IEC 61850 format with all specific data, variable mapping, for example
- SCL Files: for IEC 61850 format to import data from other IEC 61850 tools

Options

The *“Options”* command of the *“IEC61850”* category opens a dialog for the setting of different display options for the IEC 61850 configurator.

Option	Description
“Show FC besides data attribute”	Display option, shows functional constraint of attribute as a comment.
“Show type besides data attribute”	Display option, shows type of attribute as a comment
“Show trigger option besides data attribute”	Display option, shows trigger option of attribute as a comment
“Show description besides data objects”	Display option, shows description of attribute as a comment
“Enable SCL Private block”	

Option	Description
“Select all Data Objects ”	Debug-Option: Selection of all data objects (DO)
“Select all Data Attributes”	Debug-Option: Selection of all data attributes (DA)

Reset

This command of the *“IEC61850”* category deletes the whole current configuration and all objects of the current application created via the *“Generate code”* command.

Logical Name Classes (LNC)

The following LNCs are available for the configuration of the IEC 61850 Server

Name	Description
	Automatic Control Functions
ATCC	Automatic tap changer controller
	Control
CALH	Alarm handling
CCGR	Cooling group control
	Generic Functions
GAPC	Generic automatic process control
GGIO	Generic process I/O
GSAL	Generic security application
	System
LLN0	Logical Node Zero
LPHD	Physical device information
	Metering and measurement
MMTR	Metering
MMXN	Non phase related Measurement
MMXU	Measurement
MSQI	Sequence and imbalance
MSTA	Metering Statistics
	Protection
PDIF	Differential
PFRC	Rate of change of frequency
PHAR	Harmonic restraint
PHIZ	Ground detector
PIOC	Instantaneous overcurrent
PMRI	Motor restart inhibition
PMSS	Motor starting time supervision
PTOV	Overvoltage
	Sensors and monitoring

Name	Description
SARC	Monitoring and diagnostics for arcs
SIMG	Insulation medium supervision (gas)
SIML	Insulation medium supervision (liquid)
	Instrument transformers
TCTR	Current transformer
TVTR	Voltage transformer
	Wind power plant (IEC61400-25)
WALM	Wind power plant alarm information
WAPC	Wind power plant active power control
WCNV	Wind turbine converter information
WGEN	Wind turbine generator information
WMET	Wind power plant meteorological information
WNAC	Wind turbine nacelle information
WROT	Wind turbine rotor information
WAPC	Wind power plant reactive power control information
WOW	Wind turbine tower information
WTRF	Wind turbine transformer information
WTRM	Wind turbine transmission information
WTUR	Wind turbine general information
WYAY	Wind turbine yawing information
	X-Switchgear Functions
XCBR	Circuit Breaker
XSWI	Circuit Switch
	Y-Power Transformers
YEFN	Ground fault neutralizer (Petersen Coil)
YLTC	Tap Changer
YPSH	Power Shunt
YPTR	Power Transformer
	Further power system equipment
ZAXN	Auxiliary network
ZBAT	Battery
ZCAP	Capacitor Bank
ZCON	Converter
ZGEN	Generator
ZGIL	Gas Insulated Line
ZLIN	Power Overhead Line
ZMOT	Motor
ZREA	Reactor
ZRRC	Rotating reactive component
ZSAR	Surge arrestor

Name	Description
ZTCF	Thyristor controlled frequency converter
ZTCR	Thyristor controlled reactive converter

IEC 61850 Functionalities

Models Conformance

Functionality	Support	Comment
Logical device	yes	
Logical node	yes	
Data	yes	
DataSet	yes	
Substitution	yes	
Setting group control	no	
Reporting		
Buffered report control	yes	
Sequence number	yes	
Report time stamp	yes	
Reason for inclusion	yes	
DataSet name	yes	
Data reference	yes	
Buffer overflow	yes	
Entry-ID	yes	
Buffer Time	yes	
Integrity Period	yes	
General Interrogation	yes	
Config Revision	yes	
Unbuffered report control	yes	
Sequence number	yes	
Report time stamp	yes	
Reason for inclusion	yes	
DataSet name	yes	
Date reference	yes	
Buffer Time	yes	
Integrity Period	yes	
General Interrogation	yes	
Config Revision	yes	
Logging	no	
Log Control	no	

Functionality	Support	Comment
Log	no	
Control	yes	Only Operate
GOOSE	yes	
GSSE	no	
Multicast SVC	no	
Unicast SVC	no	
Time	yes	
File Transfer	no	
Maximum number of simultaneously client connections	5	Parameter in the configurator. 1...5
Maximum MMS PDU size	45 000	
Time synchronisation	yes	SNTP
SCL File support	yes	Ex-/Import in CODESYS IEC 61850 Server TOOL

Service Conformance

Table 729: Server

Sevices	Support	Comment
ServerDirectory	yes	

Table 730: Application association

Sevices	Support	Comment
Associate	yes	
Abort	yes	
Release	yes	

Table 731: Logical Device

Sevices	Support	Comment
LogicalDeviceDirectory	yes	

Table 732: Logical Node

Sevices	Support	Comment
LogicalNodeDirectory	yes	
GetAllDataValues	yes	

Table 733: Data

Sevices	Support	Comment
GetDataValues	yes	
SetDataValues	yes	

Sevices	Support	Comment
GetDataDirectory	yes	
GetDataDefinition	yes	

Table 734: DataSet

Sevices	Support	Comment
GetDataSetValues	yes	
SetDataSetValues	yes	
CreateDataSet	no	
DeleteDataSet	no	
GetDataSetDirectory	yes	

Table 735: Substitution

Sevices	Support	Comment
SetDataValues	yes	

Table 736: Reporting

Sevices	Support	Comment
Buffered report control block (BRCB)		
Report	yes	
data-change (dchg)	yes	
qchg-change (qchg)	yes	
data-update (dupd)	yes	
GetBRCBValues	yes	
SetBRCBValues	yes	
Unbuffered report control block (URCB)		
Report	yes	
data-change (dchg)	yes	
qchg-change (qchg)	yes	
data-update (dupd)	yes	
GetURCBValues	yes	
SetURCBValues	yes	

Table 737: Generic substation event model (GSE)

Sevices	Support	Comment
GOOSE-CONTROL-BLOCK		
SendGOOSEMessage	yes	
GetReference	no	
GetGOOSEElementNumber	no	
GetGoCBValues	yes	
SetGoCBValues	yes	

Sevices	Support	Comment
GSSE-Control-Block		
SendGSSEMessage	no	
GetReference	no	
GetGSSEElementNumber	no	
GetGsCBValues	no	
SetGsCBValues	no	

Table 738: Control

Sevices	Support	Comment
Select	no	
SelectWithValue	no	
Cancel	no	
Operate	yes	
Command-Termination	no	
TimeActivated-Operate	no	

Table 739: Time

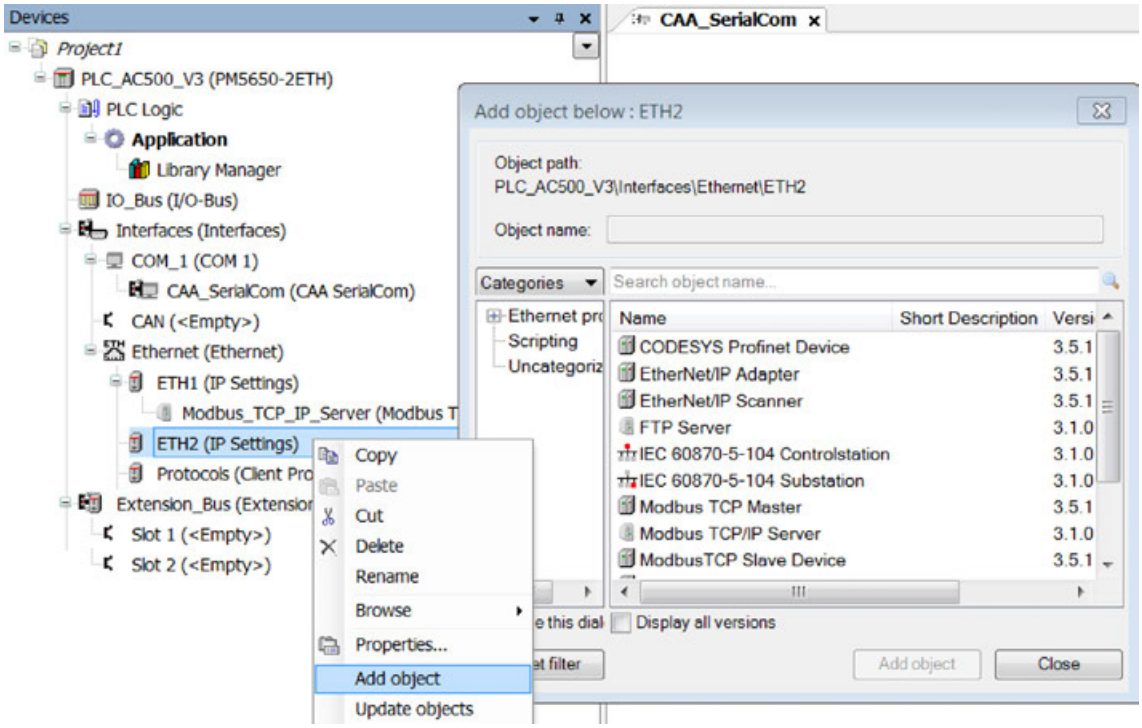
Sevices	Support	Comment		
Time resolution of internal clock	7	nearest power of 10 ms		
Time accuracy of internal clock		TL (ms)	(low accuracy)	T3 < 7 (only Ed2)
		T0 (ms)	(<= 10 ms)	7 <= T3 < 10
		T1 (µs)	(<= 1ms)	10 <= T3 < 13
		T2 (µs)	(<= 100 µs)	13 <= T3 < 15
		T3 (µs)	(<= 25 µs)	15 <= T3 < 18
		T4 (µs)	(<= 25 µs)	15 <= T3 < 18
		T5 (µs)	(<= 1 µs)	T3 >= 20
Supported TimeStamp resolution	7	nearest power of 10 ms		

1.6.6.3.3 Modbus protocol

Modbus on TCP/IP protocol

Configuration of Modbus TCP/IP server

Adding a Modbus TCP/IP server to device tree



A Modbus TCP/IP Server instance can be added to any specific Ethernet interface / IP address. Each interface supports max. one instance of *“Modbus TCP/IP Server”*. Other protocols can be added in parallel.

1. Right click on ETH interface and click *“Add object”*.
   ⇨ The window *“Add object below: ETH”* appears.
2. Select *“Modbus TCP/IP Server”* and click *“Add object”*.
   ⇨ The node *“Modbus_TCP_IP_Server”* is added.

Setting the parameters of Modbus_TCP_IP_Server

Parameter	Type	Value	Default Value	Unit	Description
Byte order	Enumeration of BYTE	Big endian	Big endian		Big endian = 1; Little endian = 0
Port	WORD(1..65535)	502	502		TCP Port on which the Server listens
Startup behaviour	Enumeration of BYTE	Active	Active		Startup behaviour
Behaviour in state inactive	Enumeration of BYTE	No activity	No activity		Behaviour in state inactive
Disable write to %MB0.x from	WORD(0..65535)	0	0		Disable write access beginning with byte in area %MB0.x
Disable write to %MB0.x to	WORD(0..65535)	0	0		Disable write access ending with byte in area %MB0.x
Disable read to %MB0.x from	WORD(0..65535)	0	0		Disable read access beginning with byte in area %MB0.x
Disable read to %MB0.x to	WORD(0..65535)	0	0		Disable read access ending with byte in area %MB0.x

Byte Order

Format/Endianess for the transmission of WORD values (register) within the request/response telegram (default: *“Big Endian”*).

Port

TCP Port on which the Server listens.

Startup Behaviour

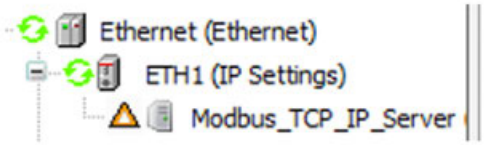
This parameter specifies how the Server behaves when configuration data is loaded (e.g. on download). It's default value is *"Active"*. This means the Server is immediately addressable after configuration has been performed. In case the Server should be activated later on during run time by means of function block *ModTcpServOnOff* this parameter value has to be set to *"No activity"*. Parameter *Behaviour in state inactive* then specifies the Server's behaviour during the inactive phase.

Behaviour in state inactive

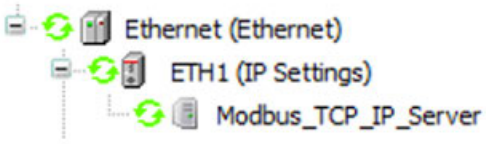
This parameter specifies how the Server behaves in inactive state. This state may be set at the very beginning (parameter *Startup Behaviour* = *"No activity"*) and/or requested during run time calling function block *ModTcpServOnOff*. It's default value is *"No activity"*. This means the Server is not addressable at all (no listening socket on TCP/IP) when it is inactive. Using this setting, any requests by Modbus TCP clients lead to the result *Failed to connect to Server* or *Timeout*. All other parameter values make the Server respond with an exception code to any requests by Modbus TCP clients.

The presentation of the icon next to the Modbus TCP Server in the device tree depends on the state of the Server:

Inactive



Active



Attention:
Exception code 9 is actually not defined by Modbus specification. This may cause problems using a different Modbus TCP client than AC500 V3.

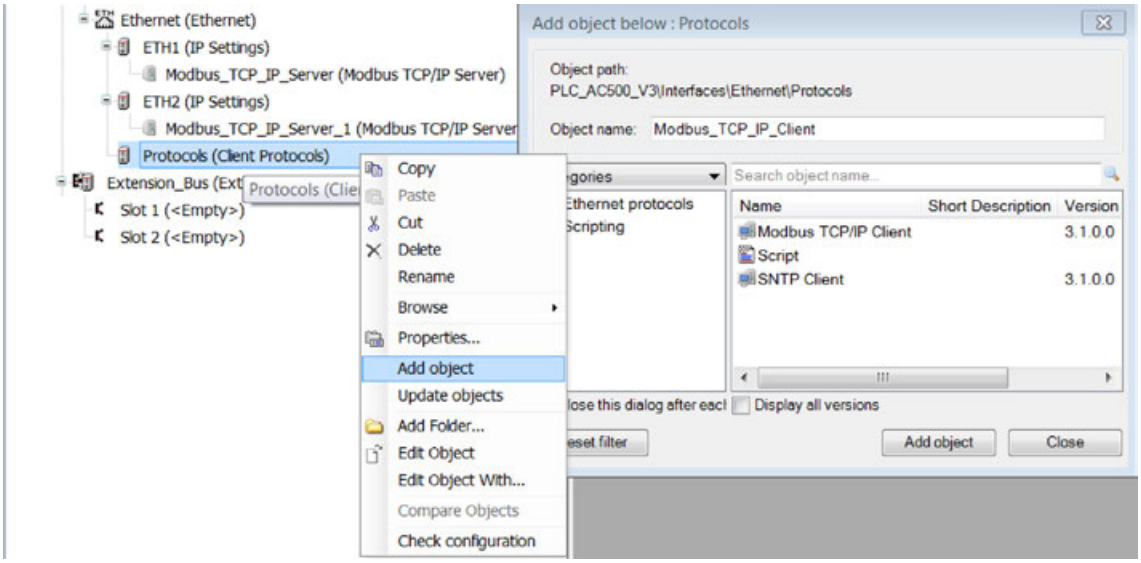
Disable

Parameter	Default	Value	Description
Disable write to %MB from	0	0 ... 65535	Disable write access starting at %MBx
Disable write to %MB to	0	0 ... 65535	Disable write access up to %MBx
Disable read from %MB from	0	0 ... 65535	Disable read access starting at %MBx
Disable read from %MBx to	0	0 ... 65535	Disable read access up at %MBx

It is possible to disable read and/or write access to individual segments. Reading/writing is disabled beginning at the set start address and is valid up to the set end address (inclusive).

Configuration of Modbus TCP/IP client

Adding a Modbus TCP/IP client to the device tree



The *"Modbus_TCP_IP_Client"* instance has to be added to the common Ethernet client protocols' node. This node supports max. one instance of Modbus TCP/IP client. Other protocols can be added in parallel.

1. Right click on the node*"Protocols"* and click *"Add object"*.
   ⇨ The window*"Add object below: Protocols"* appears.
2. Select *"Modbus TCP/IP Client"* and click *"Add object"*.
   ⇨ Node *"Modbus_TCP_IP_Client"* is added.

Depending on a Server's IP-Address the client sends it's requests via the Ethernet interfaces available.

Setting the parameters of Modbus_TCP_IP_Client

Modbus TCP/IP client does not have any parameters.

Modbus on RTU protocol

Protocol description can be found in the chapter for Serial interfaces ↳ *Chapter 1.6.6.2.14.1 "Configuring Modbus RTU on serial interface" on page 3793*.

1.6.6.3.4 NTP/SNTP protocol

Introduction of the NTP/SNTP protocol

AC500 V3 support the NTP and the SNTP protocol ((Simple) Network Time Protocol). Compared to SNTP, the NTP protocol achieves higher accuracy in time synchronization, meeting advanced requirements for accuracy and reliability of a PLC solution. In case a configured NTP protocol cannot be used, SNTP protocol is used as a fallback solution.

The protocols NTP and SNTP provide the functionality to synchronize the clock of a PLC to an external time source. For further information and specification of the protocol please refer to the document *RFC4330*.

The following modes are supported by the implementation of the AC500 V3 PLC:

- (S)NTP client
- (S)NTP server
- (S)NTP client and server

The function block *PmSntpInfo* can be used to read diagnosis information of the protocol.

Refer to the documentation of the library *ABB_Pm_AC500.lib* for further information.

- *If a high precision of system time is wanted, use a fully functional NTP server or at least an SNTP server with a high-precision time-source (e.g. DCF-77 receiver). Avoid cascading several levels of (S)NTP server / (S)NTP clients.*
- *Client requests are normally sent at intervals depending on the frequency tolerance of the client clock and the required accuracy. However, under no conditions requests should be sent at less than one minute intervals (see RFC 4330). Keep that in mind when setting polling-interval of the (S)NTP client, especially if a huge amount of clients use one single server.*
- *Be sure not to use broadcast or multicast addresses as server or backup-server since current (S)NTP implementation does not support manycast mode.*

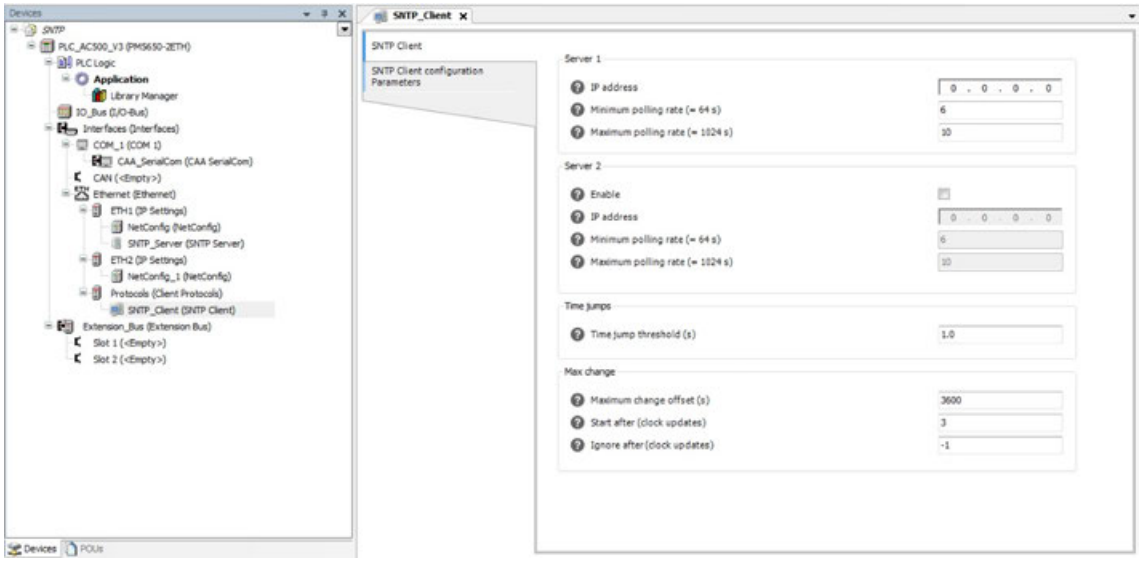
Configuration of the (S)NTP protocol

(S)NTP client configuration

Implementation of (S)NTP client and (S)NTP server is based on protocol version 4.

For (S)NTP client configuration add a new object *“SNTP Client”* under *“Protocols (Client Protocols)”*.

For a PLC only one instance of an (S)NTP client is possible.



The following parameters are available:

Not all parameters are shown in the user interface.

It should not be necessary to change the default values of the other parameters for the most applications.

But there is the possibility to edit them in the generic parameter editor.

Server 1

Parameter	Default	Value	Description
IP address	0.0.0.0	Valid IP address	IP address of a server which is used as external time source.
Minimum polling rate	6 2^6 = 64 s	-4 ... 24	Specifies the lower limit of the polling rate. It is calculated as power of 2 and has the unit [s]. The actual polling rate is determined by the protocol itself but it will not be lower than this limit.
Maximum polling rate	10 2^10 = 1024 s	0 ... 24	Specifies the upper limit of the polling rate. It is calculated as power of 2 and has the unit [s]. The actual polling rate is determined by the protocol itself but it will not be higher than this limit.

Server 2

Parameter	Default	Value	Description
Enable	FALSE	TRUE or FALSE	Enable server
IP address	0.0.0.0	Valid IP address	IP address of a server which is used as external time source.
Minimum polling rate	6 2^6 = 64 s	-4 ... 24	Specifies the lower limit of the polling rate. It is calculated as power of 2 and has the unit [s]. The actual polling rate is determined by the protocol itself but it will not be lower than this limit.
Maximum polling rate	10 2^10 = 1024 s	0 ... 24	Specifies the upper limit of the polling rate. It is calculated as power of 2 and has the unit [s]. The actual polling rate is determined by the protocol itself but it will not be higher than this limit.

Time jumps

Parameter	Default	Value	Description
Enable	TRUE	TRUE or FALSE	Enables the option 'Time jumps'
Time jump threshold	1.0 s	0 ... 3.403e+38	Specifies the threshold value for time steps in seconds
Limit	-1	-1 … 2147483647	Number of first clock updates after that this option is deactivated, a negative value activates this option permanently.

Max change

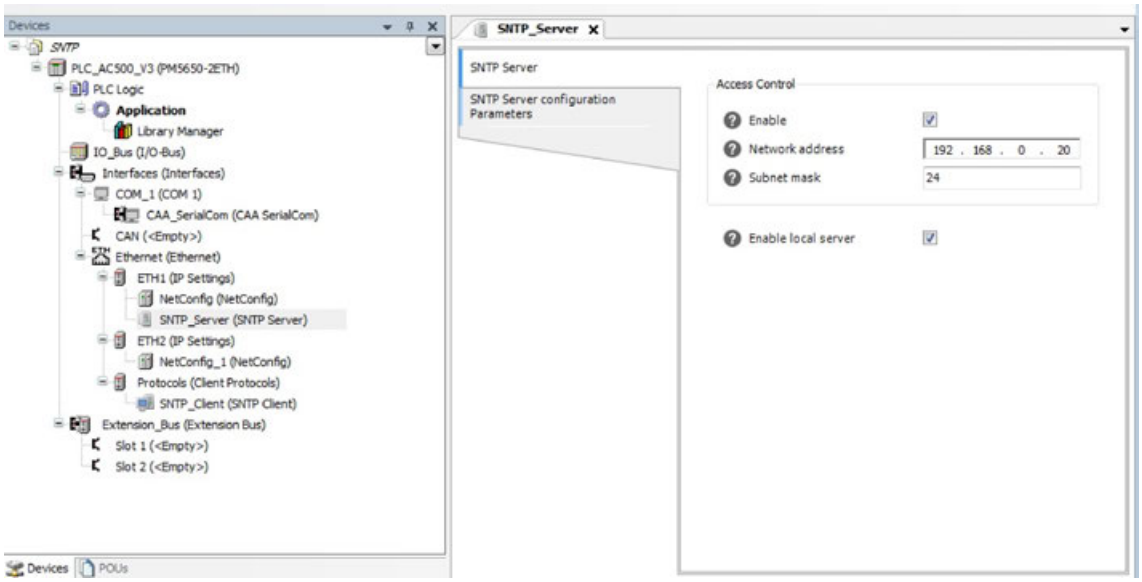
Parameter	Default	Value	Description
Enable	TRUE	TRUE or FALSE	Enables the option 'Max change'
Maximum change offset	3600 s	0 … 4294967295	Maximum allowed clock offset in seconds
Start after	3	0 … 4294967295	Specifies the number of first clock updates after that this option is activated
Ignore after	-1	-1 … 2147483647	Specifies the number of ignored clock updates which exceed the maximum offset. The protocol will be stopped when this value will be exceed. It is never stopped when a negative value is set.

(S)NTP server configuration

Implementation of (S)NTP client and (S)NTP server is based on protocol version 4.

For (S)NTP server configuration add a new object *“SNTP Server”* under of the available *“Ethernet interfaces (ETH1-ETHn)”*.

For a PLC only one instance of an (S)NTP server is possible.



The following parameters are available:

Not all parameters are shown in the user interface.

It should not be necessary to change the default values of the other parameters for the most applications.

But there is the possibility to edit them in the generic parameter editor.

Access control

Parameter	Default	Value	Description
Enable	FALSE	TRUE or FALSE	Enables Access Control
Network address	0.0.0.0	Valid IP address	Network address of allowed clients
Subnet mask	24	8 ... 32	Subnet mask of the network address

Local server

Parameter	Default	Value	Description
Enable	FALSE	TRUE or FALSE	This option enables the protocol to run as local server. That means without synchronization to an external time source.
Stratum	10	1 ... 15	Stratum of the server when it is used as local server
Distance	1 s	0 … 3.403e+38	Distances in seconds of the server when it is used as local server
Orphan	FALSE	TRUE or FALSE	Enables or disables the orphan mode

1.6.6.3.5 FTP server

Configuration of FTP server

As of SystemFW 3.1.0 the FTP server is listening only on the Ethernet interface, which the protocol is configured on. It is not possible to have an FTP server on both Ethernet interfaces.

AC500 V3 PLCs only support explicit authorization. AC500 V3 PLCs do not support implicit authorization.

1. Under *“Ethernet -> ETH [1,2,...]”* add a new object and select *“FTP Server”* from the list.
2. Double-click the *“FTP_Server ”* item to open FTP server configuration and change the default settings of the parameters, if required.

Parameter		Default	Value	Description
FTP Server				
	Port	21	21	Do not change the default setting. The parameter specifies the port which is used to connect to the FTP server on the PLC.
	Sessions	1	1...4	Enter the max. number of allowed simultaneous and parallel connections to the FTP server. Each session uses one socket. Note: Some FTP clients require several connections to work.
Passwords		-	-	Set each user’s passwords for login. No entry = no password.
	system	-	-	System RAM disk
	sdcard	-	-	Inserted memory card.
	userdisk	-	-	User section of the flash disk.
	flashdisk	-	-	Only available with PM5675-2ETH

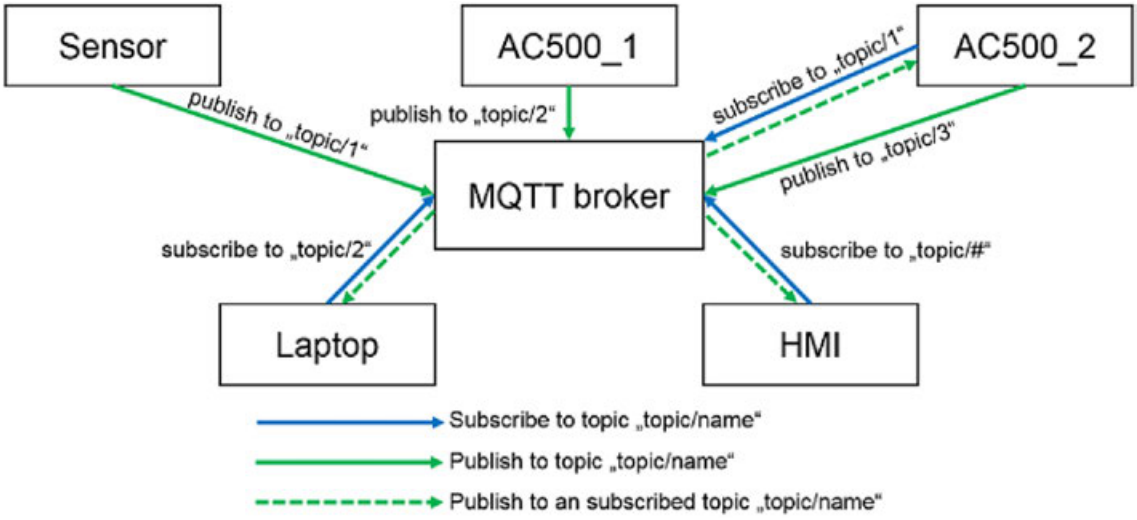
1.6.6.3.6 MQTT client protocol

System technology

The MQTT protocol is a lightweight communication protocol which is widely used on the internet to connect embedded device to the cloud.

The MQTT (Message Queuing Telemetry Transport) client library allows to integrate an AC500 processor module to act as a client in the MQTT protocol. Thus, it is possible to exchange data between the AC500 and other devices connected to the MQTT network.

In the figure below, there is an MQTT network with one broker (MQTT broker in the middle) and five clients. The figure shows the main functions of MQTT to send and receive data: publish and subscribe. The clients can publish messages with a specific topic to send data (e. g. the temperature of a connected sensor with a timestamp) to the MQTT broker. For example, the client “AC500_1” publishes a message to topic “topic/2”. On the other hand side clients can also subscribe to topics to receive data. For example, the client “Laptop” has subscribed topic “topic/2”. So all messages with the topic “topic/2” which has been published to the MQTT broker will be sent immediately to the client “Laptop”. This creates a message flow from the client "AC500_1" to the laptop.

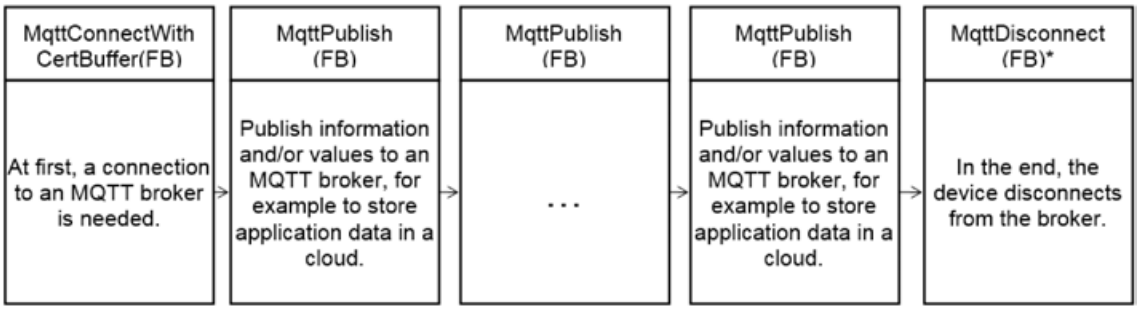


To realize the MQTT behavior, there are several function blocks implemented in the *Chapter 1.5.11 “MQTT client library” on page 2376*.

Table 740: Function blocks overview

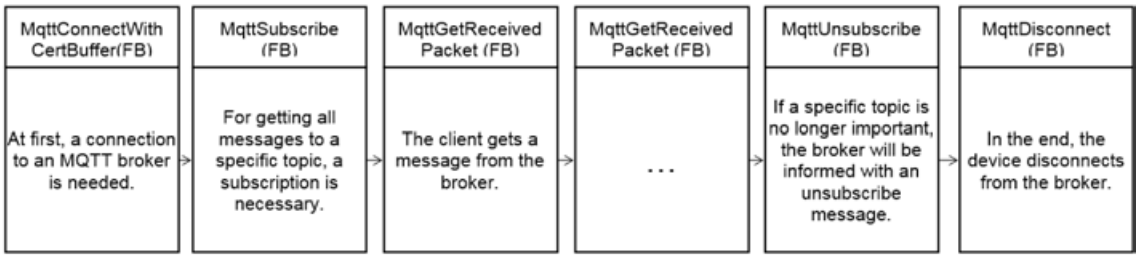
Function Block	Description
MqttConnectWithCertBuffer MqttConnectWithCertFile	Every MQTT use case starts with establishing a connection to an MQTT broker. Therefore, a connection structure needs to be created. The connection structure is used to identify the connection for subsequent operations like publish or subscribe. It is possible to establish an SSL connection. Using an SSL connection, at least a certificate for the server is needed. Certificates can be loaded from a buffer (program variable) or a file which is stored on the PLC.
MqttGetReceivedPacket MqttPing MqttPublish MqttSubscribe MqttUnsubscribe	These function blocks can be used on an established MQTT connection to realize the desired use case.
MqttDisconnect	This function block is the end of each use case.

One MQTT send use case could look like this:



*) It makes sense for several publish messages in a row (e. g. one message per second) not always open a new connection.

One MQTT receive use case could look like this:



TLS version

The MQTT client uses the TLS version 1.2.

Configuration in Automation Builder

For the MQTT client no configuration is needed.

Configuration in CODESYS

All function blocks have to be called in tasks with cyclically processing.

You can use the function blocks with:

- PLC_PRG with automatic task configuration or manual task configuration.
- One single program or different programs.
- One single task or different tasks.

With different programs assigned to different tasks you can define different cycle times and priorities.

Limitations

- No persistent session. After an interrupted connection, the client needs to subscribe on topics again in case of reconnect.
- One connection (MQTT_CONNECTION) cannot be shared between multiple tasks. Different connections can be used by different tasks or even within the same task.
- Only one FB can operate on a single connection at the same time. Always wait for the FB to complete before calling the next FB. To use two different FB's in parallel (like publish and receive) it is necessary to have two different connections, otherwise they must be called one after the other.

Hardware

The MQTT protocol requires AC500 devices with integrated Ethernet.

Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples.

MQTT can be used using the MQTT client library or JSON. An introduction to programming with JSON is given in the *application example*.

1.6.6.3.7 AC500 V3 secure protocols

Introduction

The following protocols can be secured using certificates:

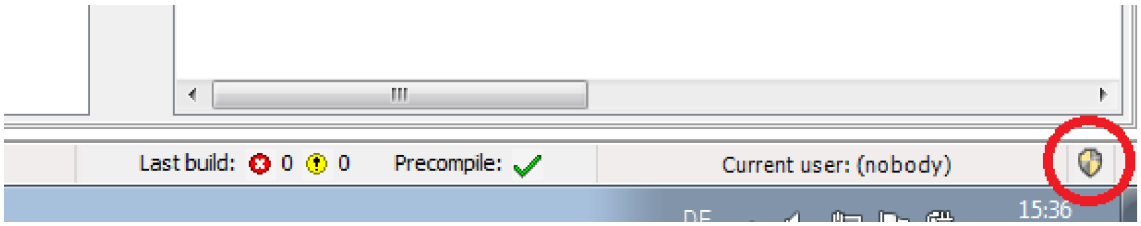
- Communication between Automation Builder and the PLC (e.g. Programming, Monitoring)
- Communication between the PLC's webserver and visualization clients (browsers)
- Communication between the PLC's FTP server and FTP clients
- Communication between the OPC UA server and OPC UA clients

As a prerequisite to enable secure communication on one or more protocols, the required certificates need to be present on the PLC.

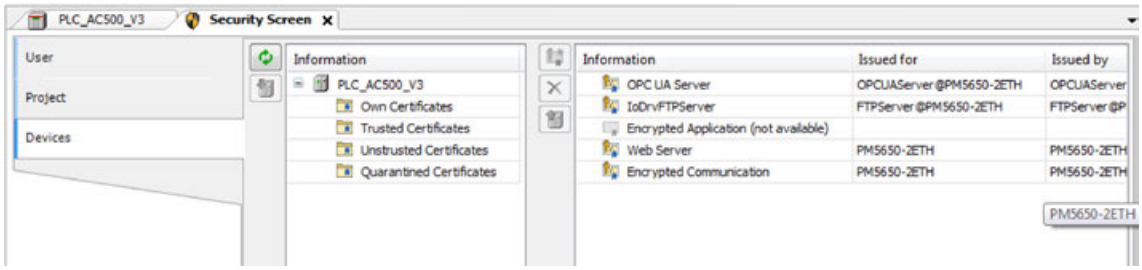
> *For security reasons ABB does not encourage the use of self-signed certificates. ABB shall not be held liable for any damage or loss that arises due to the use of self-signed certificates on AC500 PLCs.*
>
> *Self-signed certificates protect against eavesdropping if used correctly. They do not offer any secure means of authentication.*

Certificate handling

Automation Builder offers a convenient *"Security Screen"* to manage certificates on connected PLCs.



It can be accessed through the *shield icon* on the lower right corner of the main window:



Use the tab *"Devices"* to manage certificates on the PLC.

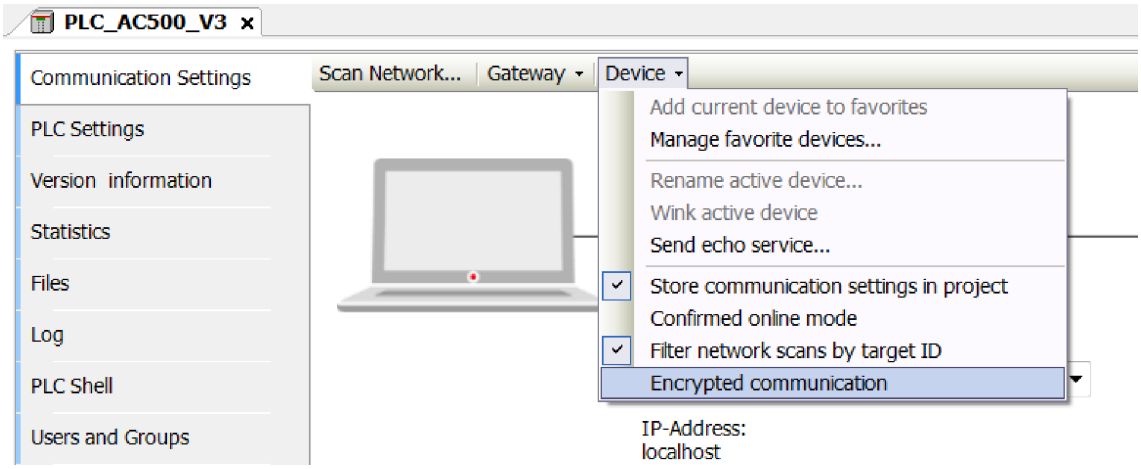
It offers to:

- show certificates available on the PLC
- import and export certificates
- create new (self-signed) certificates
- trust or untrust certificates

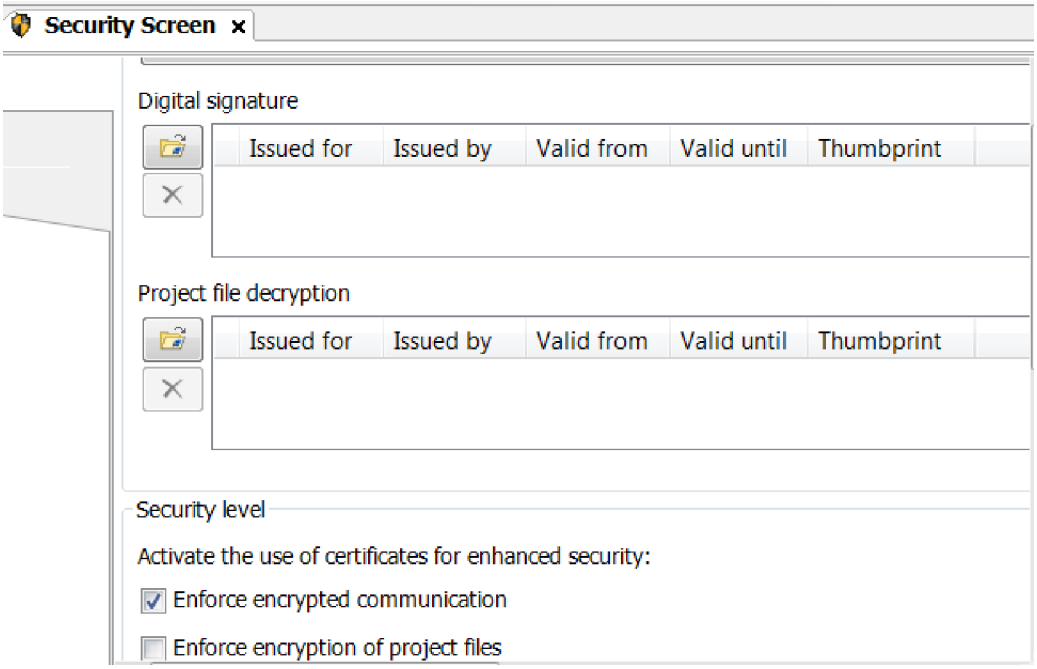
Configuring secure protocols

Encrypted communication between Automation Builder and the PLC

Via tab *"Communication Settings"*



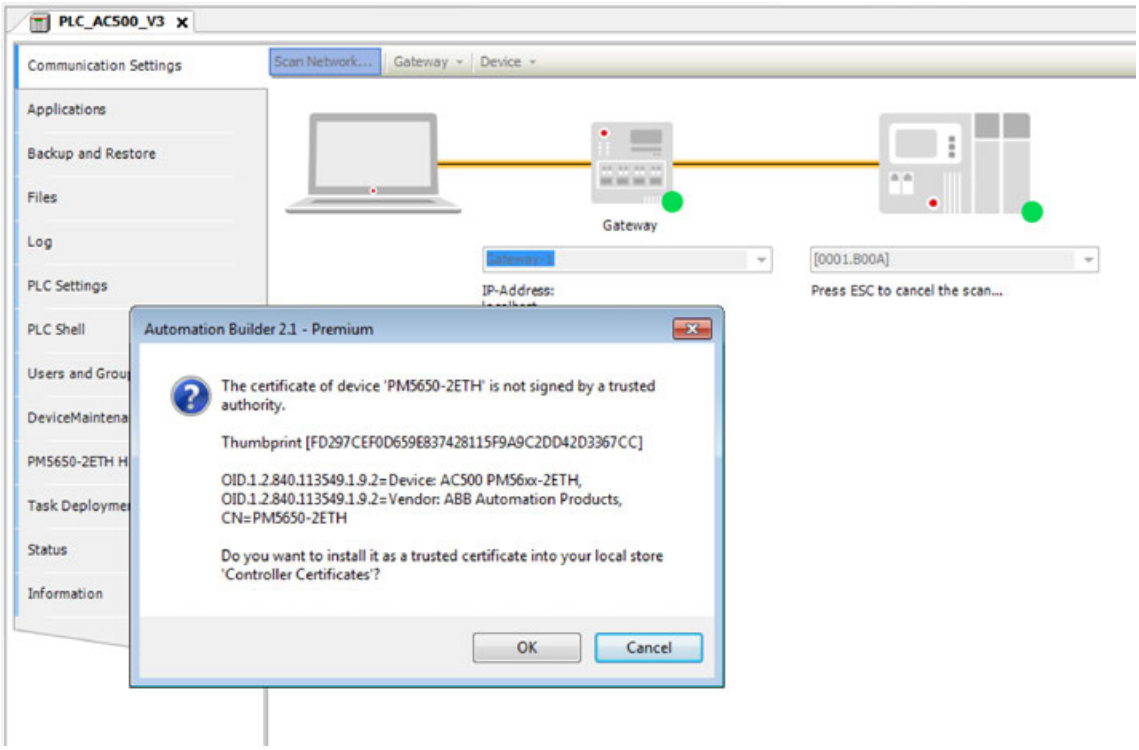
Via *"Security Screen"* in Automation Builder



Install a trusted certificate

> *Ensure the PLC clock is set to the current time and date when using certificates on the PLC. Otherwise the certificate cannot be used to secure a protocol (see also ↳ Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479 and ↳ Chapter 1.6.6.3.4.2.1 “(S)NTP client configuration” on page 3913).*

When trying to log in or when you set the PLC as active path, there will be a one-time pop-up asking you to add the PLC’s certificate to the trusted certificates:



After trusting the PLC’s certificate, the communication between the Automation Builder and the PLC is now encrypted.

This is shown by additional yellow lines around the communication path on the *“Communication Settings”* page.

Secure web server

1. Generate or import a certificate for the web server

> *Ensure the PLC clock is set to the current time and date when using certificates on the PLC. Otherwise the certificate cannot be used to secure a protocol (see also ↳ Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479 and ↳ Chapter 1.6.6.3.4.2.1 “(S)NTP client configuration” on page 3913).*

2. Attach a web server node to either ETH1 or ETH2 or both and configure security mode.
   ⇨ This will automatically insert a visualization into the project.

   The available modes of operation are:

   - http only
   - https only
   - Both (http and https)
   - Redirect http to https

3. Download and set the PLC to RUN.

4. Connect to the web server using the configured method: https://<your PLC´s IP address>/webvisu.htm.

> *In case you are using a self-signed certificate, your browser will show some warnings.*
>
> *If you are aware of the risks of self-signed certificates, this can be ignored.*
>
> *Further information on page 3920*

Secure FTP

1. Import a certificate to the PLC for FTP or create a self-signed certificate.

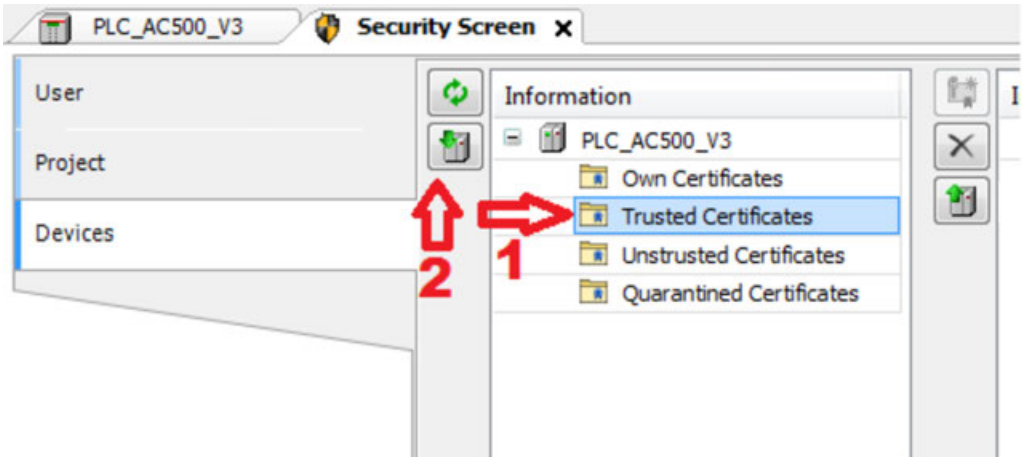
> *Ensure the PLC clock is set to the current time and date when using certificates on the PLC. Otherwise the certificate cannot be used to secure a protocol (see also Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479 and Chapter 1.6.6.3.4.2.1 “(S)NTP client configuration” on page 3913).*

2. Add an FTP server to either ETH1 or ETH2
3. Set the parameter *“Security Mode”* to either *“BOTH”* or *“FTPS only”*.
   ⇨ You can use any FTP client that supports FTPS explicit mode (FTPES).

> *In case you are using a self-signed certificate, the FTP client will show some warnings or notice that it does not know the certificate and wants you to check it.*
>
> *Further information on page 3920*

OPC UA secure

OPC UA uses mutual authentication, which means that both partners must have their own certificate and know the other’s certificate, before being able to establish a connection!



1. Create a new certificate in your OPC UA client.

   Ensure the PLC clock is set to the current time and date when using certificates on the PLC. Otherwise the certificate cannot be used to secure a protocol (see also ↳ Chapter 1.6.5.1.4.2 “AC500 battery” on page 3479 and ↳ Chapter 1.6.6.3.4.2.1 “(S)NTP client configuration” on page 3913).

2. Import that certificate to the *“Trusted Certificates”* in your PLC using the *“Security Screen”*.
3. Import a certificate for the OPC UA server on the PLC or create a self-signed certificate.
4. Export that cert to the PC and provide it as a trusted certificate to your OPC UA client.
5. Reboot the PLC and check that it is in RUN and both certificates are on the PLC (via the *“Security Screen”*).
6. Add the PLC as OPC UA server in your OPC UA client.
7. Connect to the OPC UA Server.

   ⇨ You can interact normal with the UA server.

In case you are using a self-signed certificate, you will see some warning message (depending on the OPC UA client).

If you are aware of the risks of self-signed certificates, this can be ignored.

↳ Further information on page 3920

The certificate warnings will only go away when using a certificate from a trusted certification authority or a certificate derived from this by an intermediate certification authority (e.g. a company CA).

That process is done via PLCShell command “cert-createcsr”, then getting the file from the PLC via the filebrowser tab in “cert/export” and getting that signing request turned into a real certificate by a certification authority.

Import the certificate generated by your certification authority using the security screen.

1.6.6.3.8 KNX configurator

Refer to the general description for information about the following tabs of the device editor.

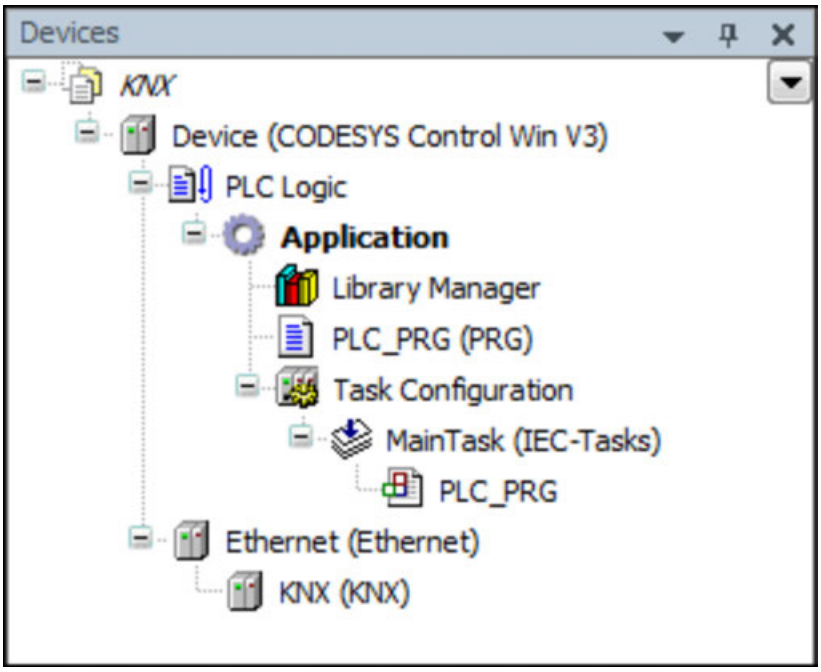
- *↳ Chapter 1.4.1.20.2.8.11 “Tab '<device name> I/O Mapping'” on page 854*
- *↳ Chapter 1.4.1.20.2.8.12 “Tab '<device name> IEC Objects'” on page 859*
- *↳ Chapter 1.4.1.20.2.8.3 “Tab 'Parameters'” on page 844*
- *↳ Chapter 1.4.1.20.2.8.18 “Tab 'Status'” on page 870*
- *↳ Chapter 1.4.1.20.2.8.19 “Tab 'Information'” on page 870*

Only in the case of special features is there an additional help page for the specific device editor.

If the "<device name> Parameters" tab is not shown, then select the “Show generic device configuration editors” option in the CODESYS options (“Device Editor” category).

With the KNX editor from CODESYS, you define the communication objects of your building automation. The communication objects are exported and made available to the ETS5 program. Linking the communication objects to the different KNX devices is performed exclusively in the ETS5 program. Therefore, only the objects are generated in CODESYS. The objects are linked to variables from the PLC program by means of *"I/O mapping"*.

You add an Ethernet adapter below the controller. Then you add the KNX device below the adapter. You can insert only one KNX device per controller.



See also

- ↳ *Chapter 1.4.1.20.4.13.6 "Dialog 'Options' - 'Device Editor'" on page 1190*

ETS5 Software - 'DCA' Plug-In

Linking the communication objects of the different KNX devices is performed exclusively in the ETS5 program. To do this, you need the ETS5 software (light or professional version). You also need the KNX product file available from KNX.

Programming steps

1. Create a project in CODESYS.
2. Download the CODESYS project to the controller.
   ⇨ The CRC is also downloaded to the controller.
3. Create an export file in CODESYS.
   ⇨ The CRC is also saved in the export file.
4. Read the export file into the configuration of ETS5.
5. Parameterize the objects in ETS5.
6. Start the program on the controller.
7. Transfer the KNX configuration to the controller.
   ⇨ The CRC is also transferred. The runtime system checks whether or not both CRCs match. When they match, the KNX device is identified as functional by the green arrow. If not, then an error is issued in the logger. In case of error, the process data (inputs/outputs) is not updated.

Tab 'KNX - General'

Object: KNX

The tab in the configurator of the KNX editor shows an overview of all communication objects. The I/Os of the communication objects are applied automatically to the I/O mapping.

Entries can be edited directly in the table or in the *“Communication object”* dialog. Existing entries can be copied via copy&paste. The next free channel number is used automatically in this case.

Table 741: “Address settings”

“Add”	Opens the *“Communication object”* dialog for adding objects
“Edit”	Opens the *“Communication object”* dialog for editing objects
“Delete”	Deletes the selected communication objects
“Export to ETS”	Exports the list of communication objects in an XML file. This file can be imported by ETS5 if ETS5 has the DCA plug-in installed. Note: The command is also available in the context menu when the KNX node is selected.
“Export to ETS”	Exports the communication objects in a CSV file
“Import CSV”	Imports the communication objects from a CSV file
“Identification”	CRC of the communication object. This must be identical to the CRC in ETS5.

Dialog 'Communication object'

“Number of group object”	Unique channel number. Gaps in the numbering is permitted. If the channel number is already assigned, then an error text is displayed and the *“OK”* button is disabled.
“Type”	Determines whether or not the object in CODESYS is used as *“Input”* or *“Output”*.
“Data point type”	The data types (DPT = Data Point Types) are specified in the KNX standard. In CODESYS, a selection of the most common data types is available. Only the basic data type can be selected, without units (for example, DPT9.*).
“Name of group object”	Any object name. Depending on the data type, a predefined text is automatically added.
“Function of group object”	Any function name. Depending on the data type, a predefined text is automatically added.
“Watchdog Timeout”	If no new message has been received after this time has elapsed, then the status bit `Timeout` is set.

Tab 'I/O Mapping'

Object: KNX

The I/O channels are generated for each communication object:

Table 742: General I/Os

Signal	Description
"Program LED Status"	This input is set by the ETS program. The signal can be used for identifying a special controller when several controllers are used (for example, by switching a LED). The status is also set when the *"Program Button"* is set to `TRUE`. Then the device is in programming mode. As soon as ETS5 has successfully set the physical address, this input switches to `FALSE`.
"Program Button"	The *"Program Button"* is needed for assigning the physical address from ETS5. If the output is set to `TRUE`, then the device is in programming mode and then ETS5 can assign the address specified there to the device.

Table 743: I/O channels of the communication object

Signal	Description
"Status byte"	Status byte as defined in the KNX stack. This allows you to determine in the application whether or not data has been received. The status can be reset by means of the `ResetStatusFlags` method.
"Trigger/Disable Cyclic, send on change" *"Trigger Output"*	Depending on the configuration in the ETS program, this output has the following function: ● If at least one of the options *"send on difference"*, *"send on change"*, or *"Cyclic sending"* is enabled, then the output is defined as deactivation. If it is set to TRUE, then cyclic sending or send on change is stopped. ● If none of the options *"send on difference"*, *"send on change"*, or *"Cyclic sending"* is enabled, then sending is triggered by a rising edge.
"Value"	Value for the input or outputs – depending on the corresponding communication object.

ETS5 - Tab 'Parameter'

The parameter page of the ETS5 configuration software is available only after you have imported the CODESYS configuration file. The parameter page is where you define the sending behavior of the values.

Table 744: "General Information"

"Default gateway"	Default gateway for sending
"Telegram rate"	Sending rate of telegrams Note: This restriction should be applied in exceptional cases only, because this causes the reaction times to be extended.
"Project title"	In CODESYS, these parameters can be defined in the project information. They are imported to ETS5 in the XML file and displayed here. *"Identifier"*: CRC of the configuration. The CRC is also displayed in CODESYS and must be identical to the CRC displayed here so that communication can be started.
"Application date"	
"Identifier"	
"Version"	
"Application state"	
"Description"	

The objects are subdivided into groups of ten (1 .. 10, 11 .. 20, 21 .. 30, etc.). A maximum of 1000 communication objects is possible.

Table 745: “Object 1 .. 10”

“<type>”	Type of the object The parameter cannot be changed.
“Communication direction”	*“Output (PLC to KNX)”*: The value is sent from the CODESYS controller to the KNX object. For this communication direction, more settings are possible (*“Send condition”*, etc.). *“Input (KNX to PLC)”*: The value is sent to the CODESYS controller. The parameter cannot be changed.
“Send condition”	*“No automatic sending”*: No send when value is changed *“Send on change”*: Send each time value is changed *“Send on difference”*: Send when the change in value corresponds to at least the value for *“Sending difference”*.
“Sending difference”	Requirement: *“Send condition”* is *“send on difference”*. The value is passed when its change is at least this value.
“Cyclic sending”	*“Disable”*: No cyclic sending *“Enable (seconds)”*, *“Enable (minutes)”*: Cyclic sending – regardless of the *“Send condition”*.
“Cycle time [hh:mm:ss]”	Rate for cyclic sending (in hours/minutes/seconds) Requirement: *“Cyclic sending”* is set to *“Enable (seconds)”*.
“Cycle time [hh:mm]”	Rate for cyclic sending (in hours/minutes) Requirement: *“Cyclic sending”* is set to *“Enable (minutes)”*.

1.6.6.3.9 BACnet-BC

Introduction to BACnet

BACnet is a standardized data communication protocol for Building Automation and Control networks as defined in the ANSI/ASHRAE standard 135 and ISO 16484-5.

The advantage is interoperability between devices of different vendors.

The BACnet protocol defines services to allow communication between devices. Examples include 'Who is', 'I am', 'Who has' and 'I have' for device and object search and identification, *“Read Property”* and *“Write Property”* for the exchange of data, up to more complex services for alarm and event management, scheduling and trending.

The BACnet protocol defines a number of object types on which the services operate. Each object is characterized by its properties.

The BACnet objects are combined in a BACnet device. A BACnet device represents the functionality of a physical device.

More background information and introduction can be found here:

http://www.bacnet.org

http://www.bacnet.org/Bibliography

AC500 and BACnet

A BACnet device can be described by its *“BACnet Interoperability Building Blocks”* (BIBB)s, which are needed to establish services. They are grouped in different areas:

- *"Data Sharing"* (DS)
- *"Alarm and Event Management"*(AE)
- *"Scheduling"* (SCHED)
- *"Trending"* (T)
- *"Device and Network Management"* (DM)

"Data Sharing" for example contains two BIBBs which are needed for the *"Service Read Property"*:

- Client side: DS-RP-A (Data Sharing - Read Property - A)
- Server side: DS-RP-B (Data Sharing - Read Property - B)

The BACnet standard defines profiles by the minimum required BIBBs, see table below. *"BACnet Simple Sensor"* (B-SS) is the simplest one, only containing one BIBB. More complex devices contain more BIBBs (from right to left).

		CP600	AC500 V3		AC500 V2		
Interoperability Areas (IA)	**BACnet Device-profiles**						
	B-OWS	**B-OD**	**B-BC**	**B-AAC**	**B-ASC**	**B-SA**	**B-SS**
Data Sharing	DS-RP-A, B	DS-RP-A, B	DS-RP-A, B	DS-RP-B	DS-RP-B	DS-RP-B	DS-RP-B
	DS-RPM-A	DS-RPM-A, B	DS-RPM-A, B	DS-RPM-B			
	DS-WP-A	DS-WP-A, B	DS-WP-A, B	DS-WP-B	DS-WP-B	DS-WP-B	
	DS-WPM-A		DS-WPM-B	DS-WPM-B			
		DS-V-A					
		DS-M-A					
Alarm & Event Management	AE-N-A	AE-N-A	AE-N-I-B	AE-N-I-B			
	AE-ACK-A		AE-ACK-B	AE-ACK-B			
	AE-INFO-B		AE-INFO-B	AE-INFO-B			
	AE-ESUM-A		AE-ESUM-B				
		AE-VN-A					
Scheduling	SCHED-A		SCHED-E-B	SCHED-E-B			
Trending	T-VMT-A		T-VMT-I-B				
	T-ATR-A		T-ATR-B				
Device & Network Management	DM-DDB-A, B	DM-DDB-A, B	DM-DDB-A, B	DM-DDB-B	DM-DDB-B		
	DM-DOB-A, B	DM-DOB-B	DM-DOB-B	DM-DOB-B	DM-DOB-B		
	DM-DCC-A		DM-DCC-B	DM-DCC-B	DM-DCC-B		
	DM-TS-A		DM-TS-B or DM-UTC-B	DM-TS-B or DM-UTC-B			
	DM-UTC-A						
	DM-RD-A		DM-RD-B	DM-RD-B			
	DM-BR-A		DM-BR-B				

The AC500 V2 supports BIBBs qualifying it as *"BACnet Application Specific Controller"* (B-ASC), by installing the BACnet B-ASC library.

AC500 V3 supports many more BIBBs qualifying it as *"BACnet Building Controller"* (B-BC), which contains a server (all BIBBs ending with -B) and a client (all BIBBs ending with -A). In fact, the AC500 contains some more BIBBs. All BIBBs under B-BC in the table above, plus:

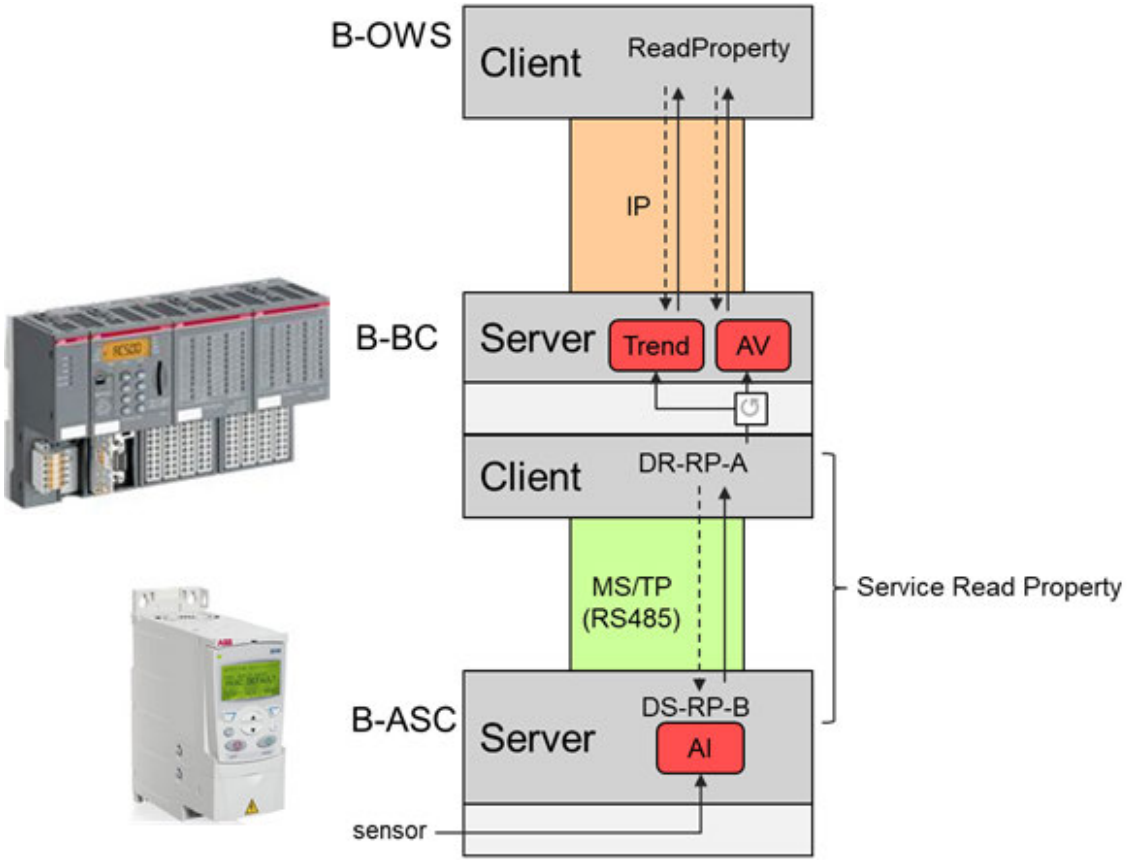
DS-COV-A, -B	(Change of Value-A, -B)
DS-COVP-A, -B	(Change of Value of Properties-A, -B)
AE-N-E-B	(Alarm and Event-Notification External-B)
AE-ASUM-B	(Alarm and Event-Alarm Summary-B)
SCHED-I-B	(Scheduling-Internal-B)
T-VMT-E-B	(Viewing and Modifying Trends External-B)

DM-TS-B	(Time Synchronization-B)
DM-UTC-B	(UTC Time Synchronization-B)
DM-MTS-A	(Manual Time Synchronization-A)
DM-LM-B	(List Manipulation-B)
DM-OCD-B	(Object Creation and Deletion-B)
NM-BBMDC-B	(BBMD Configuration-B)

...

A list with all details can be found in the Automation Builder pdf document ABB-B-BC-PICS-AC500_V3.pdf. Direction: Help/Project examples/Examples.

The figure below shows a typical application for an AC500 V3, acting as B-BC.



A drive with several actuators and sensors is acting as B-ASC, for example providing a temperature value as *“Analog Input”* (AI) object on the MS/TP network.

Chapter 1.6.6.3.9.3.1 “Supported BACnet networks ” on page 3931

AC500 B-BC as client can read this temperature value, perform some processing (scaling, limit check) and on the server side provide the processed value as *“Analog Value”* (AV) object and as *“Trend”* object on the IP network. Higher level clients like BACnet Operator Workstation (B-OWS) can access the processed objects *“Analog Value”* and *“Trend”* for supervision.

The following chapters describe the possible applications and how to configure an AC500 V3 as B-BC.

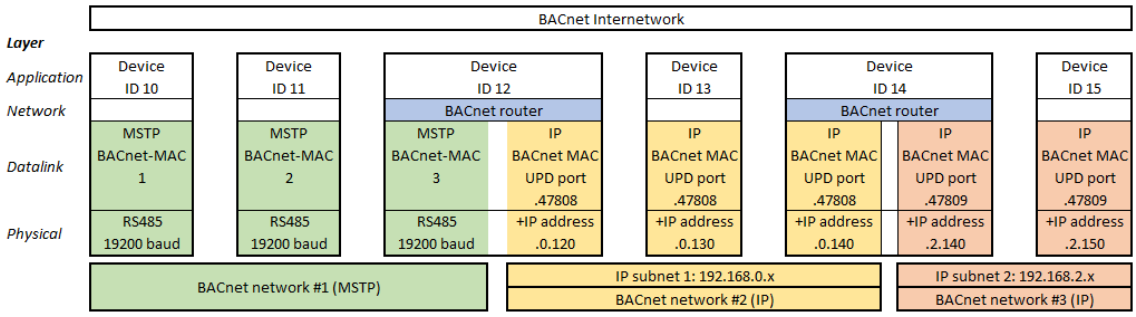
AC500 V3 as BACnet Building Controller (B-BC)

The BACnet integration into CODESYS implements the ANSI/ASHRAE standard 135-2012 (ISO 16484-5) protocol revision 14 and is based on the AMEV AS-A and AS-B standards. Integration allows access to the properties of BACnet objects and the configuration parameters of a BACnet device by means of an IEC application. You can program a dynamic BACnet configuration and have access to the BACnet functions in the BACnet network by reading and writing BACnet object properties.

Supported BACnet networks

BACnet can run on different local area network types. The AC500 B-BC supports the following ones:

- MS/TP (Master Slave / Token Passing), based on serial RS-485
- BACnet IP, based on Ethernet / UDP / IP



Different networks can be combined to one common *"BACnet internetwork"*. The figure above shows an example of some BACnet devices in one *"BACnet internetwork"*. Each device has a device ID (10 to 15) which must be unique on application level. Services on application level (e.g. read or write request) are working with these device IDs and need no addressing information of the lower levels.

The example *"BACnet internetwork"* consists of different BACnet networks:

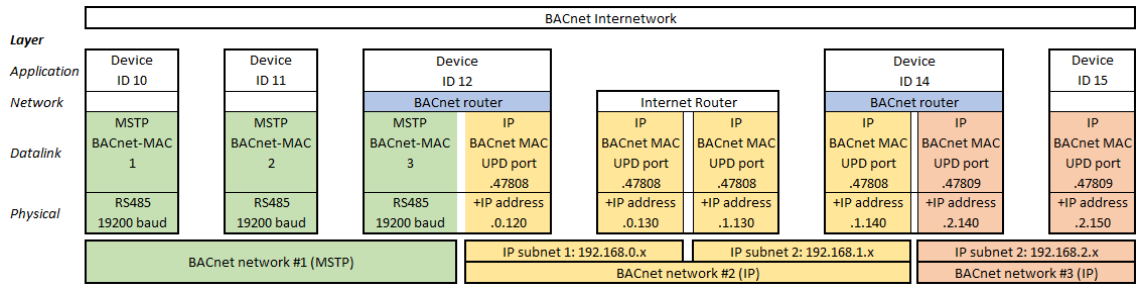
- BACnet MS/TP network connecting device 10, 11 and 12
- BACnet IP network (UDP port 47808), consisting of one IP subnets with IP range 192.168.0.x, connecting device 12, 13 and 14
- BACnet IP network (UDP port 47809), consisting of one IP subnet with IP range 192.168.2.x, connecting device 14 and 15

Addressing in a BACnet network is done through datalinks which must have a unique BACnet MAC address (which is different to an Ethernet MAC address).

- In a MS/TP network the BACnet MAC address is just one octet (1, 2, 3 in the example).
  ↳ *Chapter 1.6.6.3.9.3.4.4 "Configuration of datalinks " on page 3939*
- In an IP network the BACnet MAC address is the combination of the IP address and the UDP port number (for example 192.168.0.130.47808 for device 13). The following 16 UDP ports are reserved for BACnet: BAC0 (=47808 decimal) to BACF.
  ↳ *Chapter 1.6.6.3.9.3.4.4 "Configuration of datalinks " on page 3939*

To form a common *"BACnet internetwork"* the single BACnet networks must be combined by BACnet routers. AC500 can act as a BACnet router between BACnet MS/TP and IP networks (device 12 in the figure above) or between two different BACnet IP networks (device 14).

Two IP subnets using the same UDP ports can be combined to one BACnet IP network with an internet router.



The problem is that internet routers block local broadcast messages, which are required for BACnet communication. This can be solved by *"Broadcast Management Devices"* (BBDM). AC500 V3 can be configured as BBDM. In the figure above the devices 12 and 14 should be configured as BBDM in order to enable the BACnet communication across the internet router.

An alternative is to configure AC500 V3 as foreign BACnet device if an IP subnet contains no BBDM device to pass broadcast messages over internet routers.

Configuring the AC500 as BBDM or foreign device is described in *Chapter 1.6.6.3.9.3.4.4 "Configuration of datalinks " on page 3939*.

Supported objects and properties

Communication with BACnet is done through objects and properties.

The AC500 B-BC server of the figure below is represented as a BACnet device object with *"ID 12"*. The device contains more objects like the *Analog Input* object, representing the input of a temperature measurement device. An object contains several properties, like *"ID, Description, Present Value, Unit"* etc.

Further possible objects of an AC500 B-BC are:

- *"Binary Input"* for example from connected to a switch
- *"Analog / Binary Output"* for actuators
- *"Analog / Binary Values"* for local variables
- *"Calender"*
- *"Schedule"*
- *"Trend Log"*
- ...
- A list with all details can be found in the Automation Builder pdf document ABB-B-BC-PICS-AC500_V3.pdf. Help/Project examples/.

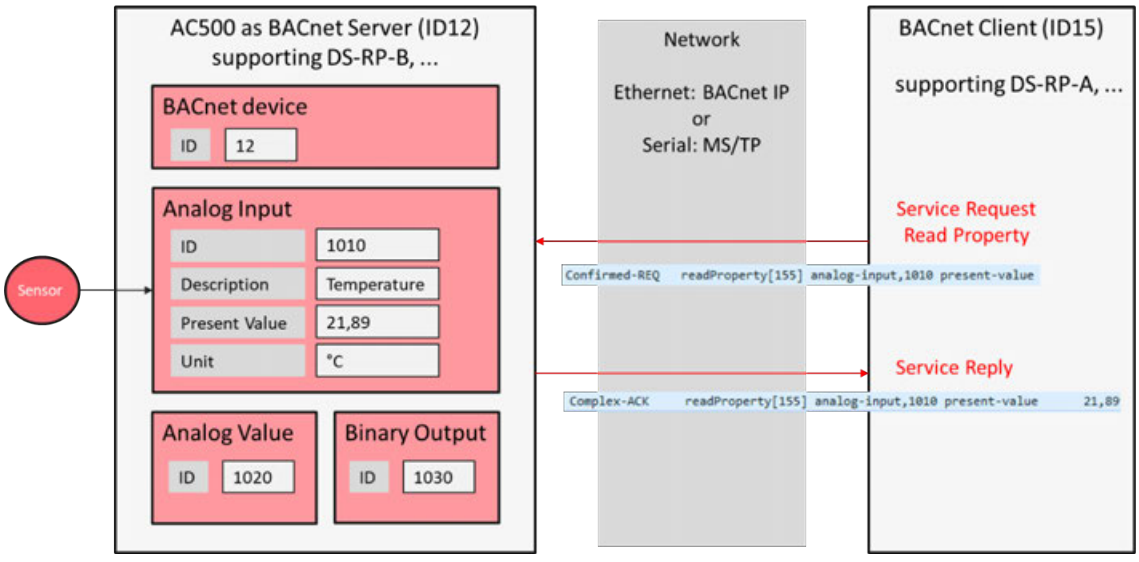


Fig. 341: BACnet objects, properties, services and BIBBs

Supported BIBBs and services

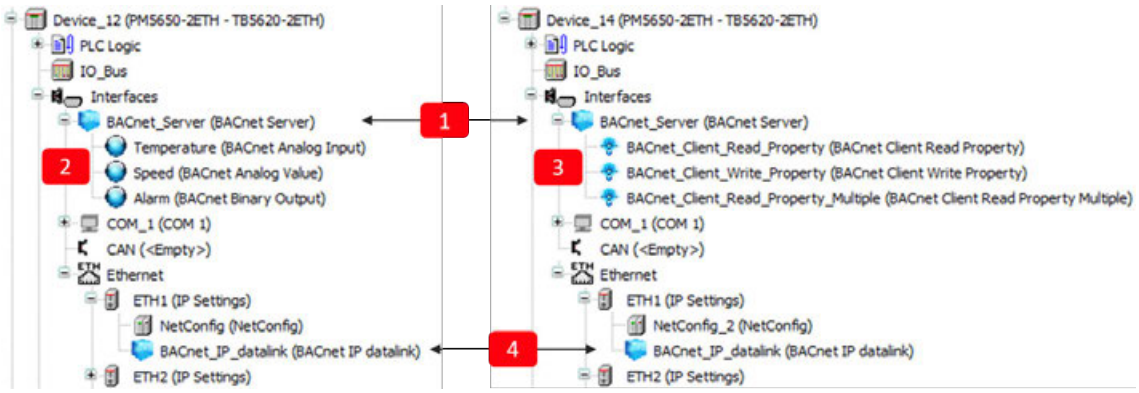
While objects and properties describe which data are communicated, the communication itself is done with services between clients and servers. A certain service can only be executed if client and server have the related BIBBs. The Fig. 341 *BACnet objects, properties, services and BIBBs* shows a simple *"Service Read Property"* which is possible because the client on the right supports DS-RP-A and the server on the left supports DS-RP-B. The service is executed in two steps:

1. The client initiates a confirmed request *"Read Property"*, asking for the present value of the *"Analog Input"* of object with *"ID 1010"*.
2. The server answers with an acknowledge, sending the present value which is 21,89°C in the example.

A list of all supported BIBBs and services of AC500 V3 is given in the Automation Builder pdf document ABB-B-BC-PICS-AC500_V3.pdf. Help/Project examples/Examples.

BACnet configuration in Automation Builder

To act as a BACnet server or client, the AC500 must be configured accordingly. The figure below shows the basic configuration of a BACnet server (left) and a BACnet server with client functionality (right). It is also possible to have server and client functionality in parallel.



Following objects need to be created:

1 *"BACnet Server"* root object. This is the root object for the server functionality, as well as for the client functionality. It is mandatory, even if only client functionality is required. ↳ *Chapter 1.6.6.3.9.3.4.1 "Configuration of BACnet server root object " on page 3934*

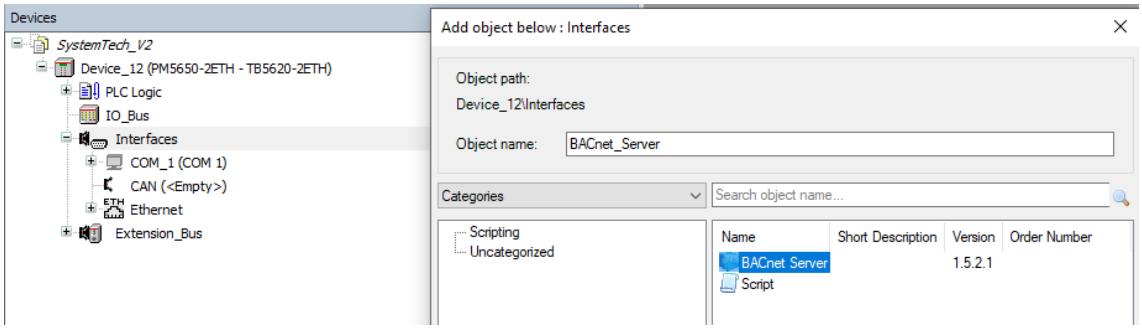
2 BACnet server objects, for example *"BACnet Analog Input"* `Temperature`. The properties of the objects must be controlled (written or read) by the PLC logic. ↳ *Chapter 1.6.6.3.9.3.4.2 "Adding BACnet server objects" on page 3935*

3 BACnet client objects, represented by a different symbol. For example, *"BACnet Client Read Property"*. The functionality of the client objects must be programmed in the PLC logic. Inserting the client objects below the server is optional. It is also possible to instantiate the objects only in a PLC logic. ↳ *Chapter 1.6.6.3.9.3.4.3 "Adding BACnet client functionality" on page 3936*

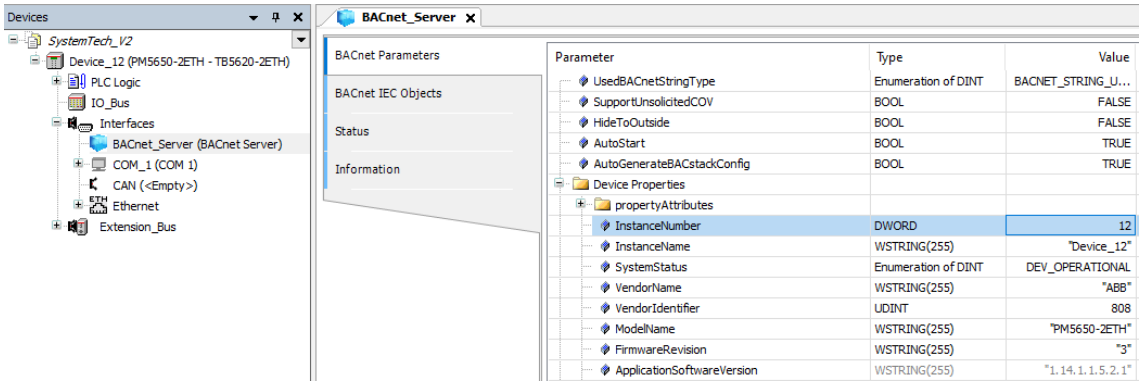
4 Datalink for the physical layer. This object links the physical interface (Ethernet IP or serial MS/TP) to the *"BACnet Server"* object. In the example above the IP address of `ETH1` is automatically retrieved by inserting the *"BACnet IP datalink"* below the `ETH1` port. ↳ *"Configuration of an IP datalink" on page 3940*. For MS/TP refer to ↳ *"Configuration of an MS/TP datalink" on page 3939*.

Configuration of BACnet server root object

1. Create an empty project with an AC500 V3 CPU type and call it fpr example *“Device_12”*.
2. Insert a *“BACnet Server”* object below the interfaces object in the device tree.

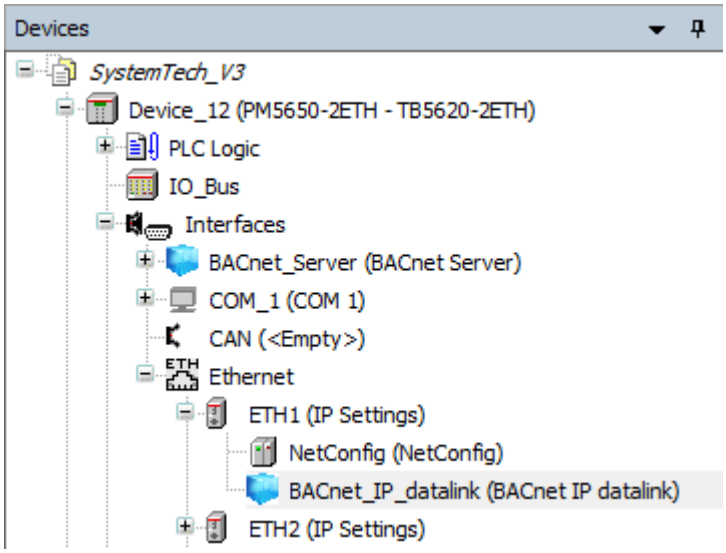


3. Set the device `InstanceNumber` in the *“BACnet Parameters”* of the *“BACnet Server”*, e.g. to `12` and the `InstanceName` to `Device_12` (according to Fig. 341 *BACnet objects, properties, services and BIBBs*).



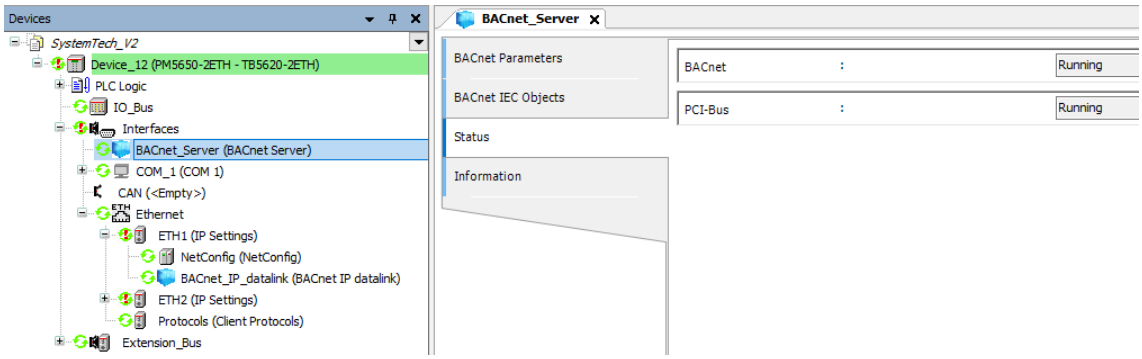
4. Add a datalink, IP or MS/TP. In the example an IP datalink is inserted below `ETH1`. Default parameters are sufficient if only one datalink is used.

↳ *“Configuration of an IP datalink” on page 3940*

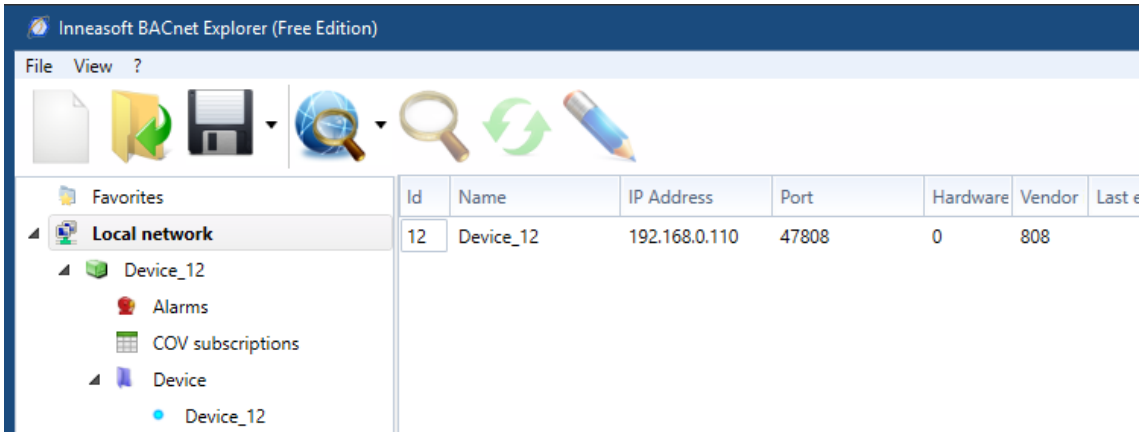


5. Build the project, download to the PLC and set it to *[RUN]*. The status of the *"BACnet Server"* should be green (running). If not, please ensure that you have installed the runtime license BACnet Protocol B-BC Runtime, verifiable by right-click on the PLC node and select *[Show license information]* from the runtime licensing menu. The project is scanned for required licenses. If you are logged in to a PLC, then the licenses available on the PLC are displayed. A missing required license is highlighted.

↳ *Chapter 1.6.6.2.2.2 "PLC runtime licensing" on page 3665*



6. Start any BACnet client to find the server, for example Inneasoft BACnet Explorer.

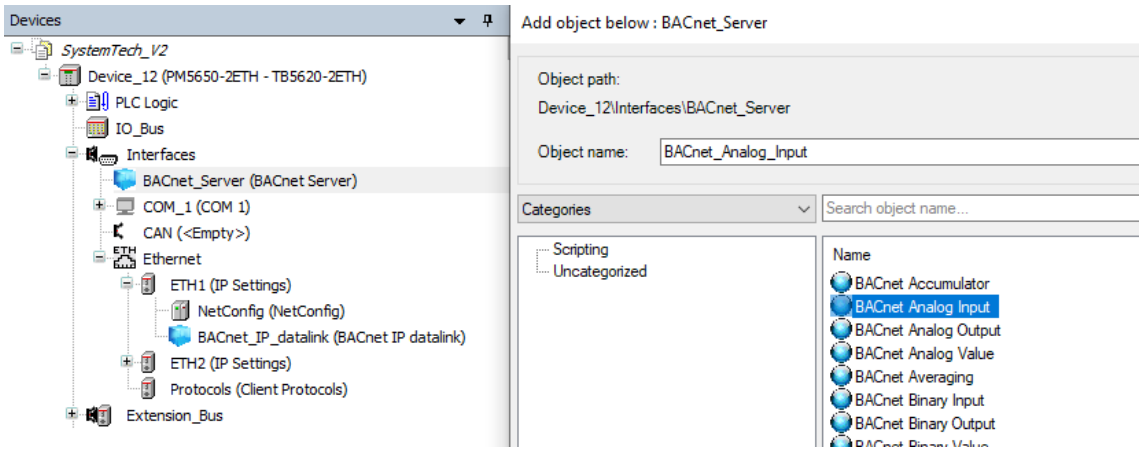


Adding BACnet server objects

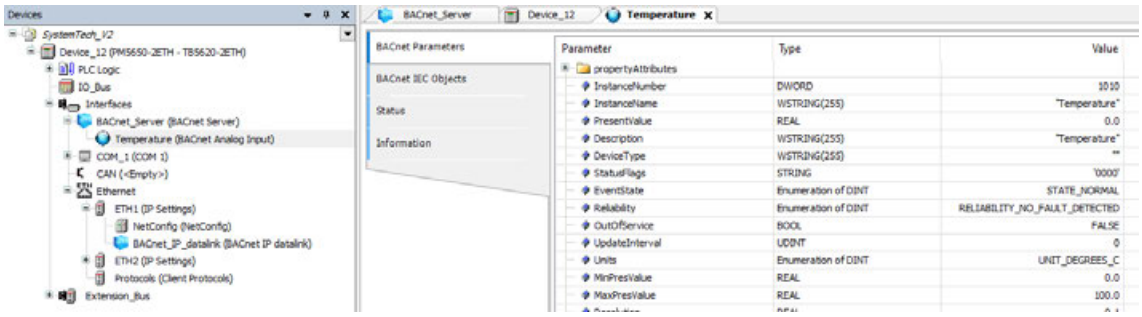
Goal is to publish an analog value as BACnet server object. This example is according to Fig. 341 *BACnet objects, properties, services and BIBBs*, left part containing a temperature value.

1. Configure a *"BACnet Server"* root object according to ↳ *Chapter 1.6.6.3.9.3.4.1 "Configuration of BACnet server root object " on page 3934*.

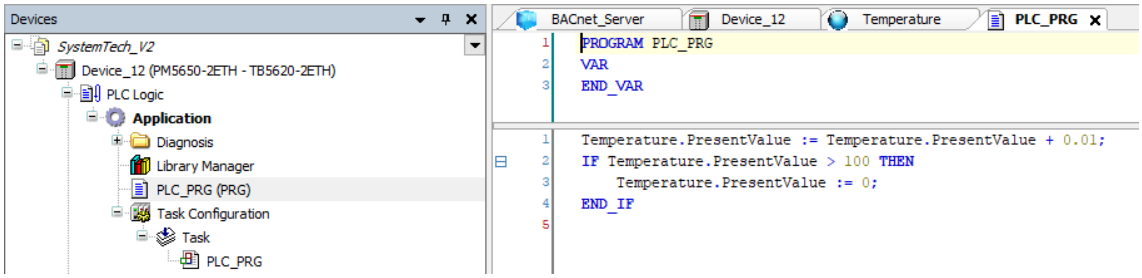
2. Add a *"BACnet Analog Input"* object below the *"BACnet Server"*.



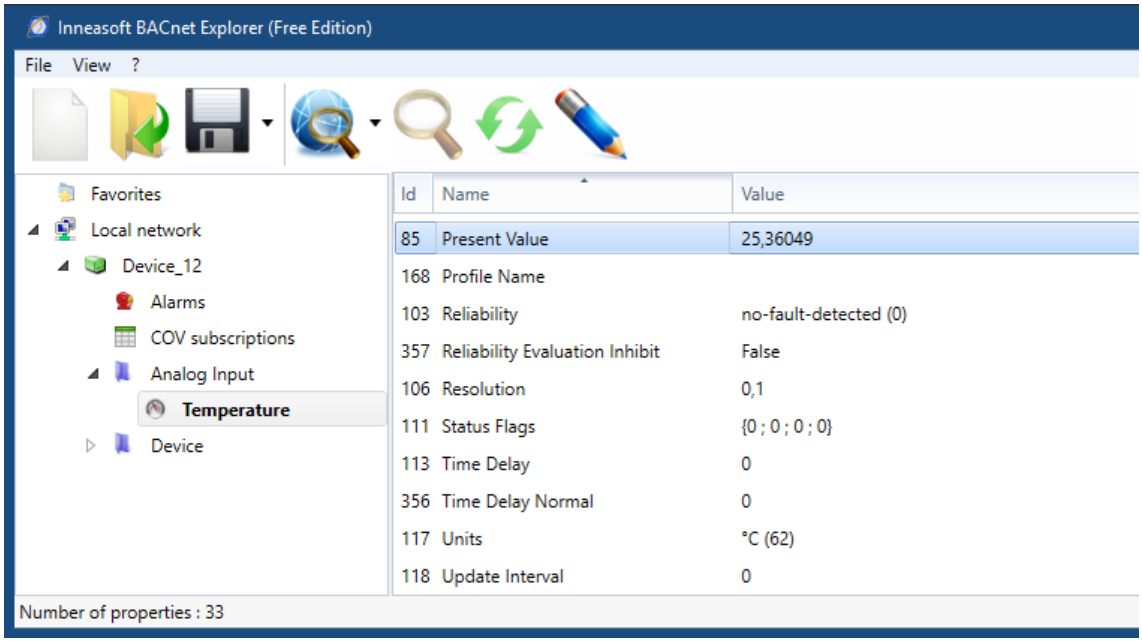
3. Rename it to `Temperature`, adjust the parameters: `InstanceNumber`: `1010`, `Description`: `Temperature`, `Units`: `UNIT_DEGREES_C`.



4. The present value of the objects `Temperature` needs to be fed with the value from the real temperature device. Alternatively, a simple PLC program can simulate this value.



5. Download the program and observe the temperature value in the BACnet client.

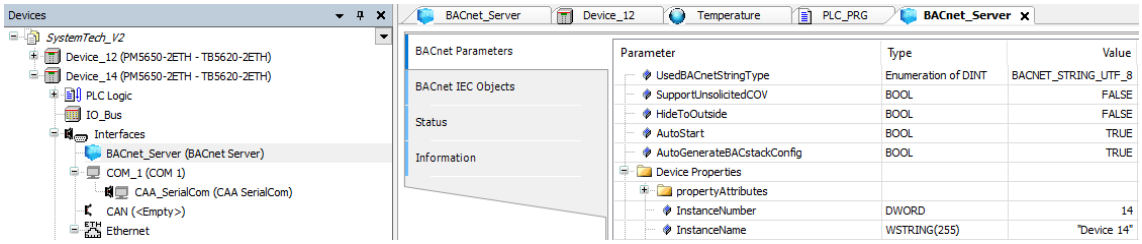


Adding BACnet client functionality

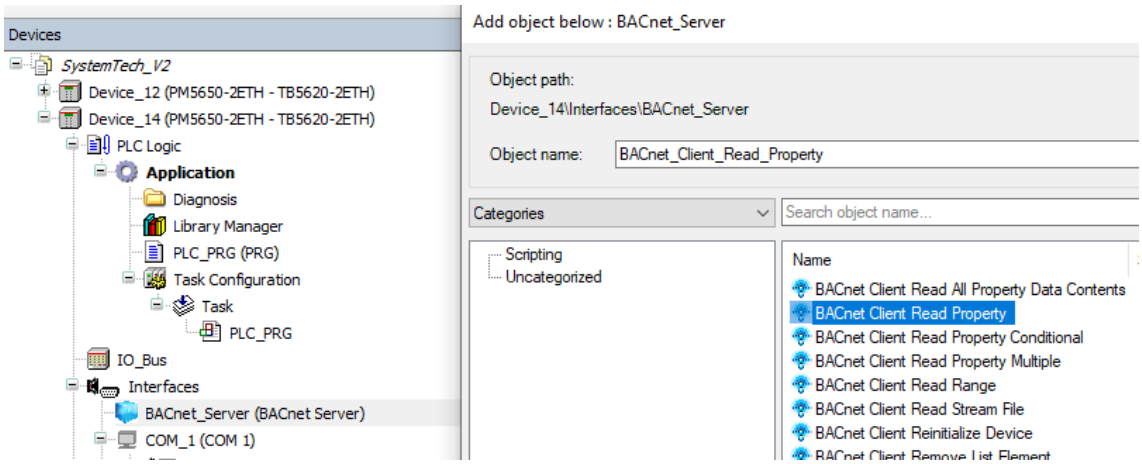
Goal is to configure a second AC500 controller as BACnet client which reads an analog value from a server. This example is according to Fig. 341 *BACnet objects, properties, services and BIBBs*, right part.

1. Add a new controller and configure a *"BACnet Server"* root object according to ↳ *Chapter 1.6.6.3.9.3.4.1 "Configuration of BACnet server root object " on page 3934*.

2. Set `InstanceNumber` to `14` and `InstanceName` to `Device 14`.



3. In addition to BACnet objects, BACnet clients can also be inserted as devices under a *"BACnet Server"*. Add a *"BACnet Client Read Property"* below the *"BACnet Server"* node.

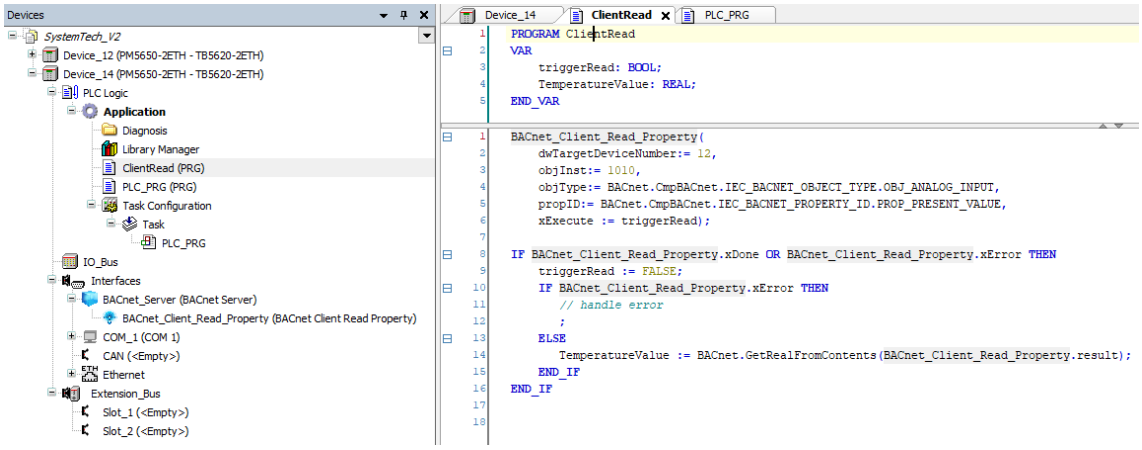


4. The created object *“BACnet Client Read Property”* generates a function block instance which can be used to program the client read functionality. The figure below shows a simple example.

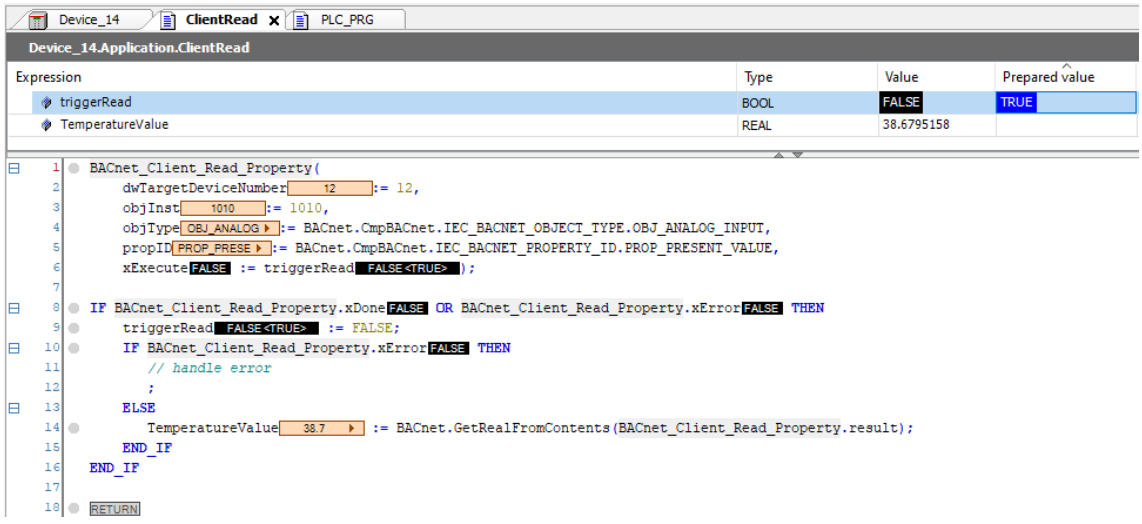
   In line 1-5 of the code part the function block is called with the following parameter:

   - Device ID of the server to read from (`12`) ↳ *Chapter 1.6.6.3.9.3.2 “Supported objects and properties ” on page 3932*
   - Object ID of the object to read from (`1010` for the *“Analog Input”*)
   - Object type (*“Analog Input”*)
   - Property to read (*“present value”*)
   - `triggerRead` to start the read operation

   When the user (or another program part) sets the variable `triggerRead` from `FALSE` to `TRUE` the edge triggered function block `BACnet_Client_Read_Property` starts operation and sends the read request to the server device. After receiving the reply from the Server, the output `.xDone` gets `TRUE` (line 8) and the temperature value can be read from the output `.result` (line 14).

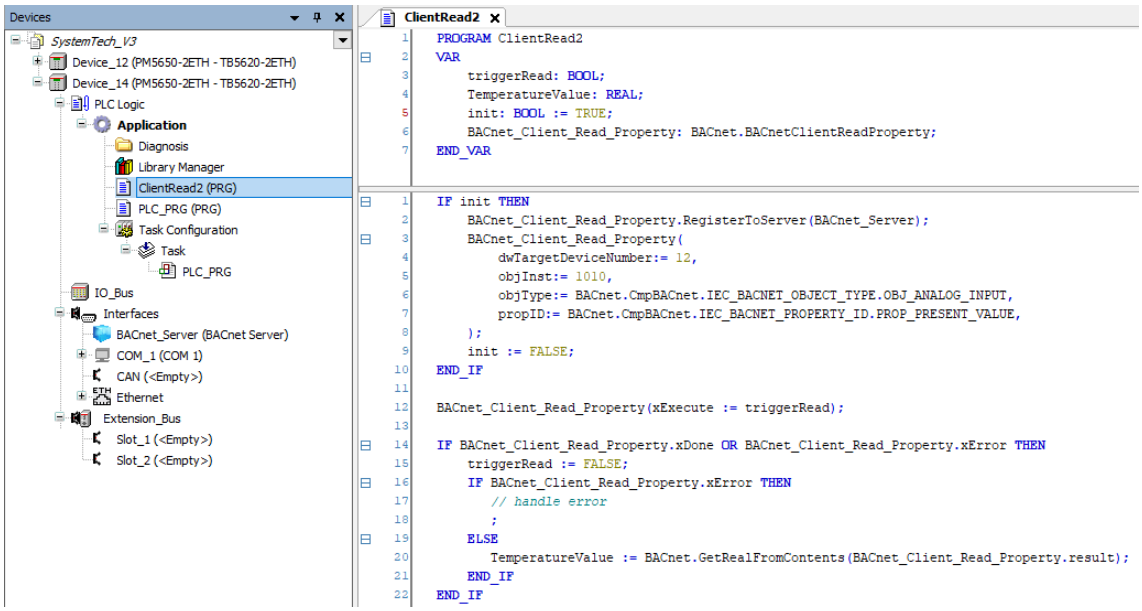


5. Download this program to another AC500 V3 controller, which is in the same IP network as the server. Set it to run and read the temperature value by setting `triggerRead` to `TRUE`. In online mode the read temperature value can be observed in line 14.

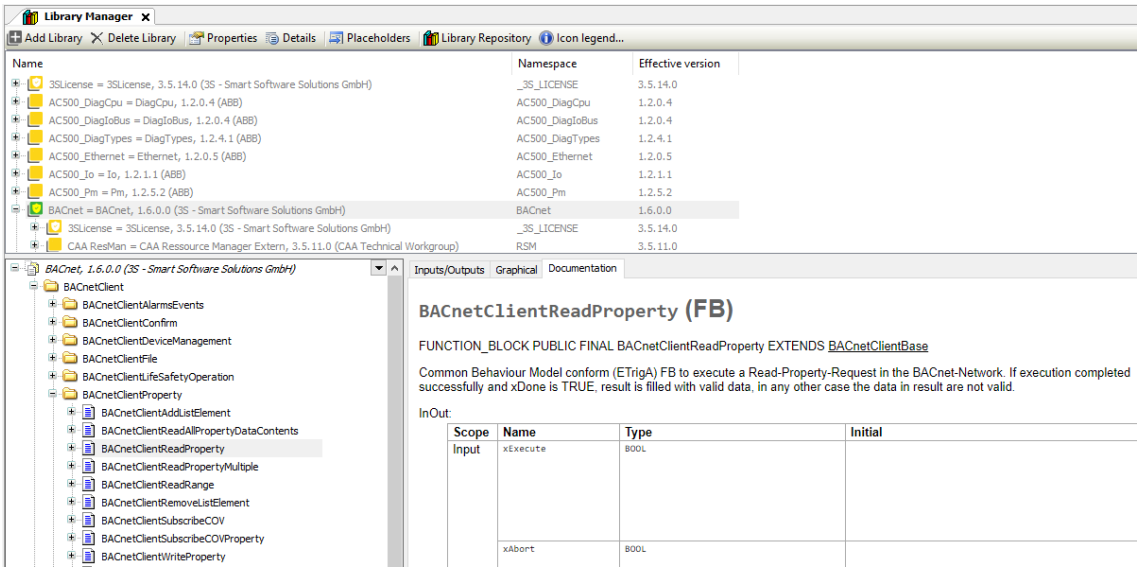


Alternative configuration

Unlike BACnet objects, a BACnet client does not require a complex (static) configuration, thus a client function block can be used without creating a BACnet client as device.



There is no `BACnet_Client_Read_Property` object created below the *"BACnet Server"*. Instead a function block `BACnet_Client_Read_Property` must be declared in the PRG (line 6 in the declaration) and initially "connected" to its *"BACnet Server"* in IEC-code via `RegisterToServer()`, and thus get activated (line 2 in the code) ↳ *Chapter 1.10 "Reference, function blocks" on page 4292*.



Configuration of datalinks

For communication with other BACnet devices AC500 provides two different possibilities: MS/TP and IP.

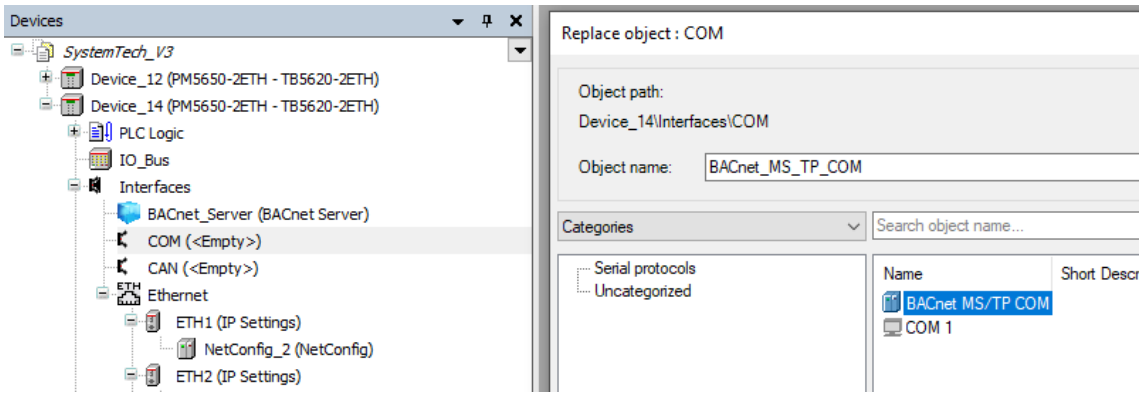
↳ *Chapter 1.6.6.3.9.3.1 "Supported BACnet networks " on page 3931*

For a non-routing device one MS/TP or IP datalink must be configured.

If more than one datalink is configured, routing between the datalinks is automatically enabled.

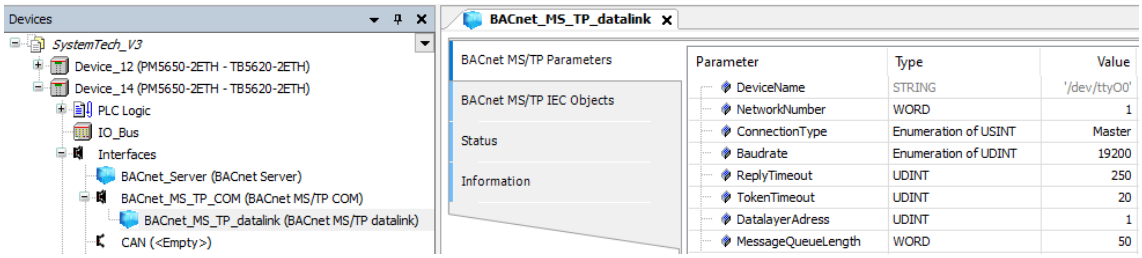
Configuration of an MS/TP datalink

- Add the *"BACnet MS/TP COM"* object below the COM port.



In fact the empty COM port is replaced by the *"BACnet MS/TP COM"*. By that the COM port is configured as RS-485 with fixed settings for MS/TP: No parity, 8 data bits, 1 stop bits.

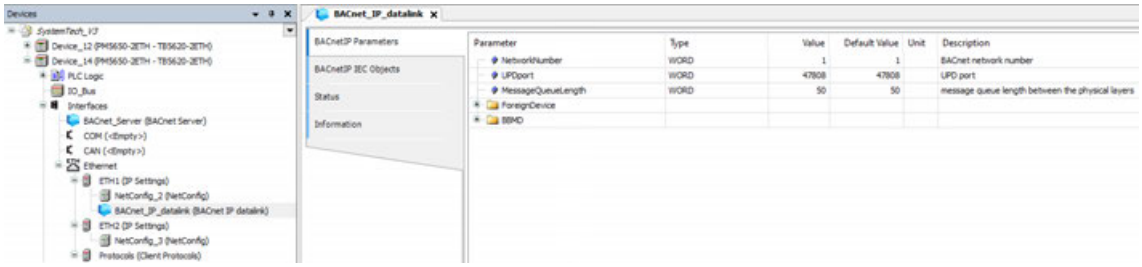
- Below the *"BACnet MS/TP COM"* port object an *"BACnet MS/TP datalink"* is inserted automatically which can be configured according to the requirements.



- `NetworkNumber`: Use the default value `1` if no routing is required. For routing, use a unique network number in one controller.
- `ConnectionType`: Use the default value `Master` if no routing is required. For routing, use *"Master – answering always postponed"*.
- `Baudrate` can be set according to requirements in the range of from 9600 to 38400 bits/s, higher values (57600 and 115200 bits/s) are not recommended.
- `DatalayerAddress`: This is the MAC address as described in *Chapter 1.6.6.3.9.3.1 "Supported BACnet networks " on page 3931*. The MAC address must be unique in the MS/TP network.
- For all other parameters the default values are recommended for typical applications.

Configuration of an IP datalink

- Add a *"BACnet_IP_datalink"* object below the Ethernet port `ETH1` or `ETH2`.



- `NetworkNumber`: Use the default value if no routing is required. For routing, use a unique network number in one controller.
- `UPDport`: Use the default value (`47808` decimal) in the normal case. Range is possible from BAC0 (= 47808 decimal) to BACF. UDPport + IP address form the MAC address of the IP datalink as described in *Chapter 1.6.6.3.9.3.1 “Supported BACnet networks ” on page 3931*. The IP address cannot be specified here. It is automatically taken from the parent Ethernet node (`ETH1` or `ETH2`); its IP address is set in the communication settings of the CPU node, *“Device_14”* in the example.
- `ForeignDevice` and `BBMD`: Special configuration is only needed if an internet router is located between two BACnet devices.
  Chapter 1.6.6.3.9.3.1 “Supported BACnet networks ” on page 3931
  AC500 can be configured as `ForeignDevice` or `BBMD`, but not the combination of both. An example for BBDM can be found in the example folder.

Configuration of Routing

Routing enables the combination of different BACnet networks to one common *“BACnet internetwork”*.

Chapter 1.6.6.3.9.3.1 “Supported BACnet networks ” on page 3931

BACnet devices from different BACnet networks can communicate with each other.

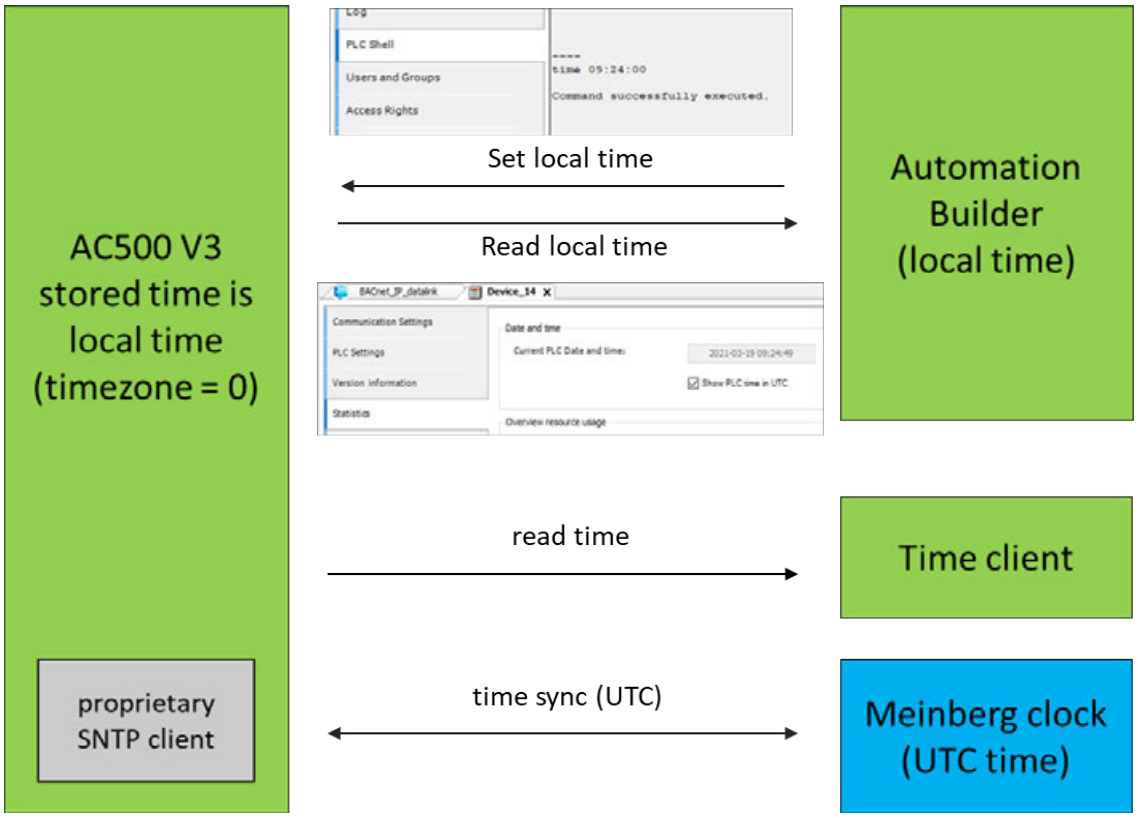
If more than one datalink is configured in one CPU, routing between the different networks is automatically enabled. It must only be ensured that the network number is unique in one controller.

Chapter 1.6.6.3.9.3.1 “Supported BACnet networks ” on page 3931

For MS/TP the `ConnectionType` must be set to *“Master – answering always postponed”*. An example for routing can be found in the example folder.

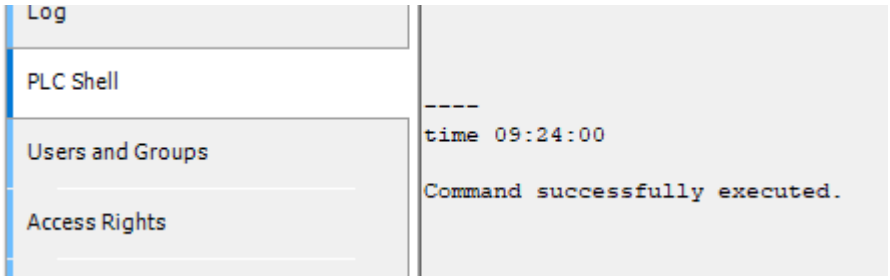
Time syncronisation

The BACnet clients expect to receive the local time. Currently the AC500 V3 does not distinguish between UTC time and local time and its time zone is set to 0. This will be improved in the near future. In the meantime, it is recommended to store the local time (green color in the following figure) in the AC500 as a workaround.

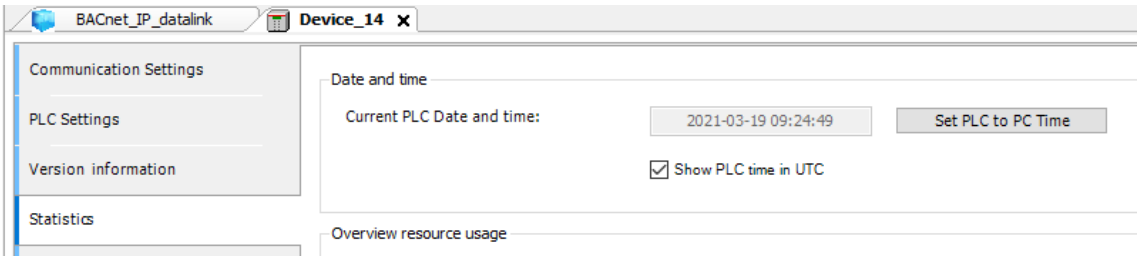


Using this workaround, the following time sync mechanisms can be used:

- Set local time from Automation Builder Tab *"PLC Shell"*:
  Set the time by the command "time hh:mm:ss"



- Read the local time from the Automation Builder Tab *"Statistics"*:
  "Current PLC Date and time" shows the PLC time as local time without conversion, if the tab *"Show PLC time in UTC"* is enabled.



For storing the local time in AC500, do not use the button [Set PLC to PC Time] (Tab "Statistics"), since this is always converting from local time to UTC time.

- BACnet clients can read local or UTC time, both requests will deliver the same (local) time information, since the timezone is 0.
- If an SNTP time sync is required (for example with a Meinberg clock), UTC times are exchanged. For conversion of UTC to local time in AC500 a proprietary STNP client must be programmed.
  Please contact the PLC support for more information.

Package content

The BACnet package PS5607-BACnet-BC can be installed with the Installation Manager and contains the following components:

- BACnet runtime component, part of AC500 firmware.
- Automation Builder package: CODESYS BACnet
  - BACnet plug-in component
  - Device descriptions for *"BACnet Server"*, BACnet objects, BACnet client and datalinks
  - Libraries: `BACnet`, `BACnetDefaultImpl` and `CmpBACnet`.
    Chapter 1.6.6.3.9.3.5.1 "BACnet libraries" on page 3943

Example folder

- Example folder
  - Examples and example documentation
    Chapter 1.6.6.3.9.3.5.2 "Application examples" on page 3944
  - Datasheet and FAQ
    BACnet Protocol Implementation Conformance Statement (PICS), acting as a datasheet, describing all BACnet objects, services and communication capabilities.
    BACnet Conformance Certificate
    FAQ – Frequently Asked Questions, including AC500 specific information, performance and limit

BACnet libraries

The IEC library `CmpBACnet` represents the integration of the BACnet stack into a CODESYS IEC environment and provides the BACnet data types as well as the `BACstack` methods. The sole use of the IEC library `CmpBACnet` (without the `BACnet` and `BACnetDefaultImpl` libraries) would result in complex and lengthy IEC application code.

The `BACnet` library simplifies BACnet application development considerably as compared to the sole use of `CmpBACnet`, especially in the following areas:

- Starting and stopping the BACnet stack
- Using BACnet server objects and their properties
- Triggering asynchronous requests (mainly client service requests) and processing the request transaction
- Processing of callbacks from the BACnet stack (see `IBACnetEventConsumer`) and distributing the callbacks to multiple receivers in the application

Furthermore, the `BACnet` library provides a plug-in mechanism (`BACnetServerPlugin`) for extending certain aspects of the `BACnet` library. `BACnetServerPlugin` is the basis for the `BACnetDefaultImpl` library.

The `BACnetDefaultImpl` library is used for the additional simplification of BACnet application development. The BACnet standard ASHRAE 135 leaves some aspects of the practical use of BACnet open. The most notable examples include the following:

- Persistence of server objects
- Storage and persistence of `Trend Log`, `Trend Log Multiple`, and `Event Log` entries
- Update of the date/time information of the device object

The IEC library `BACnet` is intended as a layer over the IEC library `CmpBACnet`. However, the layer does not hide the library because this would require the `BACnet` library to have "facade" functions for `CmpBACnet` functions. These facade functions would result in larger application code and increased runtime requirements. This is difficult for the PLC to accept. For this reason, it is necessary to know when elements from the `BACnet` library or `CmpBACnet` library are to be used.

General rules:

- Starting and stopping the BACnet stack
  Always use `BACnetServer.StartBACnetStack` and `BACnetServer.StopBACnetStack` or `AutoStart`. Never directly use the corresponding functions of the `CmpBACnet` library, such as `CmpBACnet.BACnetServerInit`.
- Using BACnet server objects and their properties
  Always use the specified function blocks in IEC-lib-BACnet, such as `BACnetAnalogValue`. Never directly use the corresponding functions of the `BACnet` library, such as `CmpBACnet.BACnetStorePropertyInstance`.
- Triggering of asynchronous requests
  Always use the specified client function blocks of the `BACnet` library, such as `BACnetClientReadProperty`. Never directly use the corresponding functions of the `CmpBACnet` library, such as `CmpBACnet.BACnetReadProperty`. All functions of the `CmpBACnet` library that require a `BACnetAsyncTransactionToken` belong to this category and should never be used directly.
- Processing of callbacks from the BACnet stack and distributing the callbacks to multiple receivers in the application
  Always use `IBACnetEventConsumer` and `BACnetServer.RegisterHook`/`UnregisterHook/RegisterCallback`/`UnregisterCallback`. Never directly use the corresponding functions of the `CmpBACnet` library, such as `CmpBACnet.BACnetSetHook` or `CmpBACnet.BACnetSetCallback`.

When is it appropriate and safe to directly call the functions of the `CmpBACnet` library?

Basically, it is only necessary to call functions of `CmpBACnet` directly when a corresponding functionality is not provided in the `BACnet` library. Check the `BACnet` library first before trying to use `CmpBACnet` directly. It is possible to use blocking functions in `CmpBACnet`, such as `BACnet*CbCompletion`, `BACnetIam(Ex)`, or `BACnetIHave(Ex)`, `BACnetUnconf*`.

Most often, you will use `BACnet*CbCompletion` to implement your specific `IBACnetEventConsumer.BACnetEventCallback`s. But first check whether or not the `BACnetDefaultImpl` library already contains an appropriate standard implementation.

Application examples

- `AC500_V3_BACnet_B-BC_Example_ABxxx.project` including simple read and write operations between client and server.
  - Use case 1: AC500 as BACnet client, read and write (with priority)
  - Use case 2: AC500 as *“BACnet Server”*, publish the analog value
- `AC500_V3_BACnet_B-BC_Example_Routing_ABxxx.project`
- Examples from 3S, including
  - Read and write operations with more options, notification class, calendar, scheduler, etc.
  - Device discovery
  - BBMD
  - Persistence
  - Logging
  - Routing

1.6.6.3.10 OPC UA

AC500 V3 controllers support the OPC UA protocol - a machine to machine communication protocol for industrial automation. Further information on OPC UA:

- How to connect robot controllers to OPC UA: *https://new.abb.com/products/robotics/home/irc5/irc5-options/opc-ua*
- Installation and configuration of an OPC UA server: *Chapter 1.6.6.5.2 “OPC UA server for AC500 V3 products” on page 3981*
- Configuration and handling of OPC UA in Automation Builder: *https://library.e.abb.com/public/1d1cbdc36f2d417cb455c946835d12ea/Application%20Note%203ADR*

1.6.6.4 Data transfer and programming

1.6.6.4.1 Source download/upload

Source download

Prerequisites

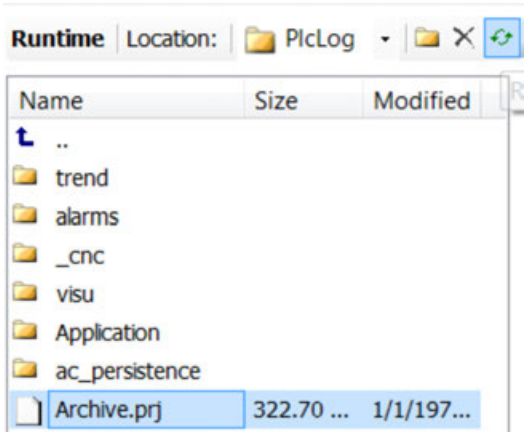
- Communication settings are correct
- Project is saved on PC
- PLC is connected

1. Click *“Online* ➔ *Source download to connected device”*.

   ⇨ Project archive will be downloaded to PLC.

2. To verify download double-click node *“PLC_AC500_V3”*, select view *“Files”* and double-click folder *“PlcLogic”* of the *Runtime* view (if necessary click refresh button of *Runtime* view).

   ⇨ File *Archive.prj* will appear if download was successful.

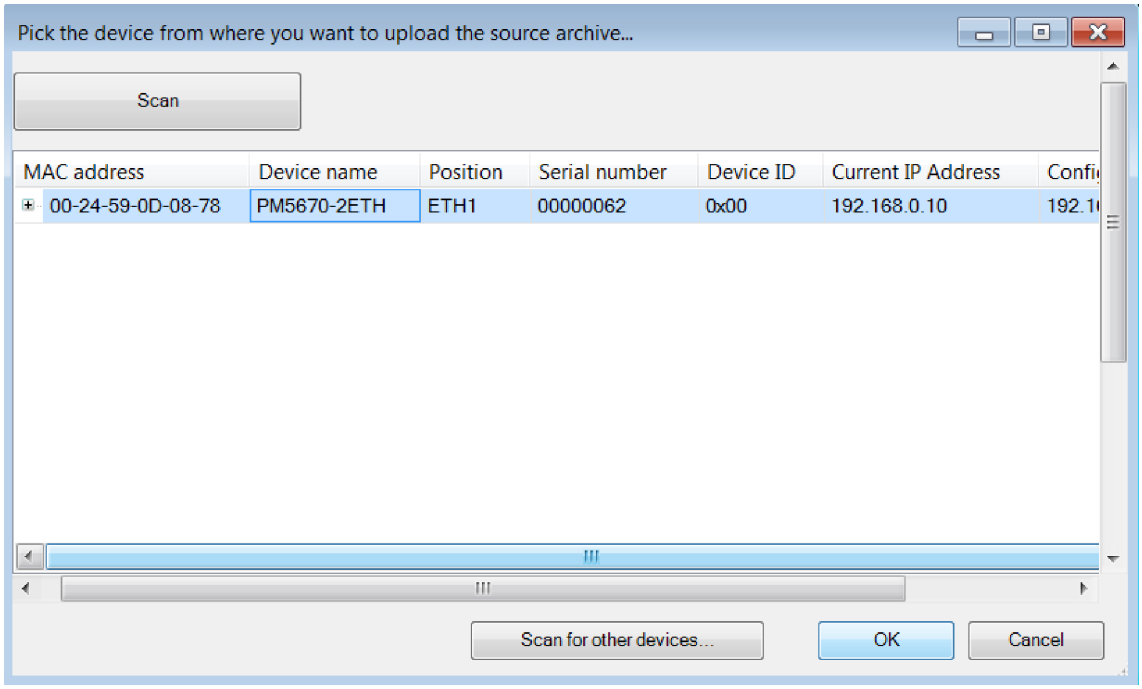


Source upload

Prerequisite

- Project archive on PLC available (from previous source download)
- PLC is connected

1. Open Automation Builder.
2. Click *“File ➔ Source upload...”*.
   ⇨ Window *Pick the device from where you want to upload the source archive...* appears.



   If you get an error click *[Scan for other devices]*.
3. Select your PLC with the archive and click *[OK]*.
   ⇨ Dialog *Extract Project Archive* appears.
4. Select your preferred folder and click *[Extract]*.
   ⇨ Then you are prompted to open the project archive.
5. Click *[Yes]*.
   ⇨ The project opens.

   Upload was successful.

1.6.6.4.2 Programming and testing

For information on programming see

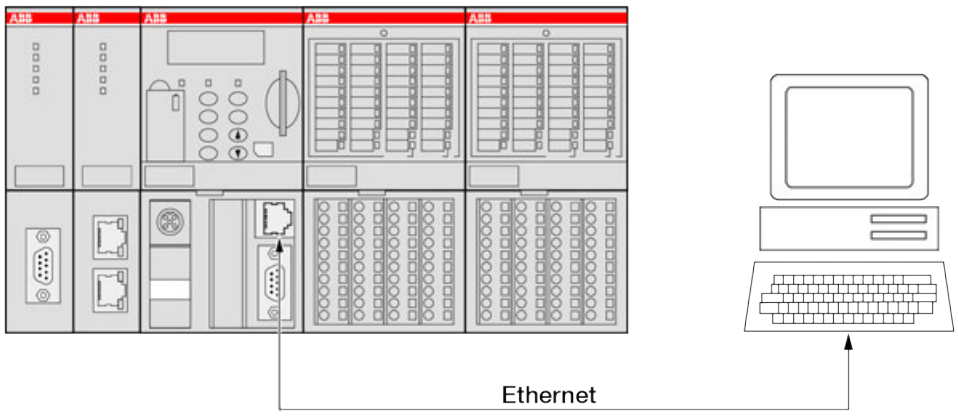
- *Chapter 1.4.1.8 “Programming of Applications” on page 222*
- *Chapter 1.4.1.10.1 “Configuring the Connection to the PLC” on page 380*
- *Chapter 1.6.6.4.3.1 “Enter a known PLC IP address” on page 3947*
- *Chapter 1.4.1.10 “Downloading an Application to the PLC” on page 379*

For Information on testing/debugging see *Chapter 1.4.1.11 “Testing and Debugging” on page 394*

1.6.6.4.3 Configuration of communication via Ethernet (TCP/IP)

Programming via Ethernet is only possible on a PC with Ethernet board and installed network. Programming can be done via the internal (onboard) Ethernet communication module.

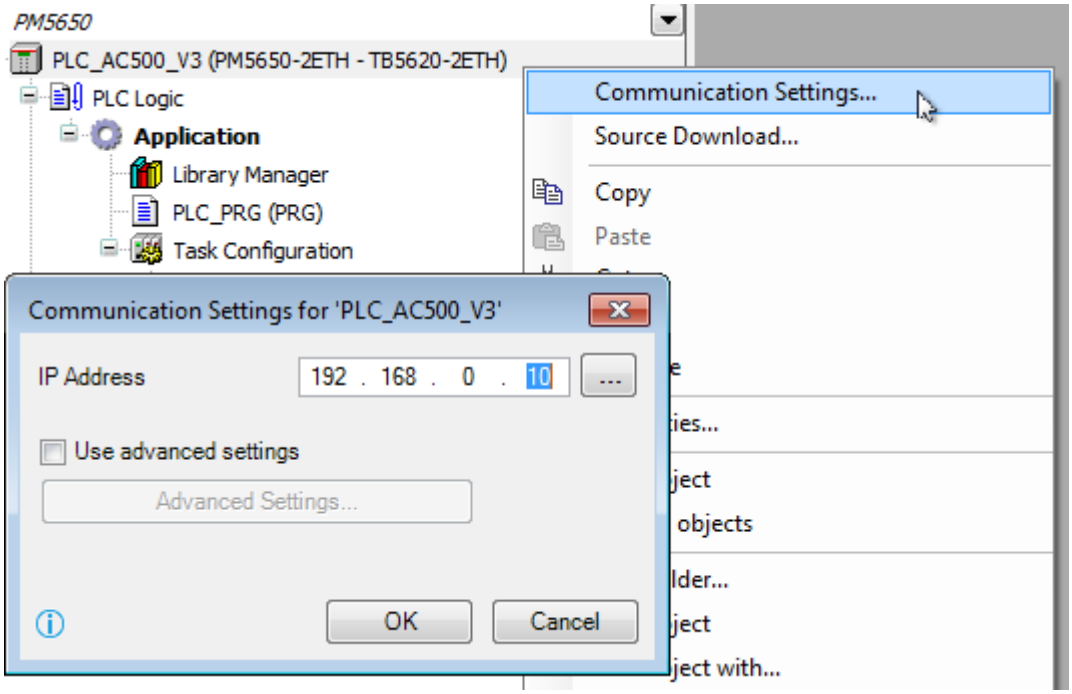
An application note describes the configuration of an AC500 V3 PLC for *EtherNet/IP communication* ↳ *Chapter 1.4.2.4 "EtherNet/IP Configurator" on page 1220*.



Enter a known PLC IP address

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.

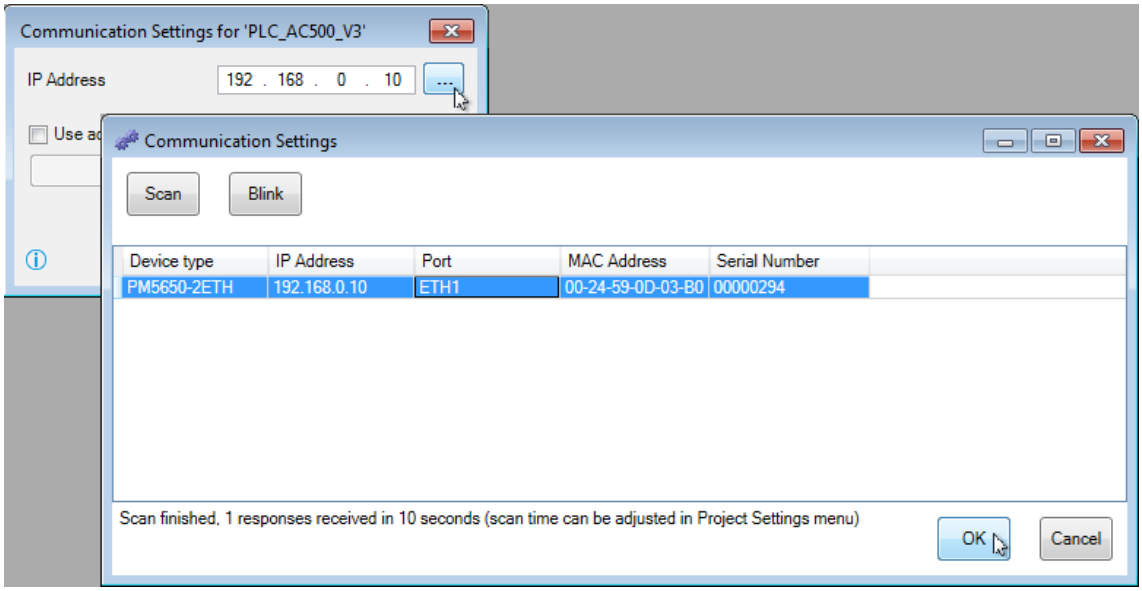


2. Enter your PLC *IP Address* and click *[OK]*.

Enter PLC IP address by scanning devices

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.



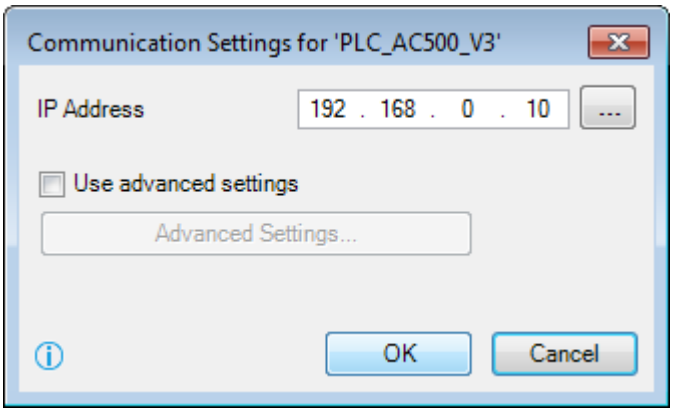
2. Click *[...]*.

   ⇨ Dialog box *Communication Settings <...>* appears.

3. Click *[Scan]*, select your desired PLC and click *[OK]*.

   ⇨ Entry is transferred to the dialog box *Communication Settings <...>*.

   Click *[OK]*.



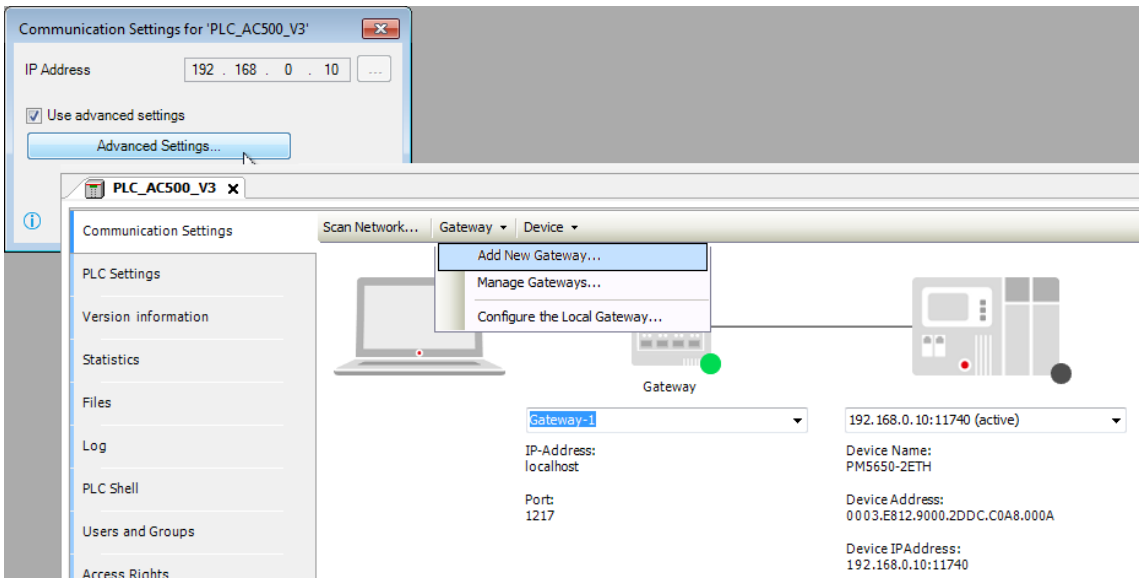
4. Click to log in the *"PLC_AC500_V3"* project.

Enter PLC IP address by *[Advanced Settings...]*

If a remote gateway instead of a local one has to be used it can be configured in the *[Advanced Settings...]*.

1. Right-click the top node *“PLC_AC500 <...>”* and select *“Communication Settings”* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.



2. Enable checkbox*Use advanced settings* and click *[Advanced Settings...]*.

   ⇨ Tab *“Communication Settings”* opens.

3. Check gateway or change if required.

   ⇨ Successful connection is indicated by green dot on the gateway icon.

4. *Manual entry of the IP address.*

   Check IP adress or change if required.

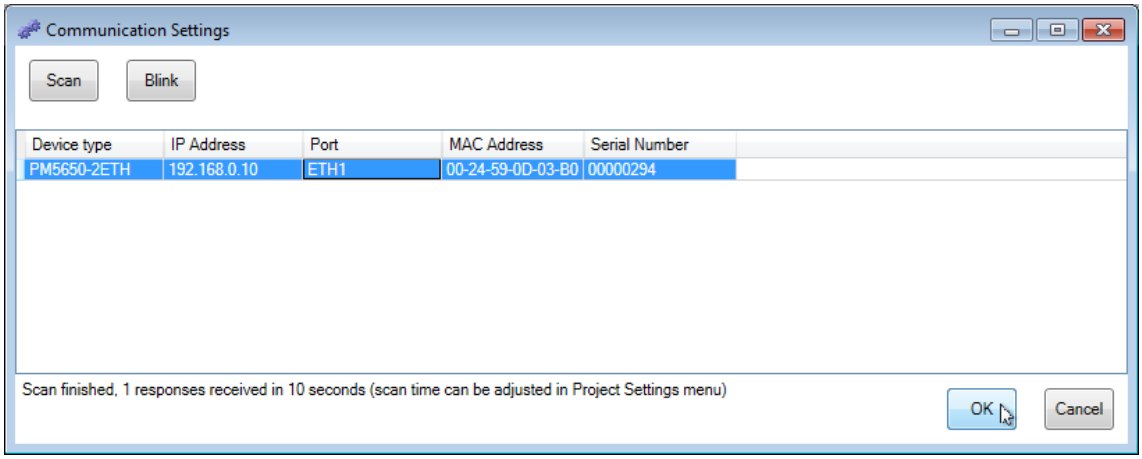
5. Press ENTER to confirm changed IP address.

   ⇨ Successful communication is indicated by green dot on the PLC icon.

6. Or instead of the last two steps:

> *Set the IP address via a scan.*

Click *[Scan Network]*, select your desired PLC and click *[OK]*.



⇨ Successful connection is indicated by green dot on the gateway icon.

7. Click to log in the *"PLC_AC500_V3"* project.

1.6.6.4.4 PLC shell commands

The PLC shell is used for requesting specific information from the controller. By entering a device-specific command the response is returned in a result window. The PLC shell can be issued without login.

Proceed as follows:

1. Ensure the gateway is configured properly and a connection to the controller can be established.
2. In Automation Builder double-click the PLC node and open the tab *"PLC Shell"*.
3. Enter "?" in the command line of the tab window. All available PLC commands are listed.

If the gateway is able to establish a connection to the controller, an online connection to the PLC is opened automatically.

> *The commands listed in online mode can differ from the commands shown when pressing the button [...] as Automation Builder version and firmware version can differ.*
>
> *See:*
>
> *Chapter 1.2.6 "Further information" on page 49*
>
> *Chapter 1.6.6.1.4 "Firmware identification and update" on page 3652.*

1.6.6.4.5 Watchlists

Chapter 1.4.1.12.1.2 "Using watch lists" on page 416

Chapter 1.4.1.12.2 "Changing Values with Recipes" on page 417

1.6.6.4.6 Reference to libraries

Library configuration is described in the chapter ↳ *Chapter 1.5 “Libraries and solutions” on page 2146*.

1.6.6.4.7 Reference to application libraries

Application libraries can be used in AC500 V3 PLCs. The requirements for the use of the function blocks of the application libraries and information and prerequisites for the general handling of application libraries are described in the application examples:

- HTTP library
  In order to be able to use the PLC as a client for web services, the HTTP function block library can be used. Setup and use are described in the *application example*.
- MySQL library
  With the help of the MySQL function block library, MySQL databases can be used to store and access AC500 V3 data. Setup and use are described in the *application example*.
- MSSQL library
  With the help of the MSSQL function block library, MSSQL databases can be used to store and access AC500 V3 data. Setup and use are described in the *application example*

1.6.6.4.8 Programming in C code

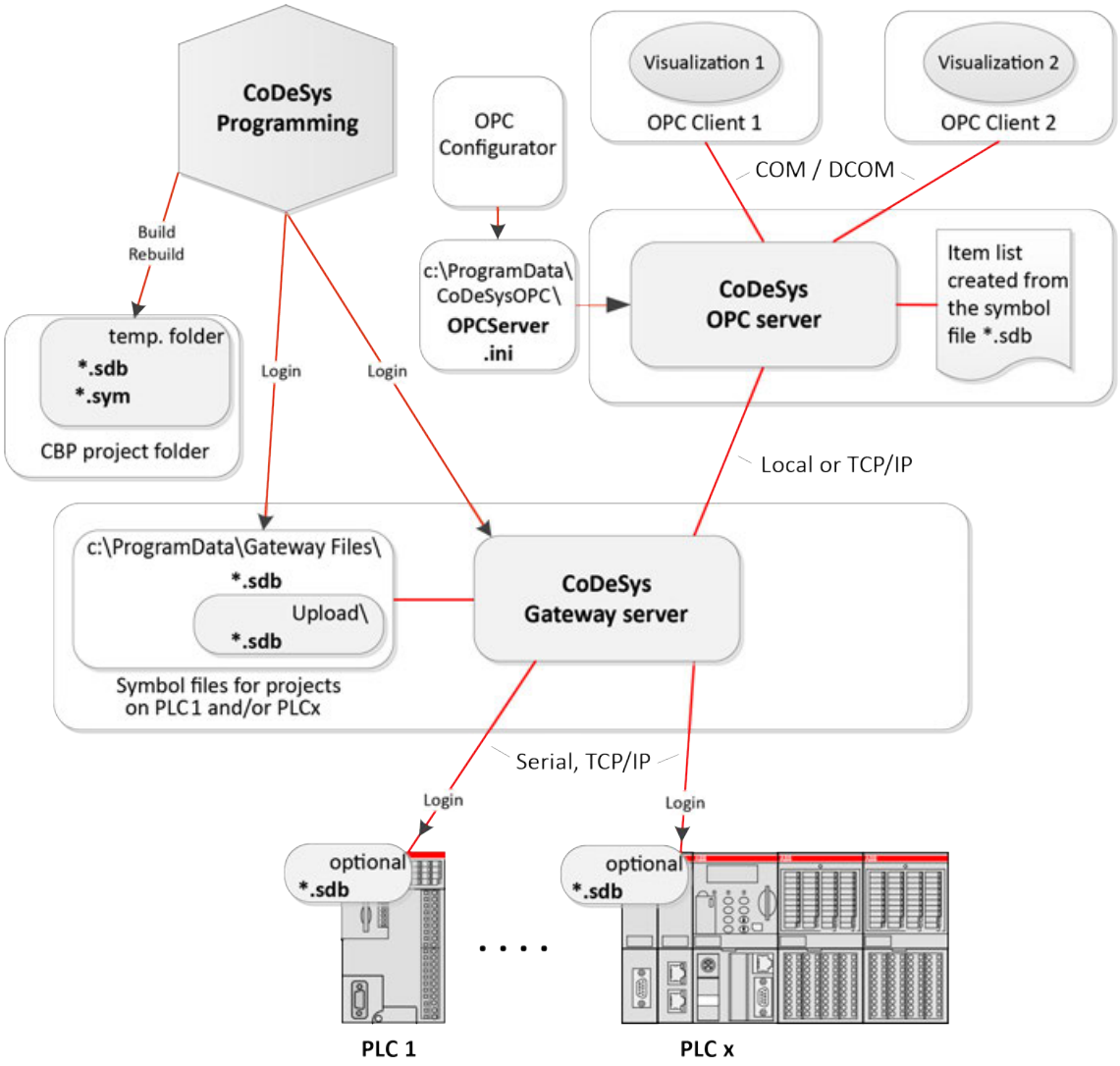
With the C code integration plugin from CODESYS, externally implemented C code files can be included in Automation Builder projects. For further information see CODESYS description ↳ *Chapter 1.4.1.8.10 “Integrating C Modules” on page 275*.

1.6.6.5 Server installation

1.6.6.5.1 OPC server for AC500 V3 products

Introduction

Architecture of the CODESYS OPC server



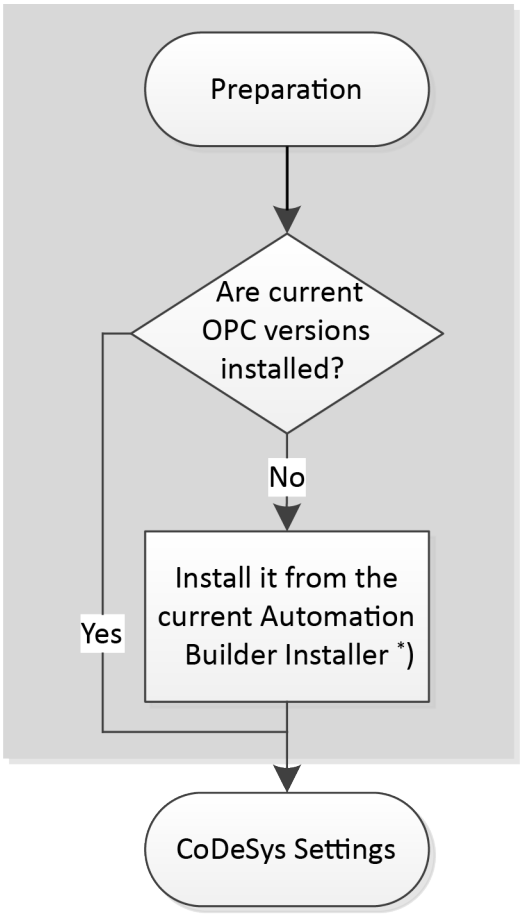
Essential documents

For further information see ↳ *Chapter 1.6.6.5.1.2 “Hints” on page 3957*.

	File name	Com-ment	Where to find
REF1	*OPC_V3_how_to_use_E.pdf* *OPC_V3_how_to_use_D.pdf*	OPC V3	C:\Program Files\ABB\CoDeSys OPC Server 3 AE
REF2	*AeConfigurator_User-Guide.pdf*	OPC V3	C:\Program Files (x86)\3S CODESYS\CODESYS OPC Server 3
REF3	*ReadMe.rtf*	OPC V3	Installation ABB DM Suit 1.0.: \PLC - AC500\OPC Server\OPC-ServerV3.xAE\
REF4	*ReleaseNotesOPCV3 AE for HA*	OPC V3	Installation ABB DM Suit 1.0.: \PLC - AC500\OPC Server\OPC-ServerV3.xAE\

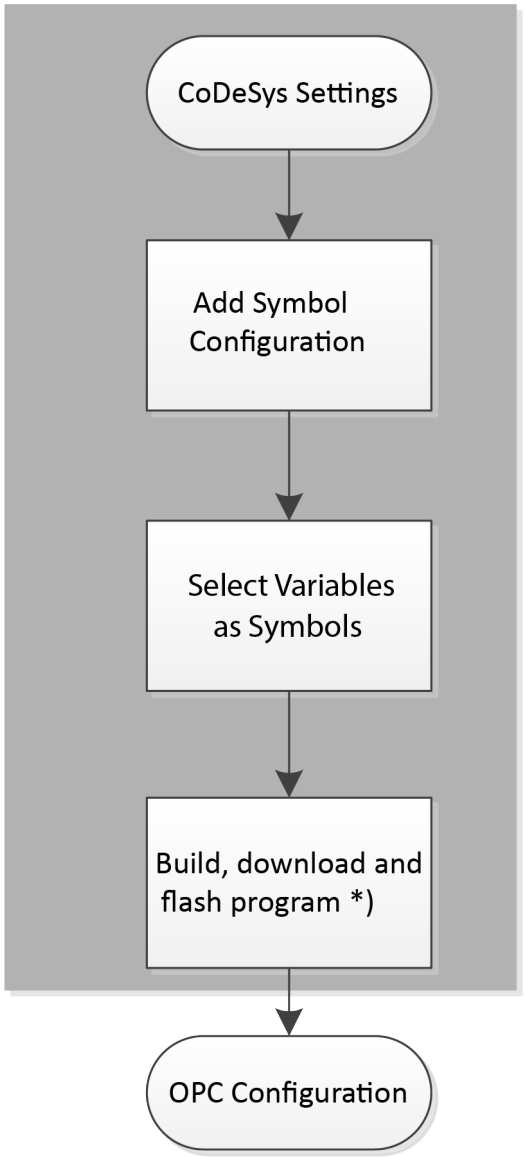
Work flow

Consideration and preparation

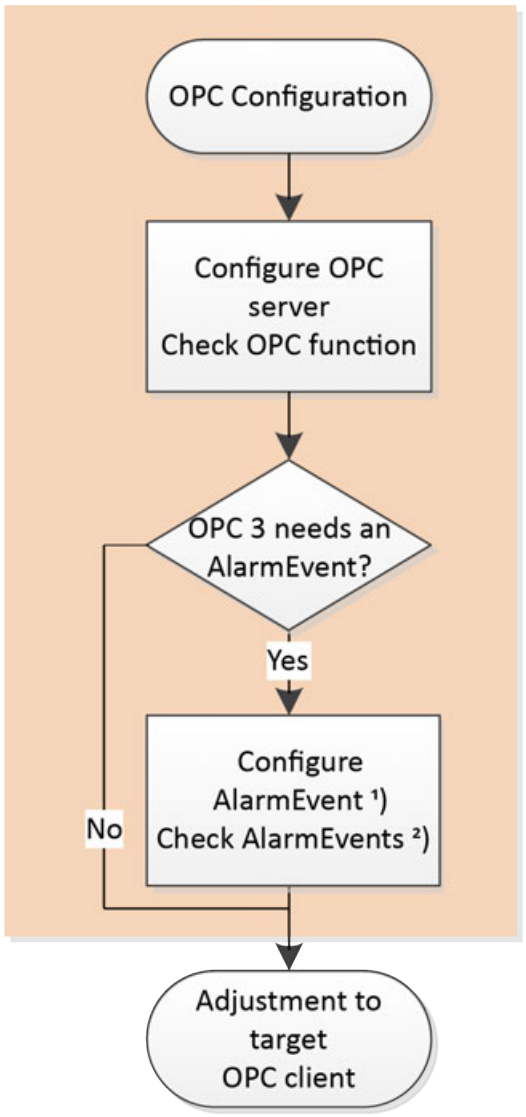


*) ↳ *Chapter 1.6.6.5.1.2.2 “Installation of OPC server” on page 3960*

Commission OPC server



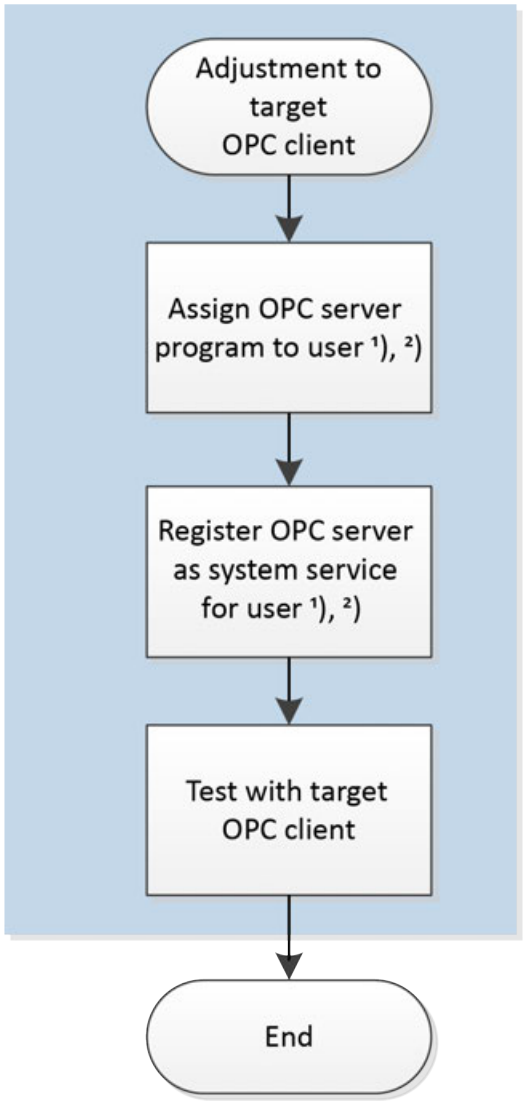
*) *Chapter 1.6.6.5.1.2.3.1 “Define symbols” on page 3963*



[1]) *Chapter 1.6.6.5.1.2.5 “Configure AlarmEvents” on page 3969*

[2]) *Chapter 1.6.6.5.1.2.5.1 “Check AlarmEvents” on page 3969*

Adjustment to target OPC client



[1]) *Chapter 1.6.6.5.1.1.2 “Essential documents” on page 3953 REF4*.

[2]) *Chapter 1.6.6.5.1.2.6 “Configure user account for OPC server” on page 3969*

Hints

Default folder and contents

Windows 7, Windows Server 2008/2016 (64-bit)

OPC Server V3	Windows 7 64-bit, Windows Server 2008 64-bit, Windows Server 2016 64-bit
WinCoDeSysOPC.exe *OPCConfig.exe* *AEConfiguration.exe* *CoDeSys_OPC_Server_V3_User_Guide.pdf* *CoDeSys_OPC_Server_V3_Benutzerhand-buch.pdf* *AeConfigurator_UserGuide.pdf*	C:\Program Files (x86)\3S CoDeSys\CoDeSys OPC Server 3\
OPCServer.ini *OPCServerA.ini* *OPCServer.log*	C:\ProgramData\CoDeSysOPC\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\ProgramData\Gateway Files\ After starting the OPC server: C:\ProgramData\Gateway Files\Upload\
Gateway.exe	C:\Windows\SysWOW64\

Windows 7 (32-bit), Windows Server 2008/2016 (32-bit)

OPC Server V3	Windows 7 32-bit, Windows Server 2008 32-bit, Windows Server 2016 32-bit
WinCoDeSysOPC.exe *OPCConfig.exe* *AEConfiguration.exe* *CoDeSys_OPC_Server_V3_User_Guide.pdf* *CoDeSys_OPC_Server_V3_Benutzerhand-buch.pdf* *AeConfigurator_UserGuide.pdf*	C:\Program Files\3S CoDeSys\CoDeSys OPC Server 3\
OPCServer.ini *OPCServerA.ini* *OPCServer.log*	C:\ProgramData\CoDeSysOPC\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\ProgramData\Gateway Files\ After starting the OPC server: C:\ProgramData\Gateway Files\Upload\
Gateway.exe	C:\Windows\System32\

Windows Server 2008/2016 (32-bit)

OPC Server V3	**Windows Server 2008 32-bit, Windows Server 2016 32-bit**
WinCoDeSysOPC.exe *OPCConfig.exe* *AEConfiguration.exe* *CoDeSys_OPC_Server_V3_User_Guide.pdf* *CoDeSys_OPC_Server_V3_Benutzerhand-buch.pdf* *AeConfigurator_UserGuide.pdf* *OPCServer.ini* *OPCServerA.ini* *OPCServer.log*	C:\Program Files\3S CoDeSys\CoDeSys OPC Server 3\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\WINDOWS\Gateway Files\ After start CODESYS OPC server: C:\WINDOWS\Gateway Files\Upload\
Gateway.exe	C:\Windows\System32\

*If folder **C:\ProgramData** is missing, select "Show hidden files, folders and drives" at "Control Panel ➔ All Control Panel Items ➔ Folder Options ➔ View ➔ Hidden files and folders".*

Installation of OPC server

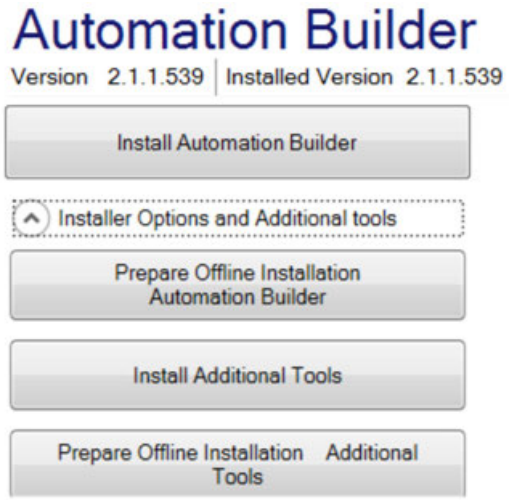
Prerequisites

The following applications are closed:
- *All OPC clients*
- *ABB OPC tunnel*
- *CODESYS gateway server*

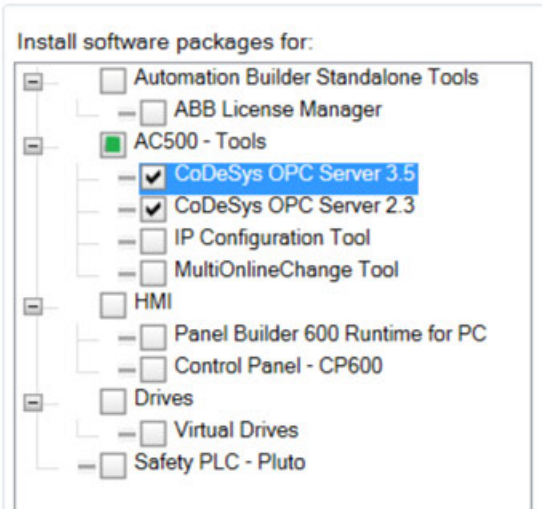
Ensure termination of the following processes:
- *Gateway.exe*
- *CoDeSysOPC.exe*
- *WinCoDeSysOPC.exe*
- *OCTsvc.exe*

Installing with Automation Builder

1. Go to homepage *http://new.abb.com/plc/automationbuilder/platform/software* .
2. Click button of *Latest Automation Builder version (recommended)* and run the installer.



3. Open *"Installer Options and Additional Tools"* and click *[Install Additional Tools]*.
4. Agree to the *"License Terms"*.



5. Select *“Version 2 and/or 3”* and install.

   ⇨ All required files are installed for OPC and the OPC server is registered automatically as user application.

Manual registration and unregistration

It is possible to register or to uninstall the OPC server manually either as COM server (user application) or as a service.

Register the OPC server as interactive software in the Windows registry:

Command for OPC 3: ***`WinCoDeSysOPC/RegServer`***

Register the OPC server as system service:

Command for OPC 3: ***`WinCoDeSysOPC/Service`***

Unregister the OPC server from the Windows registry and from the service entry:

Command for OPC 3: ***`WinCoDeSysOPC/UnRegServer`***

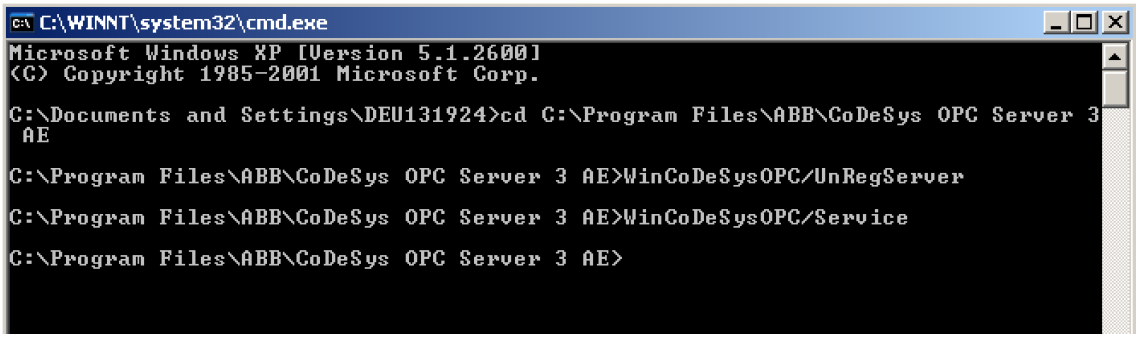
Please see REF1 chapter 3 (OPC 3) ↳ Table on page 3953 for details.

Register OPC server V3 as a system service

Prerequisites

- All programs, processes and services which connect to the OPC server are closed.

1. Start the *"Command Prompt"* with command *"cmd"* in the *"Start ➔ Run... "* window.



2. Go to the *CoDeSysOPC V2* installation folder.
3. Unregister the OPC server with `WinCoDeSysOPC/UnRegServer`.
4. Register the OPC server as system service with `WinCoDeSysOPC/Service`.

OPC clients for tests

Free of charge test clients can be found in the web:

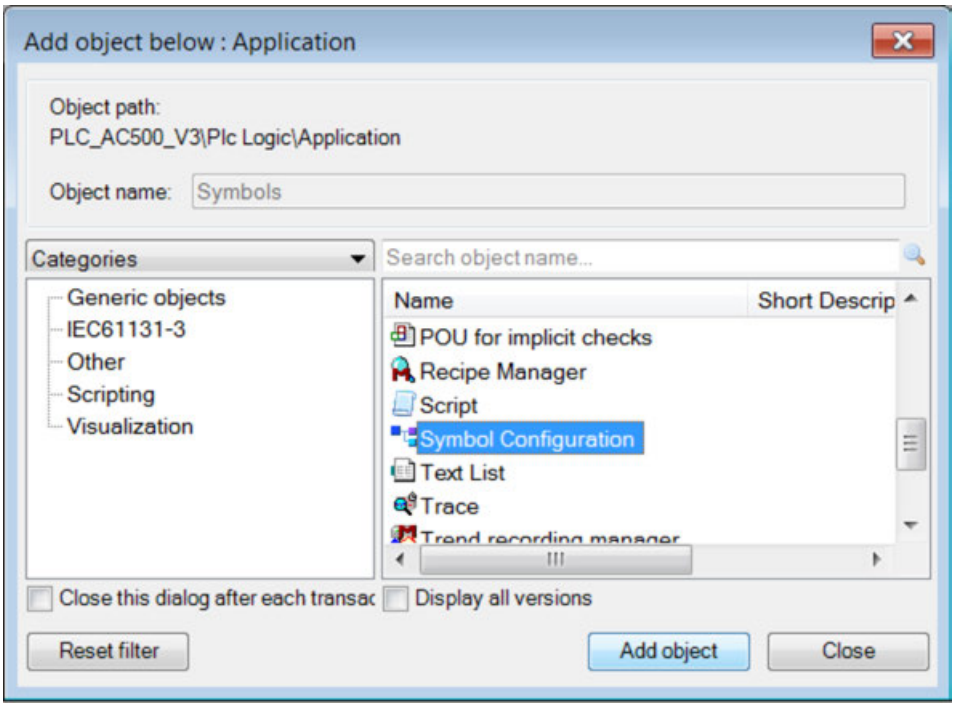
https://industrial.softing.com/us/downloads.html

http://www.matrikonopc.com/products/opc-desktop-tools/index.aspx

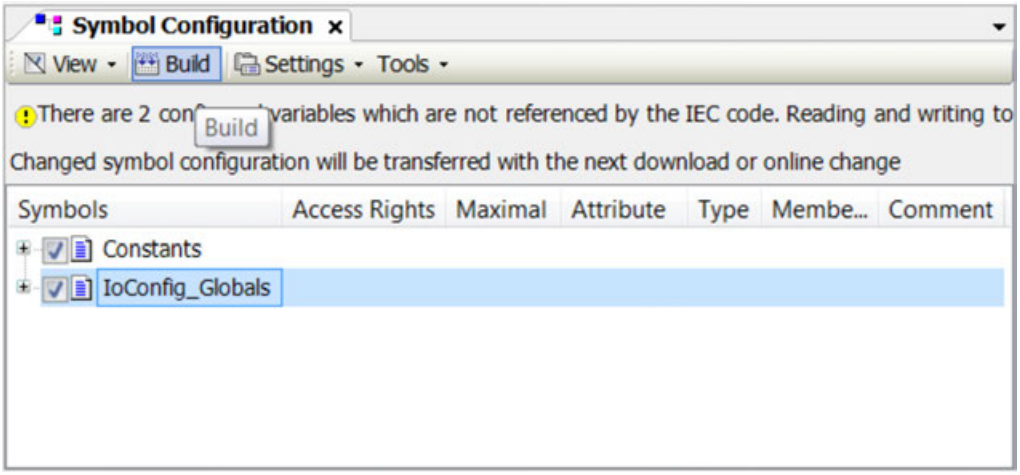
Symbol file

Define symbols

1. Right-click on *"Application"* in CODESYS V3 project and click *"Add Object"*.



2. Choose *"Symbol Configuration"* and click *[Add object]*.



3. Select your programs and/or single symbols and click *[Build]*.
   ⇨ A symbol file will be automatically downloaded to the PLC with Project Download.

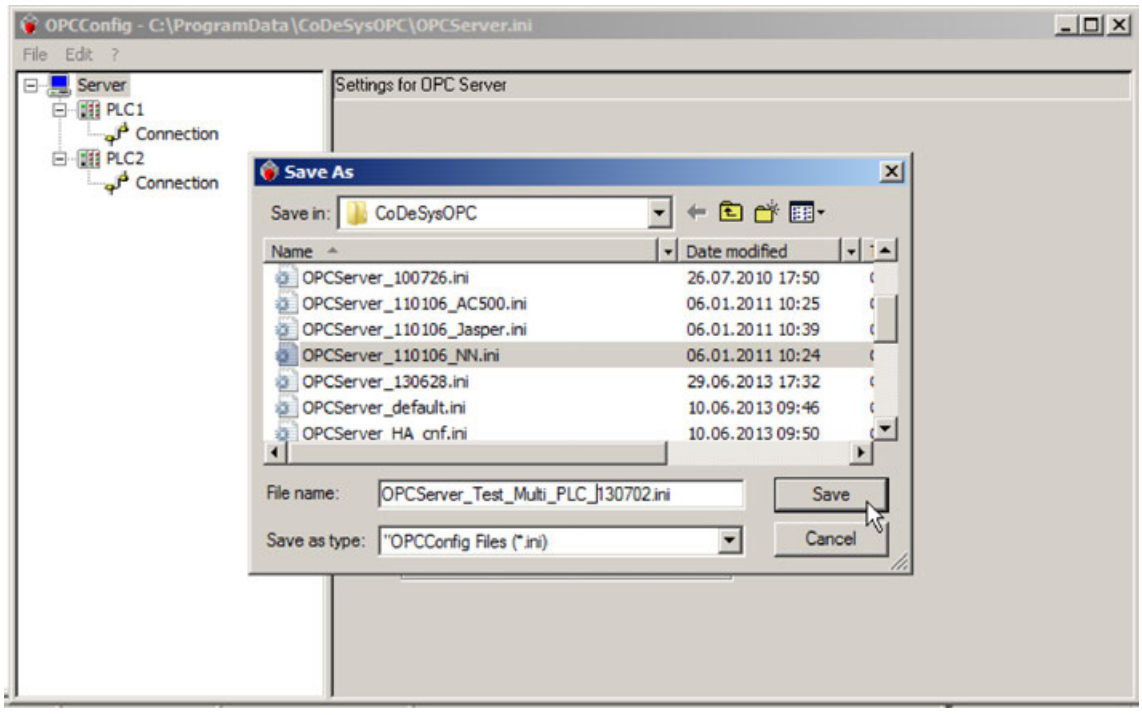
With double-click in the device tree to *"Symbol Configuration"* you can change the *"Symbol Configuration"* settings.

> *To restrict traffic and load, choose only symbols you need.*

Configure OPC server

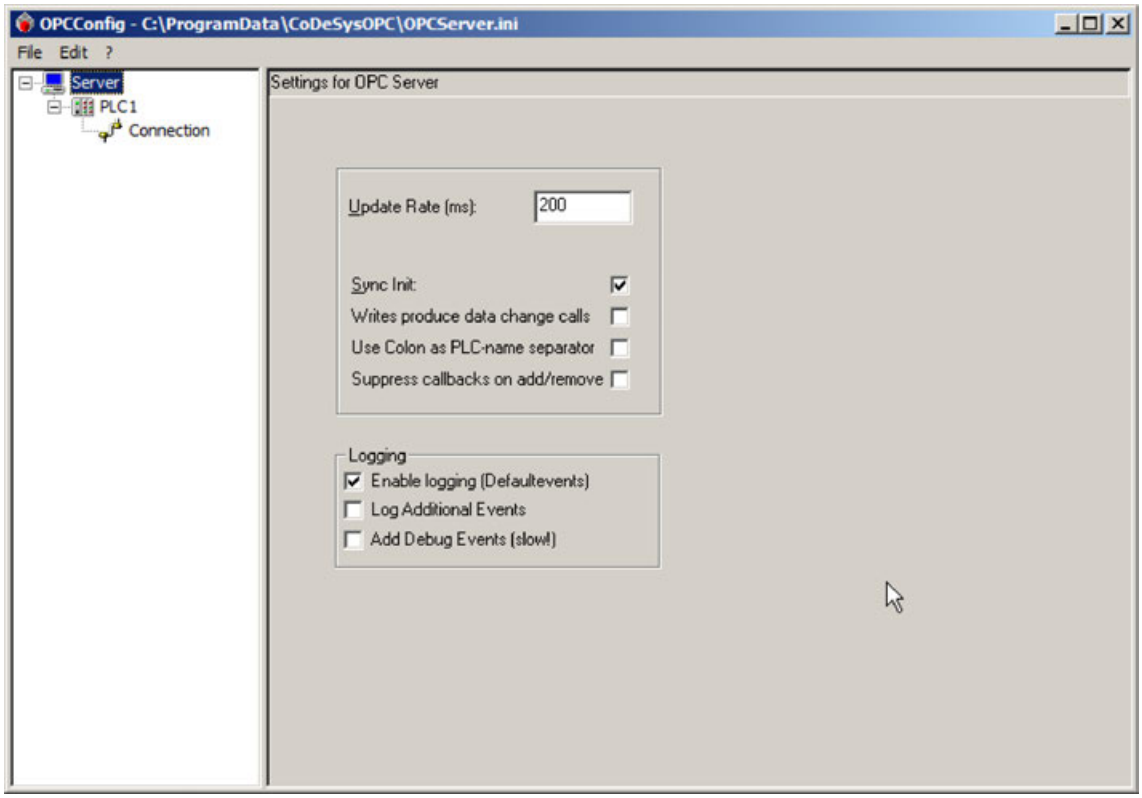
Configure OPC Server V3

1. Start CODESYS/ CoDeSysOPC Server V3/OPC Configurator.



2. If the configuration is needed furthermore, save the configuration.

   The actual configuration at start of OPC server will always be read from *OPCServer.ini*.



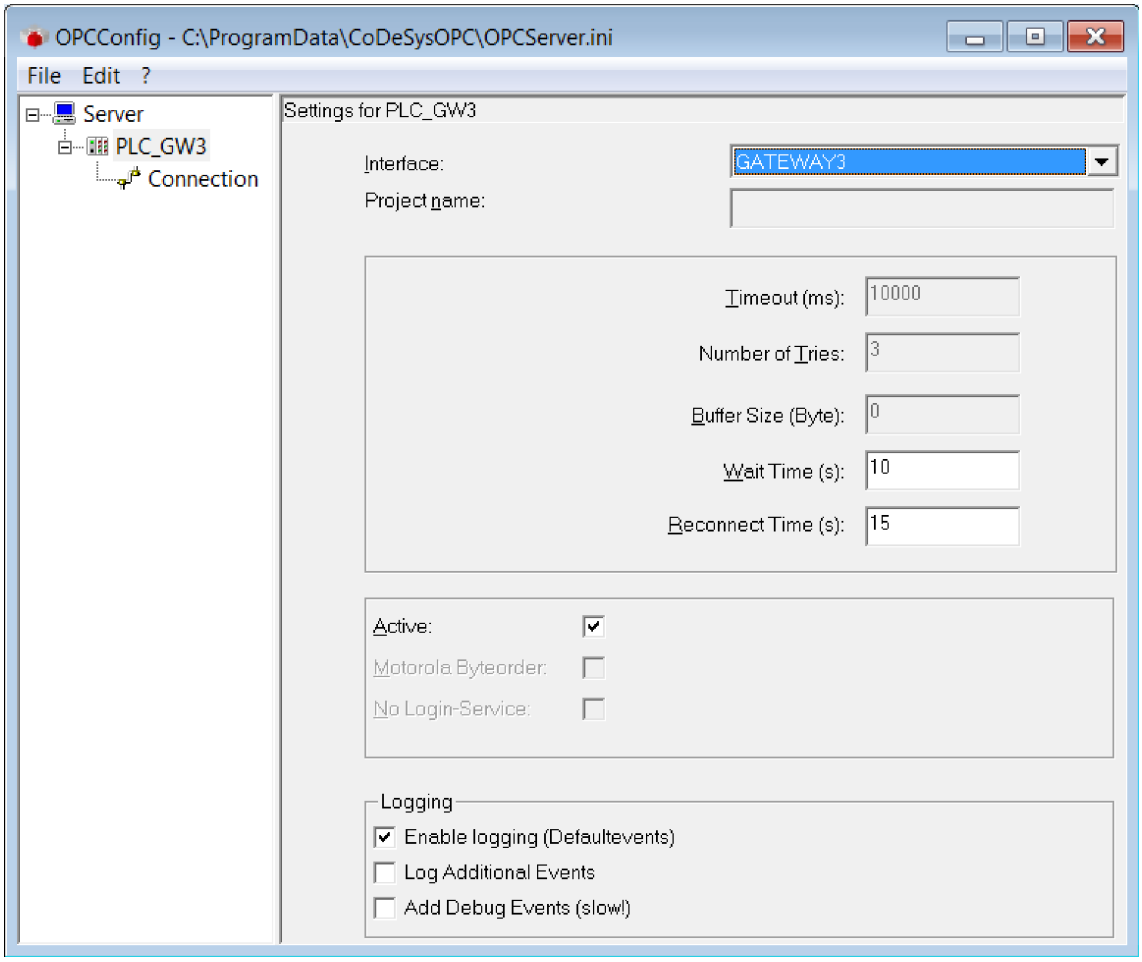
⇨

Update rate

- *The Update Rate may not be 0 (ms)!*
- *The default value of 200 ms is a suitable value of many applications.*
- *The adjustment for the Update Rate depends on the number of symbols (variables).*
- *For a big number of symbols it can be better to increase the Update Rate.*

The checkboxes Sync Init and Enable logging (Defaultevents) must be enabled.

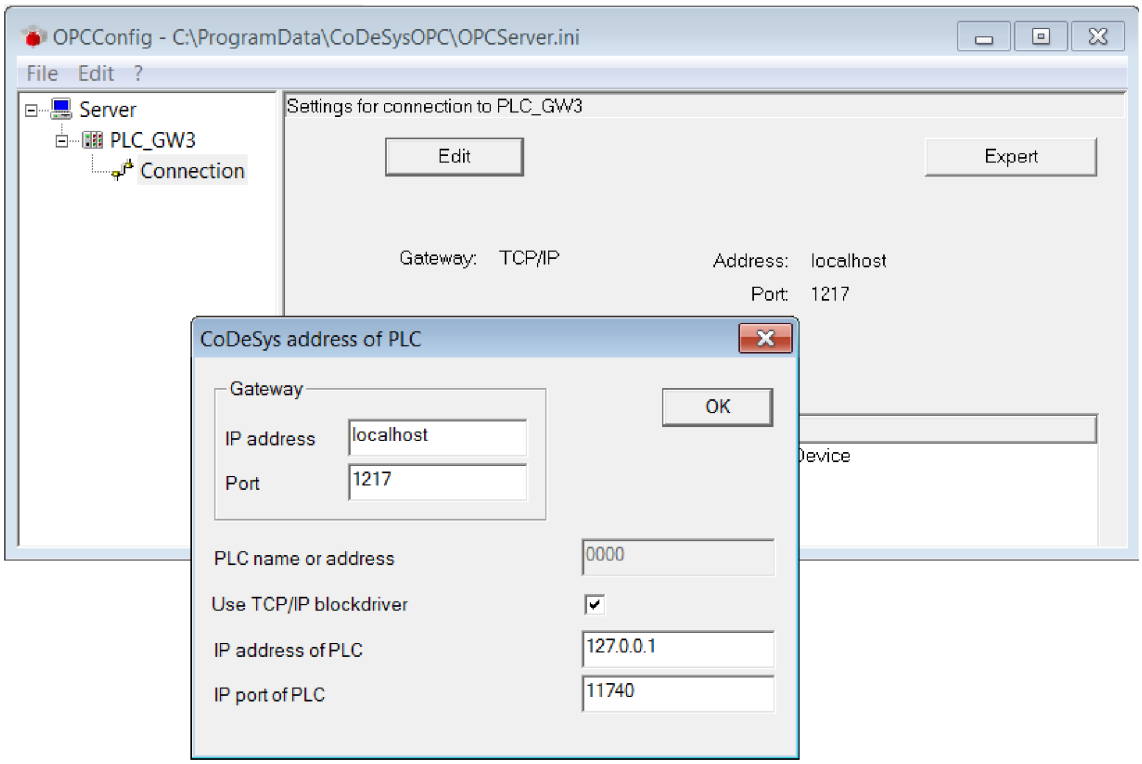
3. Select *“PLC_FW3”*.



⇨

- *If the *.sdb files should be loaded from the “Gateway Files” directory on PC, the project name must be identical with project name in CODESYS. The extension is not necessary.*
- *If the symbol information should be loaded from AC500 V2.x, the project name is not required and can also be empty.*
- *The parameters displayed in the screenshot above are recommended default settings.*
- *The checkbox Active must be enabled.*
- *Enabled checkbox “Enable logging” allows a later diagnosis.*

4. Select *"Connection"* and click *[Edit]*.



5. Enter the TCP/IP address of the target PLC at *PLC name or address* and enable *Use Tcp/Ip blockdriver*.
6. Enter the TCP/IP address of the target PLC at *IP Address of PLC* and click *[OK]*.
7. Click *"File ➔ Save" OPCserver.ini* and *"File ➔ Exit" OPCConfig*.

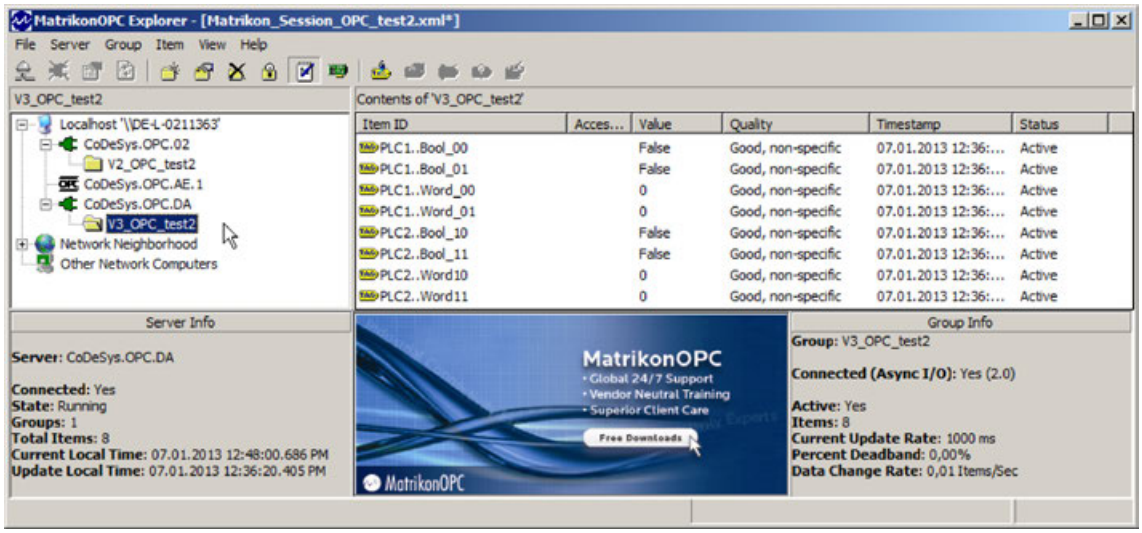
Check OPC function with AC500

It is urgently recommended to check the function of the previous configuration steps.

In order to check the OPC function without AC500, see ↳ Chapter 1.6.6.5.1.4.1 "Test OPC function without AC500" on page 3974.

Check OPC server V3

1. Start *OPCExplorer.exe* and connect *“CoDeSys.OPC.DA”*.

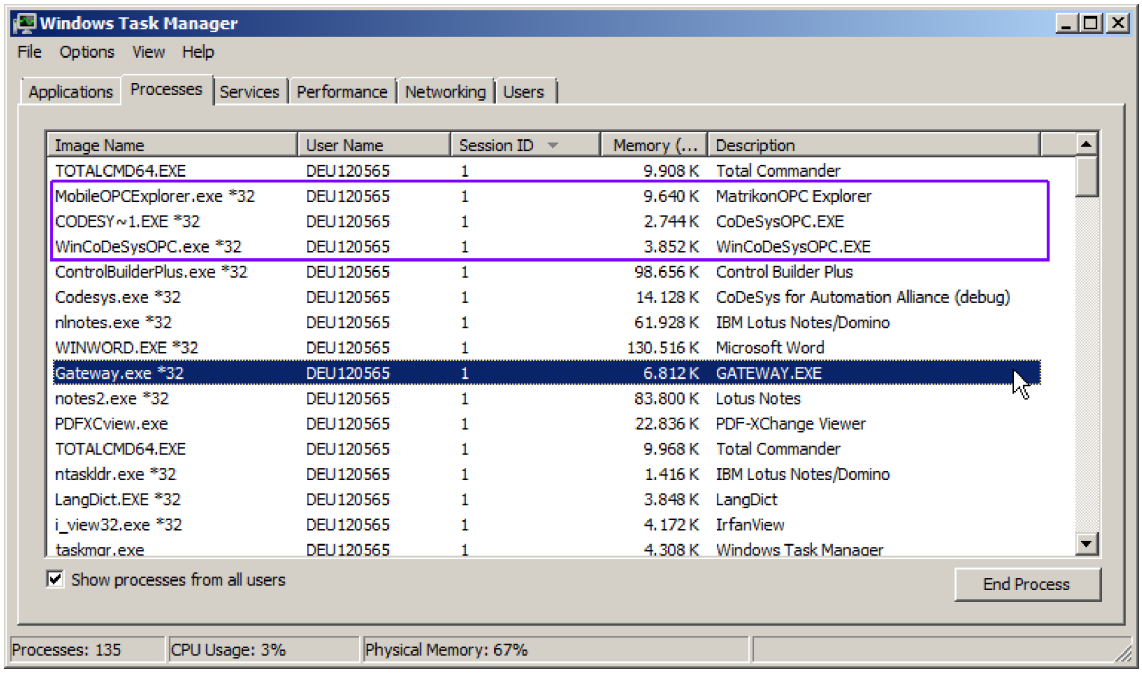


2. Add Group, add Items, select availabe Items in Server *“CoDeSys.OPC.DA”*.

   Add to Tag List, close the Item browser.

   ⇨ If anything is right, then *“CoDeSys.OPC.DA”* is connected, is running and the *“Quality”* of the items is good.

Check processes with windows task manager



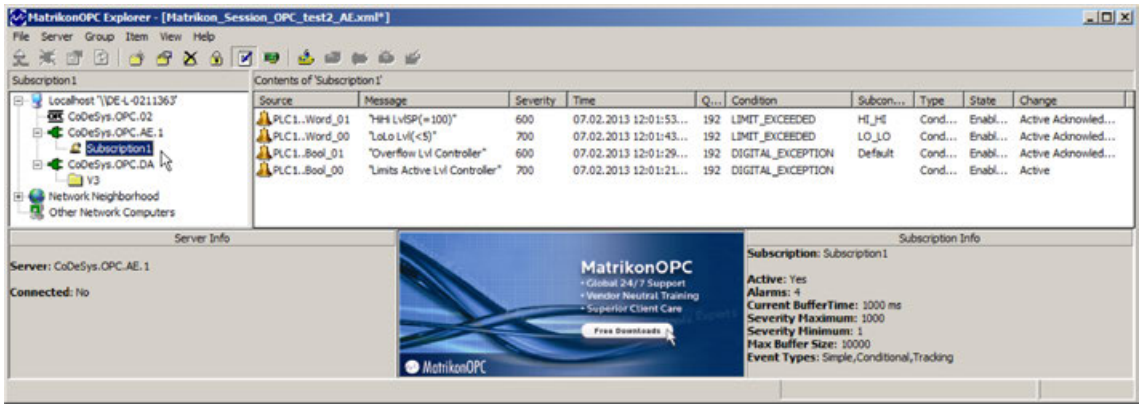
Correct configuration: All processes run with the same *“User Name”* and with the same *“Session ID”*.

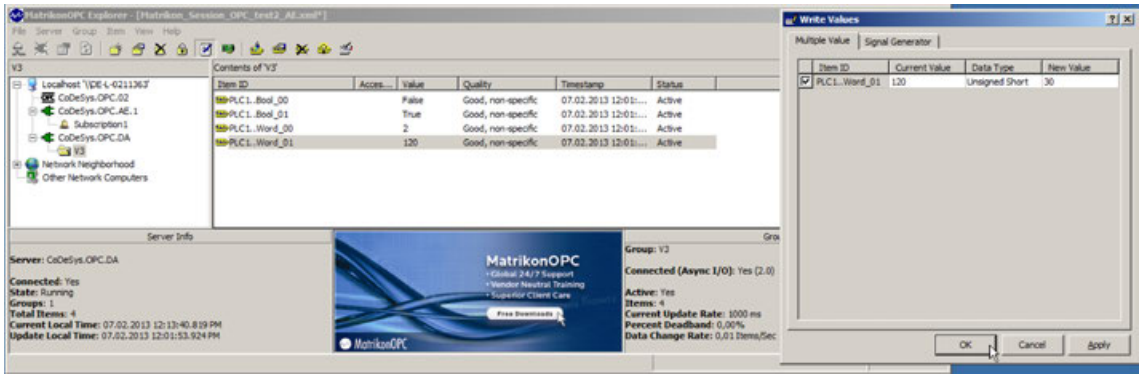
Configure AlarmEvents

> *Refer to REF2 AeConfigurator_UserGuide.pdf for details* ↳ *Table on page 3953.*

Check AlarmEvents

The function of the *“AlarmEvents”* can be checked with *“MatrikonOPC Explorer”*.





The *“AlarmEvents”* can be simulated by writing the value of the Items.

Configure user account for OPC server

> *Please refer to REF3 ReadMe.rtf and REF4 ReleaseNotes OPCV3 AE for HA* ↳ *Table on page 3953.*

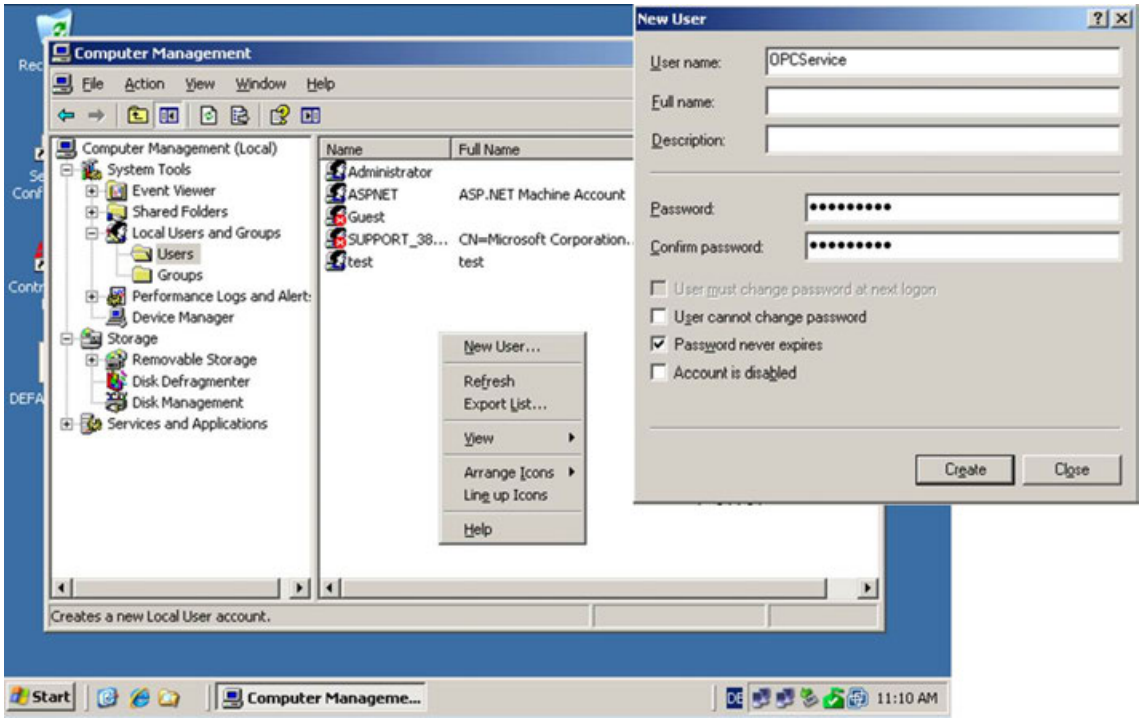
OPC server V3 on Windows Server 2003/ 2008/ 2012/ 2016

When running the OPC server V3 on Windows Server 2003/ 2008/ 2012 /2016 multiple sessions need to be supported. Therefore the installation of the OPC server as service running with a dedicated user account is recommended.

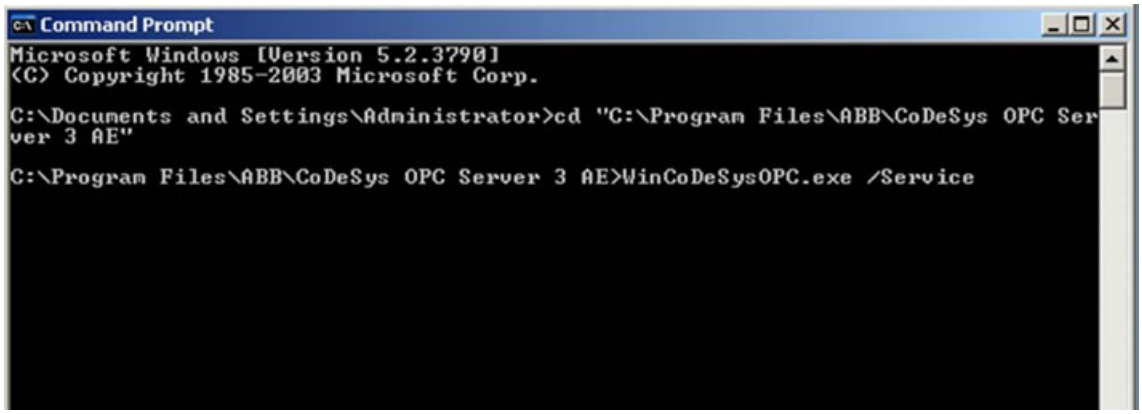
Configuration steps

- Create specific user, no administrator account is required
- Register V3 OPC server as service
- Configure V3 OPC server as service

Create specific user



Registration



Register the OPC Server executable as service from the command line.

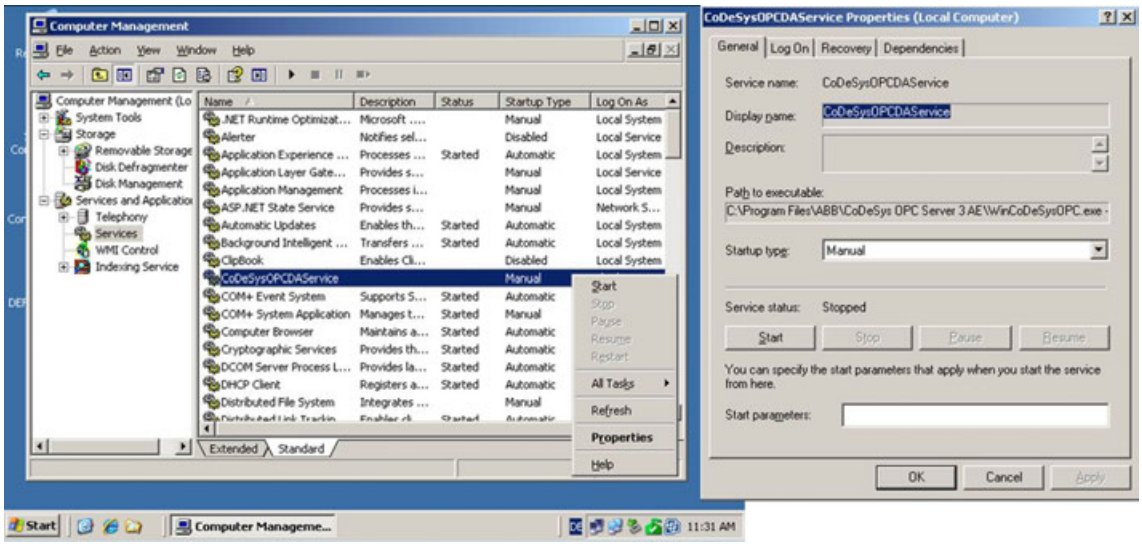
With command “WinCoDeSysOPC /Service” WinCoDeSysOPC.exe gets installed as system service.

Started once, the service will stay “started” until the system gets terminated.

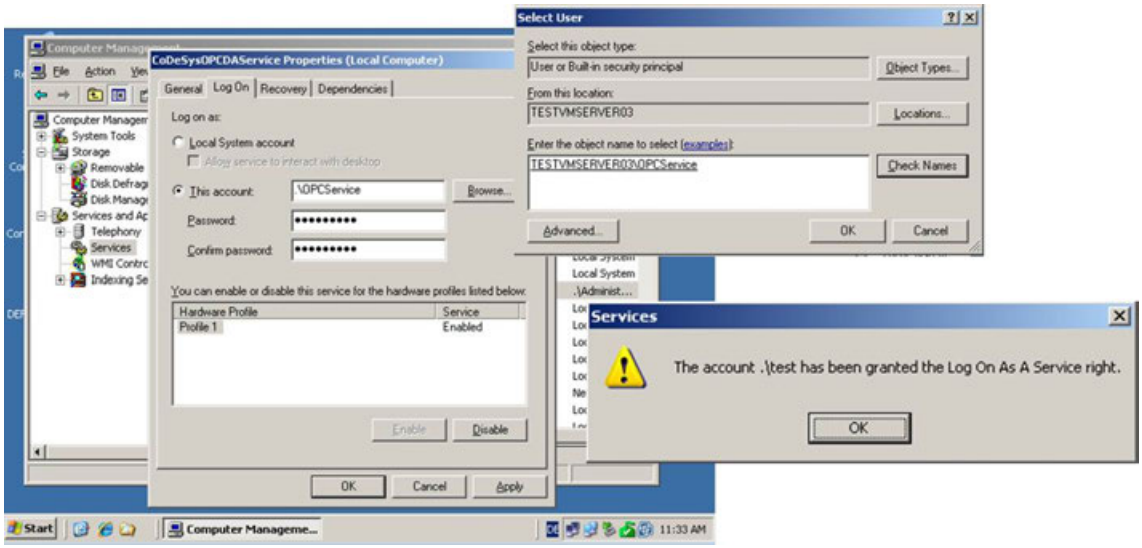
The communication to the configured PLCs survives.

Also here the service gets installed in the current position of WinCoDeSysOPC.exe.

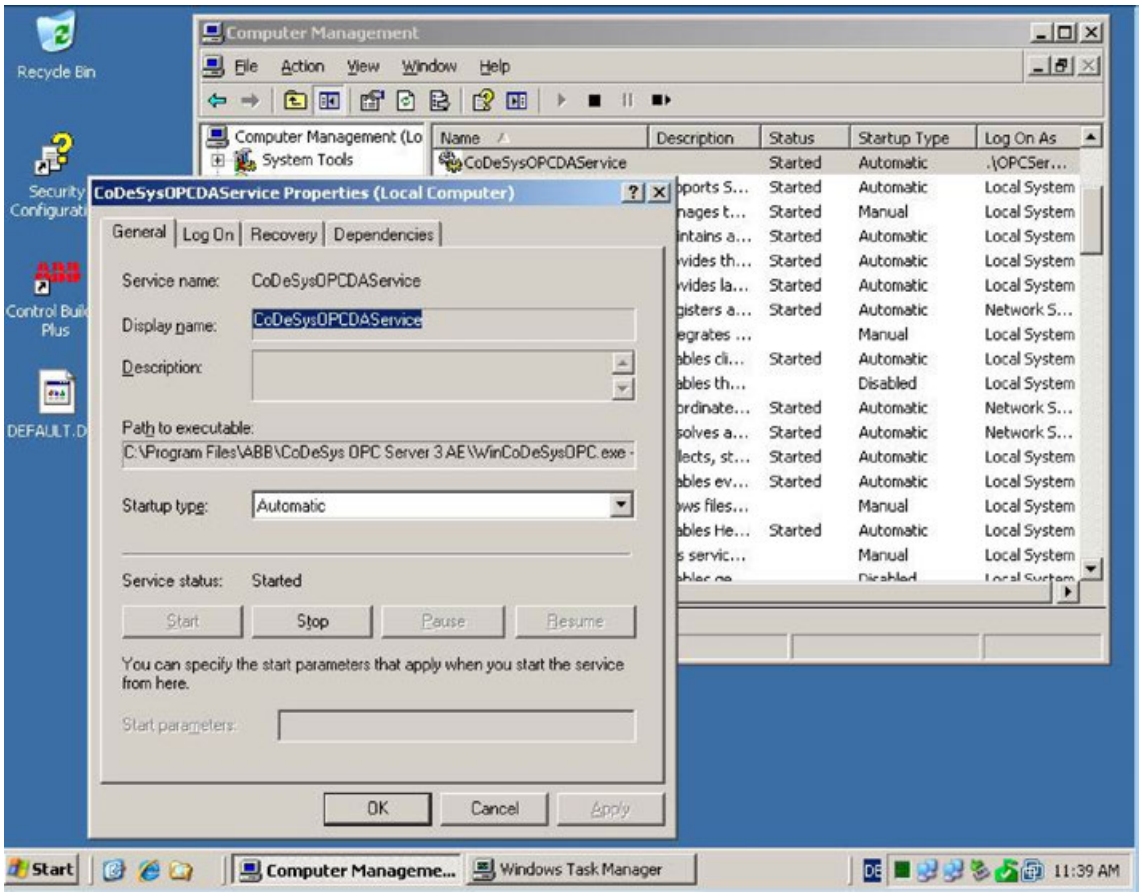
Configuration



At *“Computer Management* ➔ *Services and Applications* ➔ *Services”* open the *“Properties”* of the *“CoDeSysOPCDAService”*.

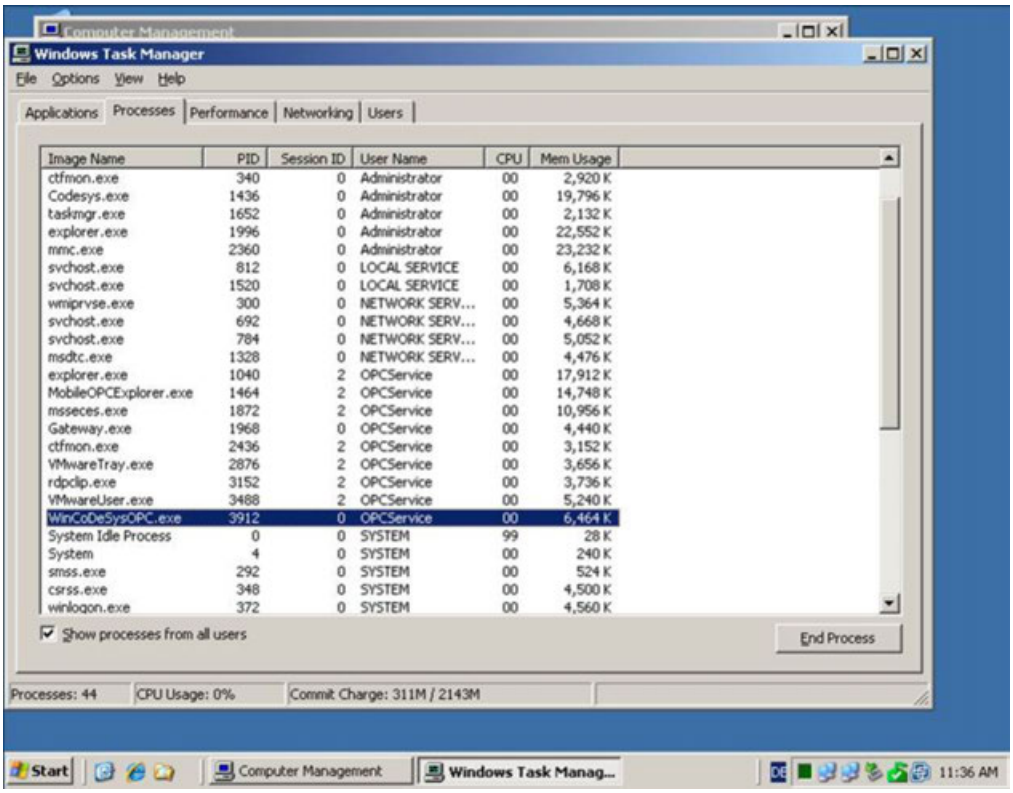


Complete the Service Configuration



Testing

Check Users and Session during Test Cases



Check the *“Session ID”* and *“User Name”* of

- *Gateway.exe*,
- *WinCoDeSysOPC.exe* and
- OPC Client

on different test cases like multi session with terminal service sessions.

Potential issues

Session isolation

Situation

With Windows Server 2003, Windows Server 2008, Windows Server 2016 the Windows 7 services are alone in session 0. User applications run in session 1 (2 and so on).

Services:

A Windows service is a computer program that operates in the background.

Windows services can be configured to start when the operating system is started or can be started manually and run in the background as long as Windows is running. They can operate when a user is not logged on.

Services are:

Windows operating systems include numerous services. OPC client like S+ OPC scanner PGIM, Aspen CIM-IO Manager, ICONICS, .. can also installed as a service.

User applications are:

Microsoft Word, Notepad, MatrikonExplorer, *ControlBuilderPlus.exe* and *Codesys.exe*

Problem

Service and user application are isolated in their session. They can not communicate with each other directly.

OPC Server uses, like the CBP and CODESYS, the gateway server from CODESYS (*gateway.exe*) for the communication with the AC500 and starts the gateway in their session. That creates undefined behavior, if the OPC Server runs as a service. The gateway server is not able to run in multi sessions.

Resolutions

- Install all OPC clients and OPC Server, which use the gateway server, in the same session.
- The OPC Server as a service (session 0) may not be connected at the same time (in parallel) with an OPC server as a user application or CBP or CODESYS (all in session 1) with the AC500. If this function is necessary, different PC or virtual machines must be used.
- Use tools like OPC tunnel. In a DigiVis 500 setup context the OPC server must not be registered as service. The OPC tunnel itself starts the OPC server within its service.

See also http://msdn.microsoft.com/en-us/windows7trainingcourse_sessionisolation_unit .

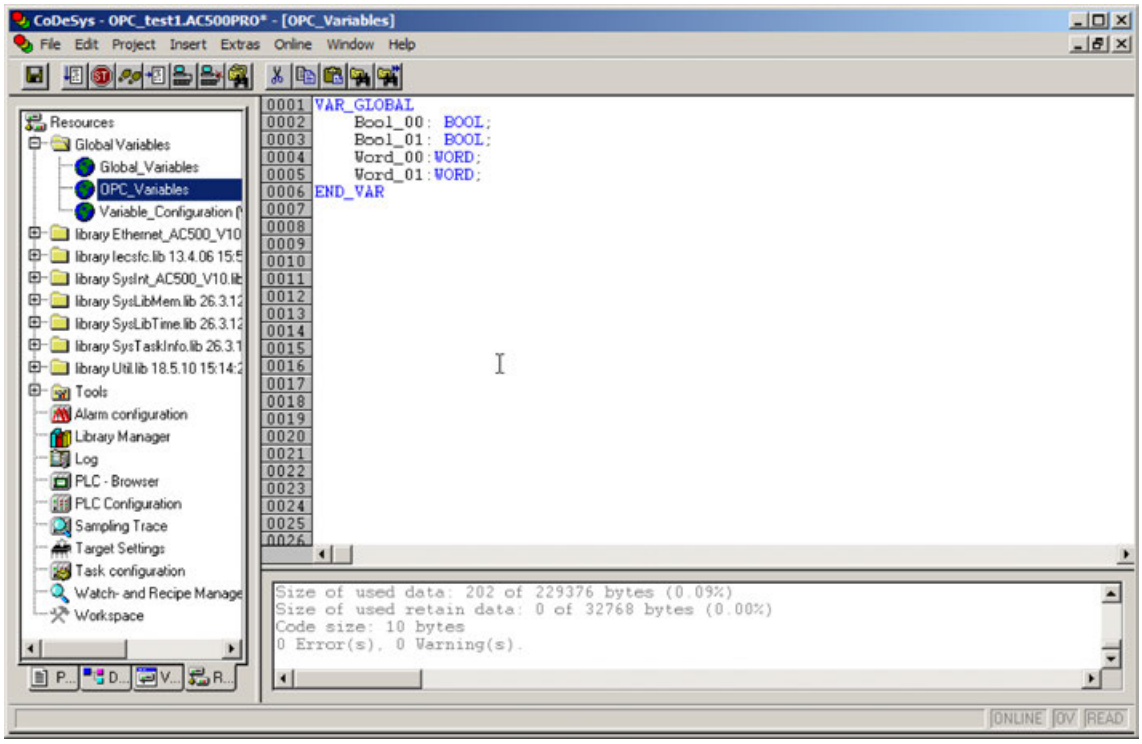
Examples

Test OPC function without AC500

The example shows, how the OPC server V2/V3 can be tested/simulated without available AC500.

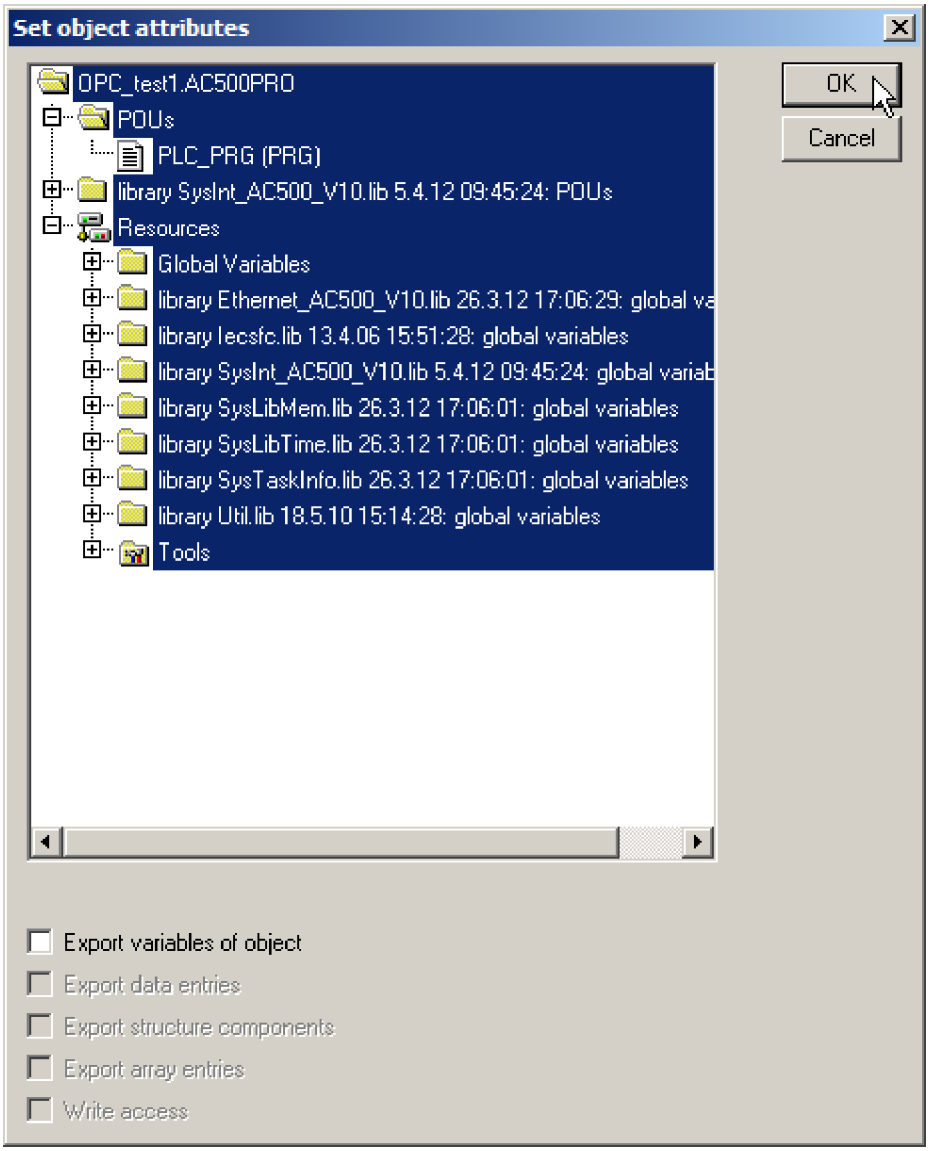
AC500 project

1. Open *CoDeSys Application*.



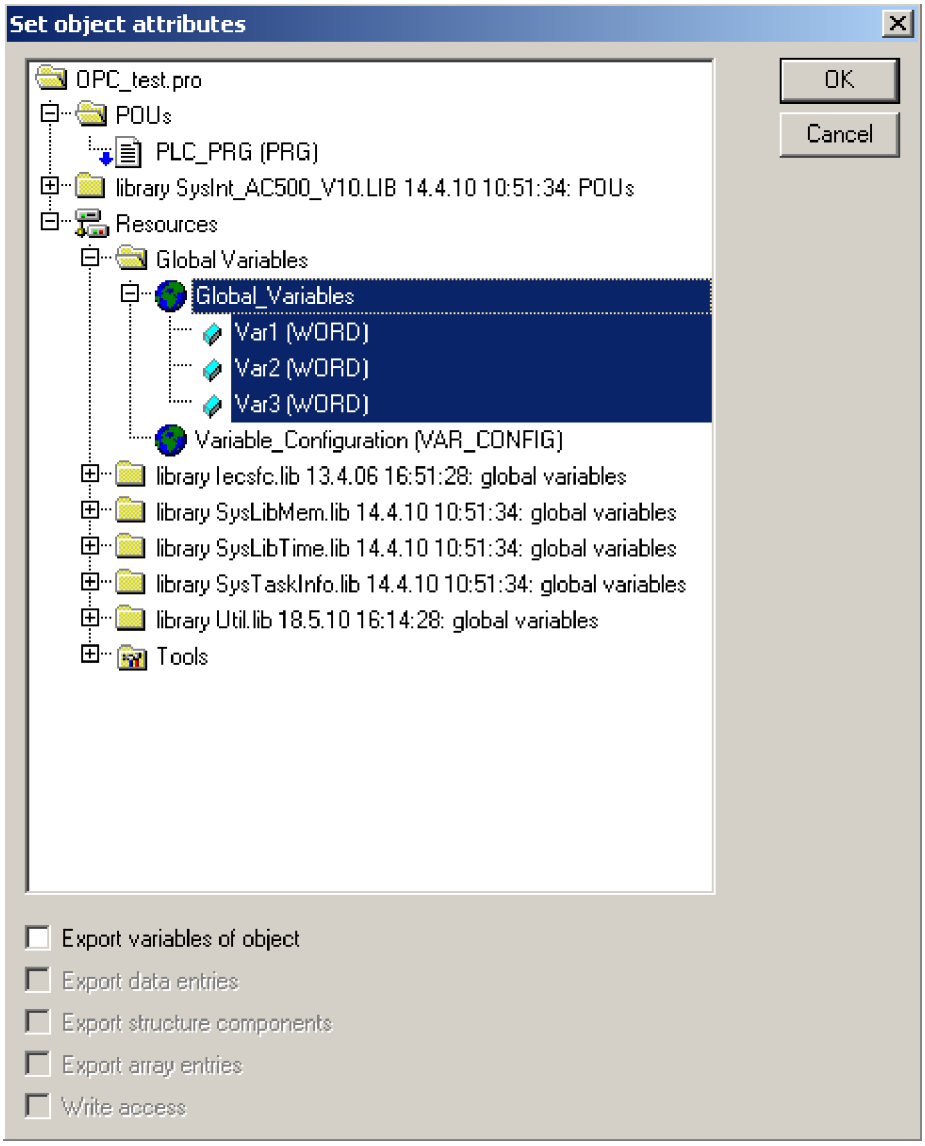
2. Collect all OPC variables in a separate *“Global Variables”* list.

3. Under *"Project ➔ Options "* select the *"Symbol configuration"*.
   Enable checkbox *"Dump symbol entries"* and click *[Configure symbol file]*.



4. Disable all the checkboxes and confirm twice with *[OK]*.

5. Under *"Project ➔ Options " "Symbol configuration"* click *[Configure symbol file]* again.

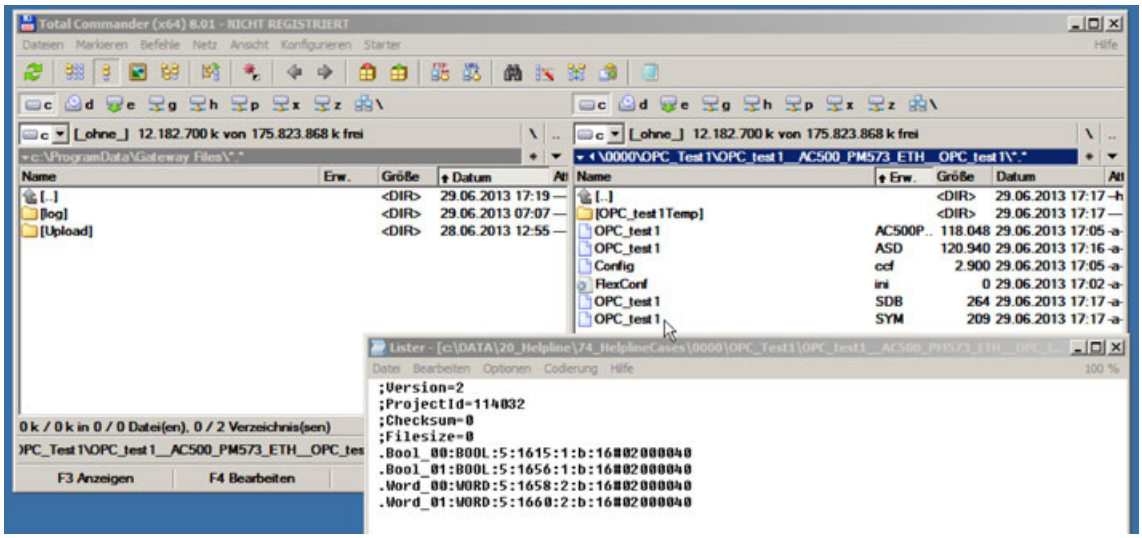


6. Select the variables which should be communicated as symbol.

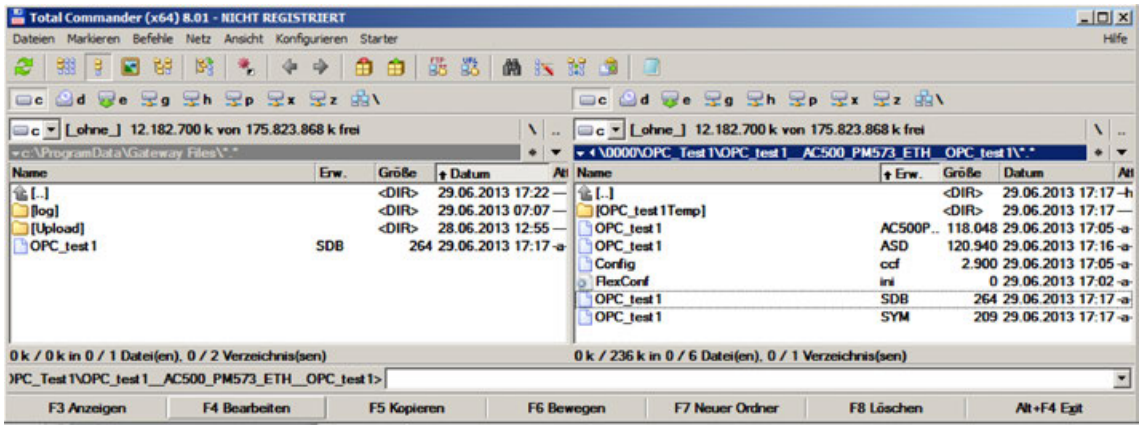
   Enable the following checkboxes:

   - *Export variables of object*
   - *Export structure components*
   - *Export array entries*
   - *Write access*

7. Confirm twice with *[OK]*.
8. Under *"Project ➔ Rebuild all"* rebuild the project.

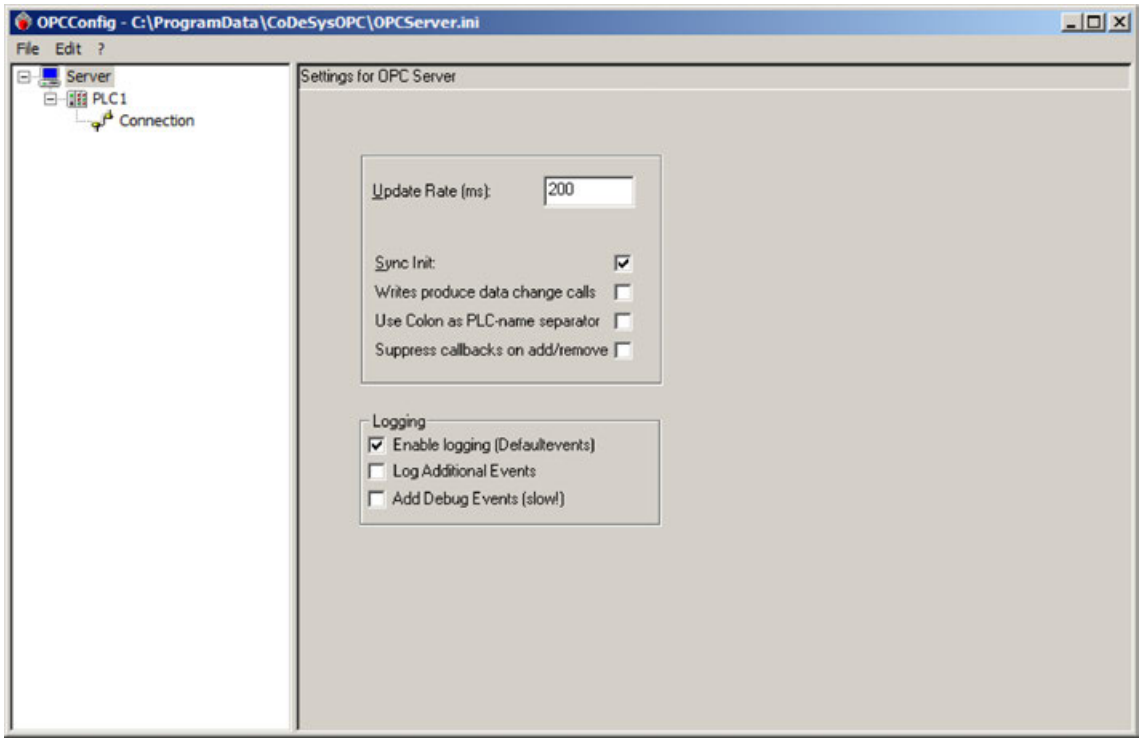


9. In the project folder is the subfolder *"OPC_test1__AC500_PM573_ETH__OPC_test1"*. It contains symbol files **.SYM* and **.SDB* with the time of the *"Rebuild all"*. The items in the file **.SYM* can be checked with Notepad. The binary file **.SDB* contains the items for the OPC server. With *<Online> <Login>* it will copied in the gateway files directory and optionally on the AC500.

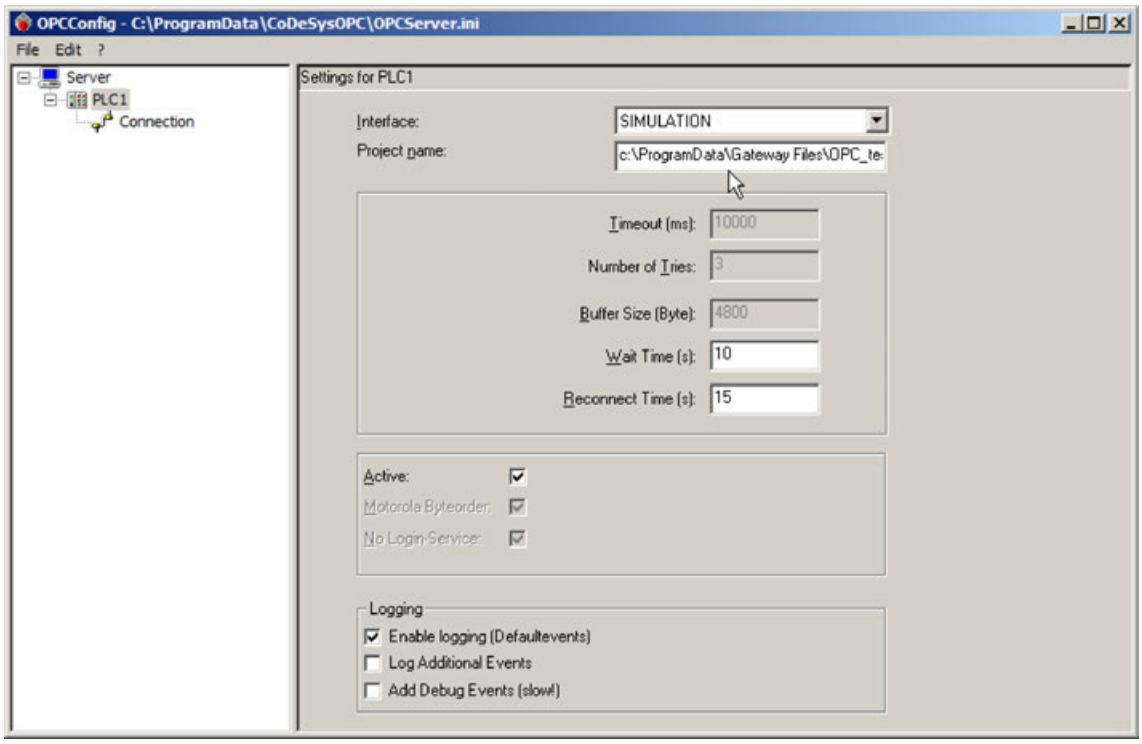


10. The folder *"OPC_test1__AC500_PM573_ETH__OPC_test1"* is a temporary folder, if the CBP project is opened. For the simulation of the server OPC it is copied **.SDB* by hand.

Configure OPC server V3

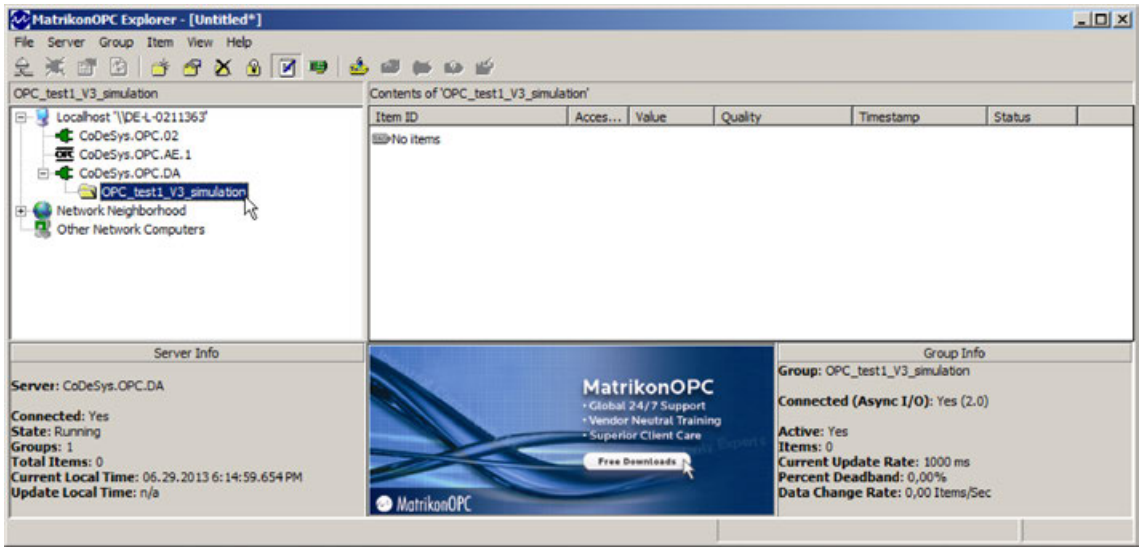


1. Select **Edit**, append **PLC** and keep the default values.

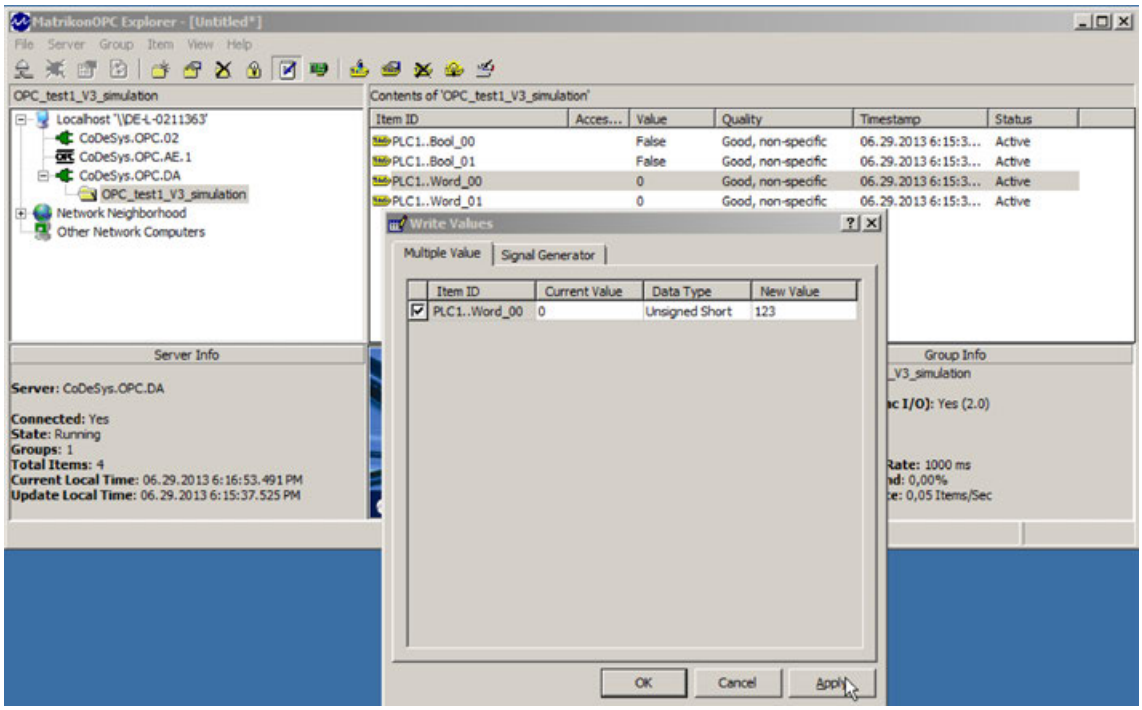


2. You must specify *“Project name”* with the *“directory name”*.
   Connection settings are not required for the simulation.

Check OPC server with MatrikonOPCExplorer

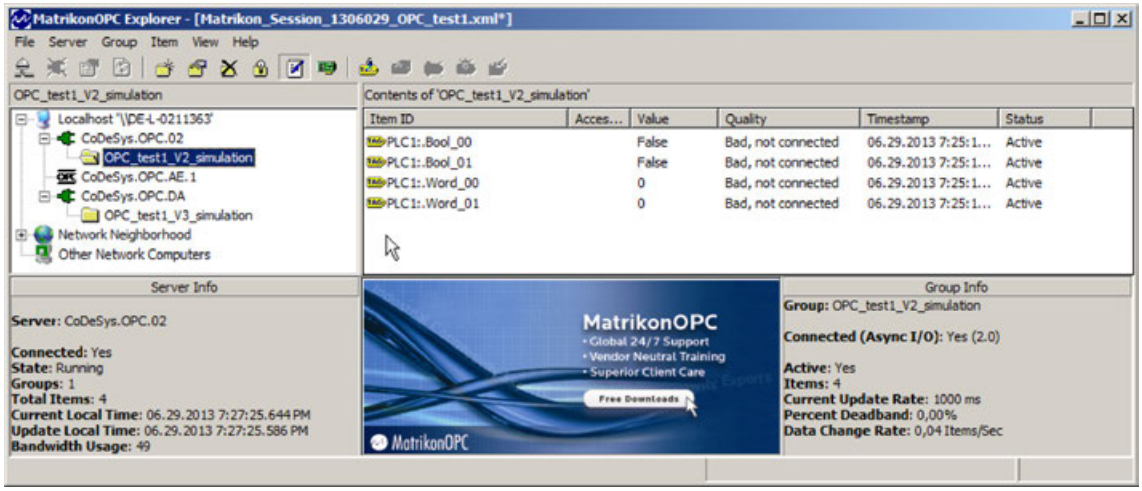


1. OPC Server V3: *"Connect CoDeSys.OPC.DA"*. Add *"Group"*, add *"Items"*, select *"Availabe Tags"* and add to *"Tag List"*.



2. The OPC Server V3 (*"CoDeSys.OPC.DA"*) is connected, running and the *"Quality"* is good.

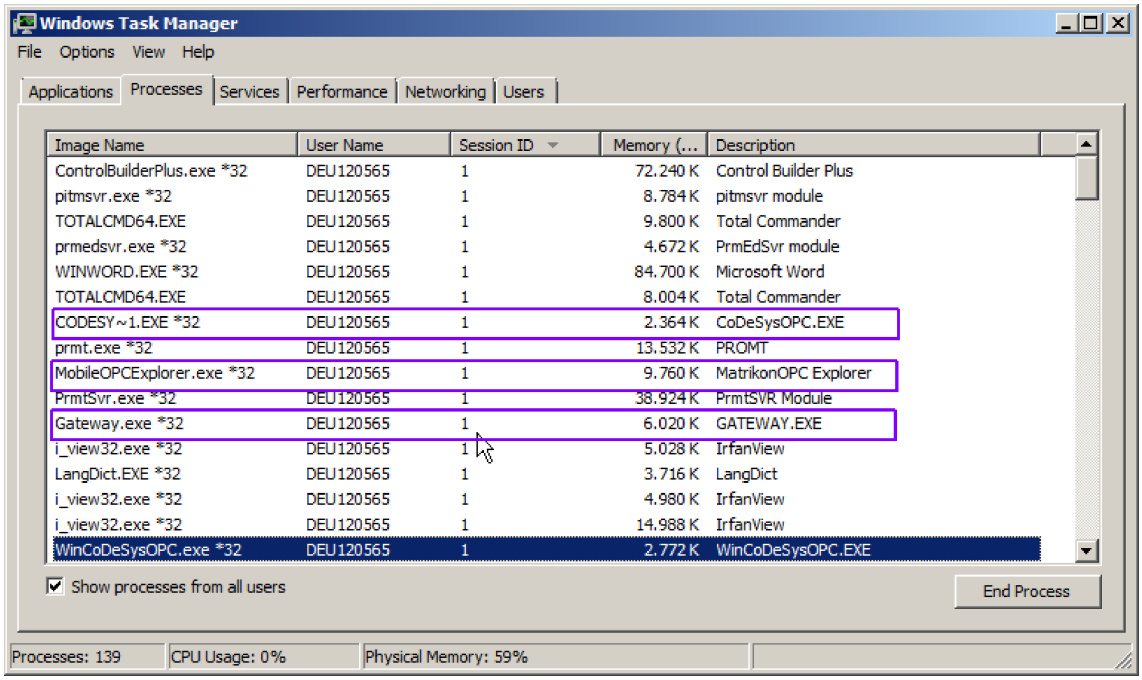
   One OPC client can read / write the values of the items.



3. Similar configuration as above.

   The OPC Server V2 (*"CoDeSys.OPC.02"*) is connected, running and the configured items are found. But the *"Quality"* is bad. One OPC client can not read / write the values of the items.

Check processes with windows task manager



Correct configuration: All *"Processes"* run with the same *"User Name"* and with the same *"Session ID"*.

Summary

The correct function of OPC Server V2 and V3 can be checked without AC500.

With OPC Server V3 with the configuration "SIMULATION" the Project name with the directory name has to be specified. The values of the items can be read and write by one OPC client.

With OPC Server V2, as well as with OPC Server V3 in configuration "GATEWAY", only the project name may be specified. The configured items are found, but the quality is bad. The values of the items can not be read and not write by one OPC client.

Refer to REF5 Online Help of PS501 chapter OPC for details ↪ Table on page 3953.

1.6.6.5.2 OPC UA server for AC500 V3 products

General

OPC UA server can be added as an object below the Ethernet interfaces ETH1 or ETH2.

The user can access the variable interface of the PLC via a client. At the same time, communication can be protected by means of encryption.

The CODESYS OPC UA server supports the following features:

- Browsing of data types and variables
- Standard read/write services
- Notification for value changes: subscription and monitored item services
- Encrypted communication according to "OPC UA standard (profile: Basic256SHA256)"
- Imaging of the IEC application according to "OPC UA Information Model for IEC 61131-3"
- Supported profile: Micro Embedded Device server Profile
- By default, there is no restriction in the number of sessions, monitored items, and subscriptions. The number depends on the performance of the respective platform.
- Sending of events according to the OPC UA standard.

Application example

The application example How to use OPC server V3 - for DA and UA is available to gain a deeper understanding of the OPC UA protocol and to configure AC500 V3 accordingly.

Creating a project for OPC UA access

1. Click *"File ➔ New Project ➔ AC500 project"* in Automation Builder 2.1 or newer.
2. Choose a *PLC - AC500 V3* and click *[Add object]*.
3. Right-click on node *ETH1 or ETH2* and *"Add object"*.
4. Choose *OPC UA Server* in the dialog and click *[Add object]*.
5. Declare some variables of different types in the program.
6. Right-click *"Application ➔ Add object"*. Choose *Symbol configuration* and click *[Add object]*.
7. Enable checkbox *Support OPC UA Features* in the dialog *Add symbol configuration*.
8. Double-click *"Symbol configuration"* in the *Devices* tree to open the editor *Symbol configuration*.

9. Click *[Build]*.

   ⇨ The variables are displayed in a tree structure.

10. Activate the variables that you want to publish to an *OPC UA client*. Specify the access rights.
11. Download the project to the PLC.

Use node name

1. Double-click node *“OPC_UA_Server”*.
2. Set parameter *Use node name* to TRUE.
3. Double-click node *“PLC_AC500_V3 <...>”*.
4. Click *“Device”* and *“Rename active device...”*
5. Enter new device name in the following dialog and click *[OK]*.

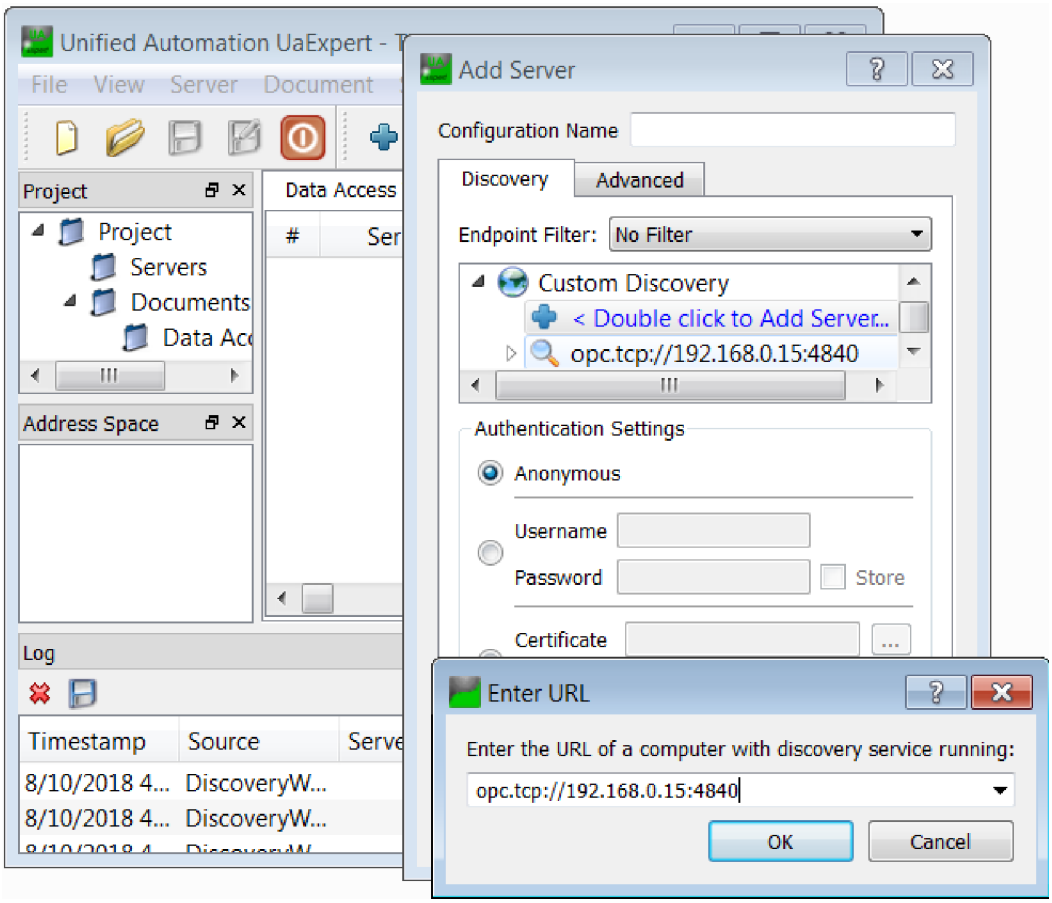
Use UaExpert client

The OPC UA client *UaExpert* is available for download from the Unified Automation website and can be used free of charge (freeware license).

Using this client, you can connect to the AC500 OPC UA server.

The following description refers to this program. Other OPC UA clients work in a similar way.

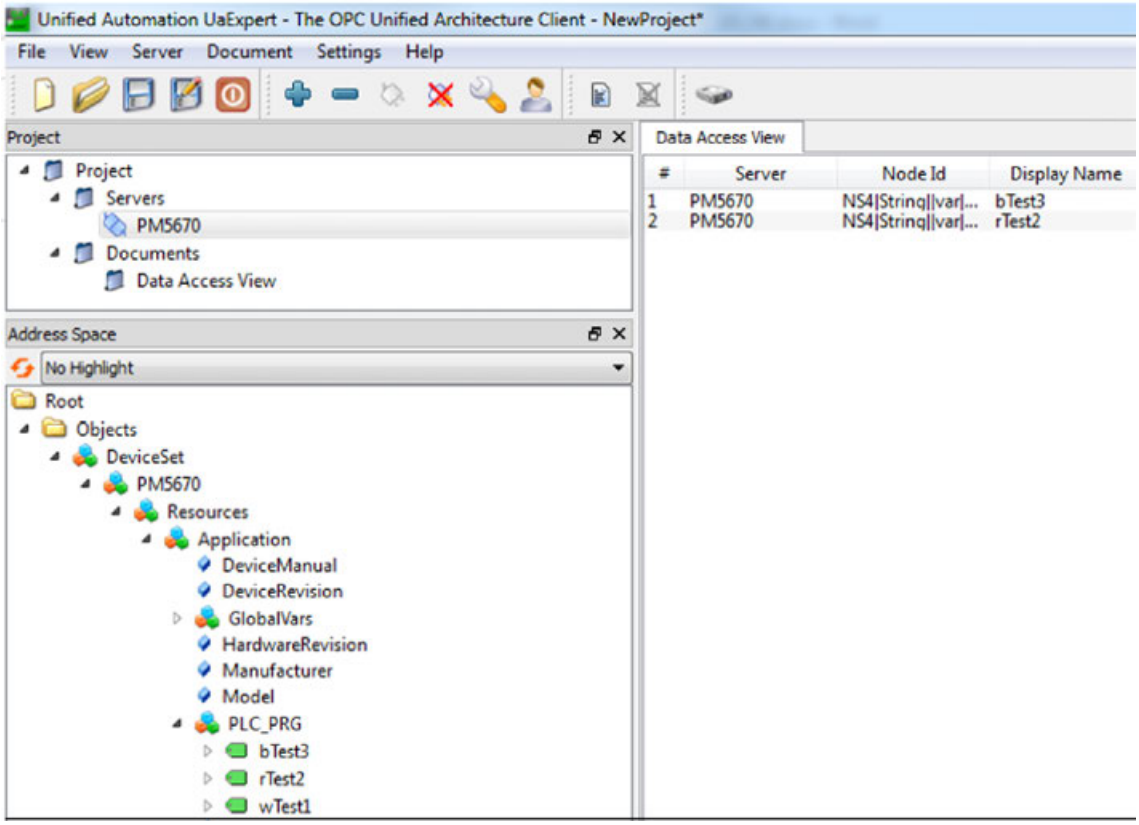
1. Start the *UaExpert* program.



2. Click on the *“blue cross symbol”*.
3. Double-click on the *“blue cross symbol”* in the *Add Server* dialog.

4. Enter URL and click *[OK]*.
   ⇨ The URL appears in the *Add Server* dialog.
5. Select *"Advanced"* tab and click *[OK]*.
6. Click *[Connect]* button.

7. Expand the project tree in the *Address Space* window.



8. Drag and drop the needed symbols to *Data Access View*.

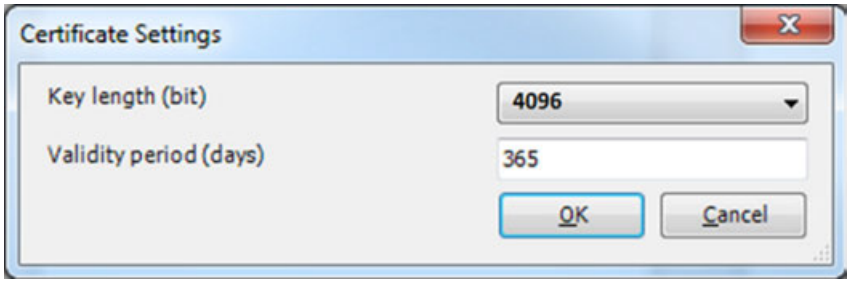
Working with encryption

Creating a certificate for the OPC UA server

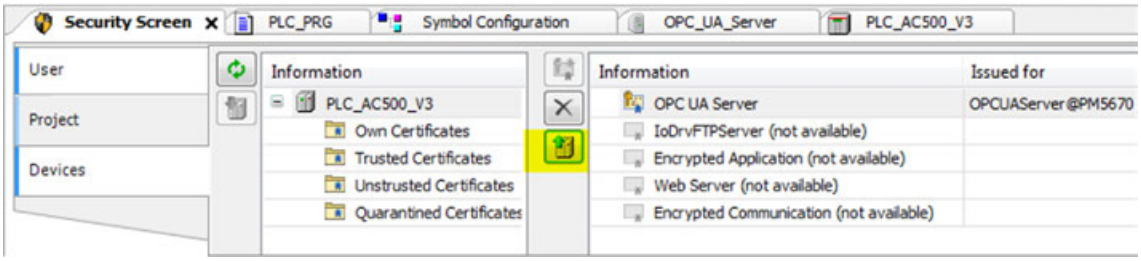
☑ Prerequisite: A battery is inserted and the clock is set to actual time.

1. Double-click the Security symbol in the lower right corner of Automation Builder.
2. Select the *"Devices"* tab.
   ⇨ The certificate information opens.
3. Select the PLC in the left *Information* view.
   ⇨ All services of the PLC that require a certificate are displayed in the right *Information* view.
4. Select the service *"OPC UA Server"*.

5. Click the icon to create a new certificate for the device.
   ⇨ *Certificate Settings* dialog appears.



6. Define the certificate parameters according the figure above and click "*[OK]*.
   ⇨ The certificate is created on the PLC.

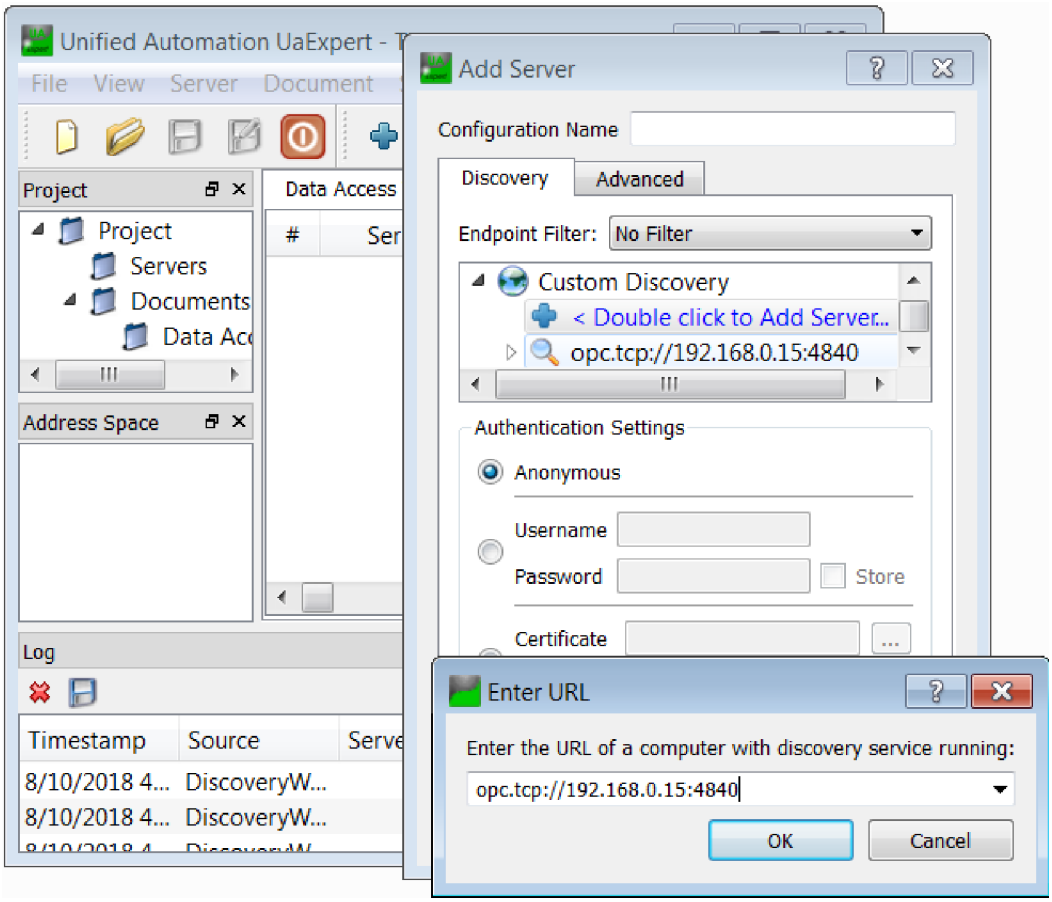


7. Upload the certificate to your PC.
8. Restart the runtime system.

For further information see ↳ *Chapter 1.6.6.3.7.3.4 "OPC UA secure" on page 3923*.

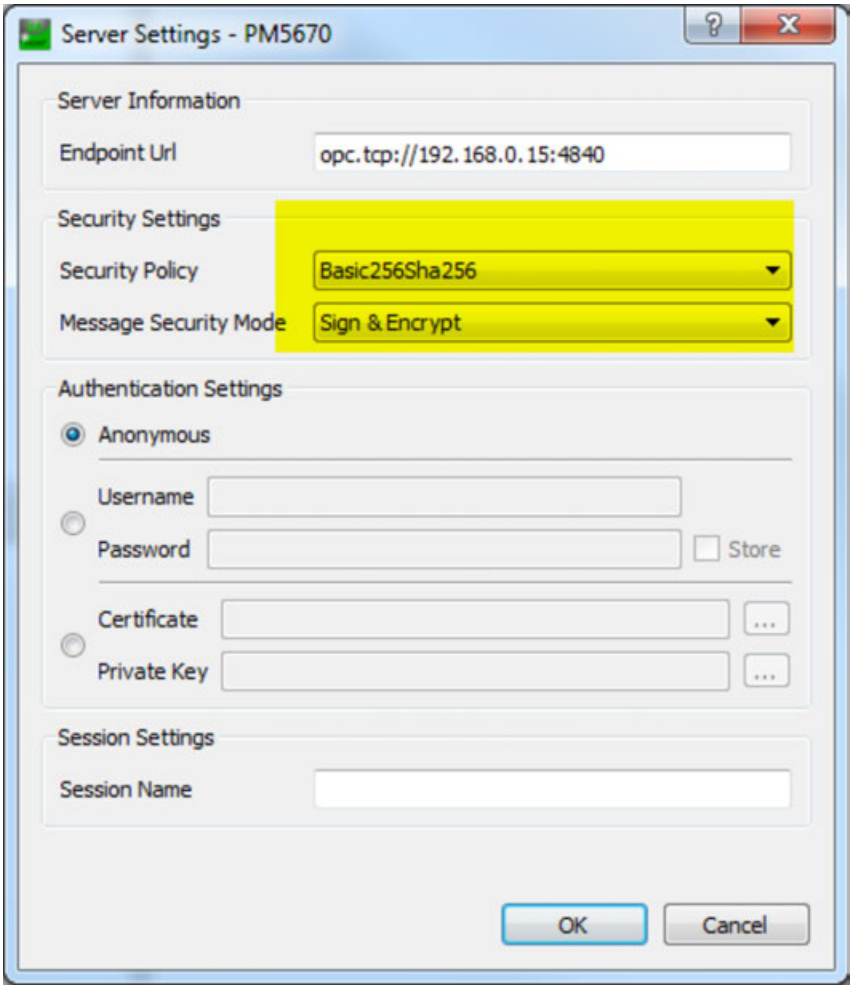
Encrypted connection with UaExpert client

1. Start the *UaExpert* program.

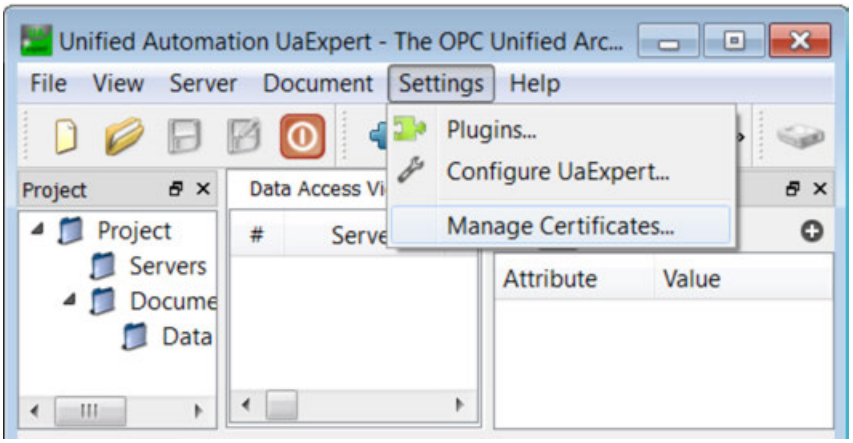


2. Click on the *“blue cross symbol”*.
3. Double-click on the *“blue cross symbol”* in the *Add Server* dialog.
4. Enter URL and click *[OK]*.
   ⇨ The URL appears in the *Add Server* dialog.

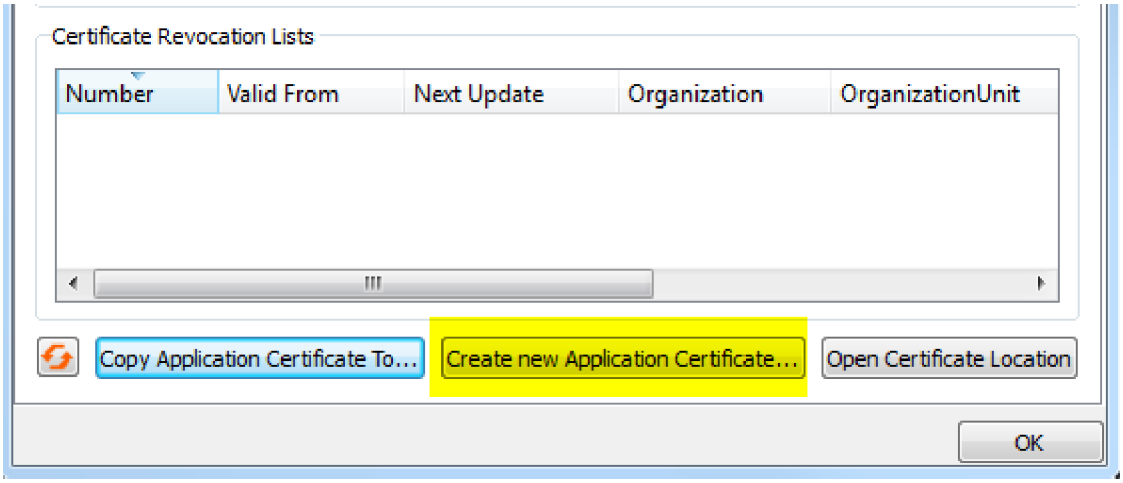
5. Select *"Advanced"* tab.



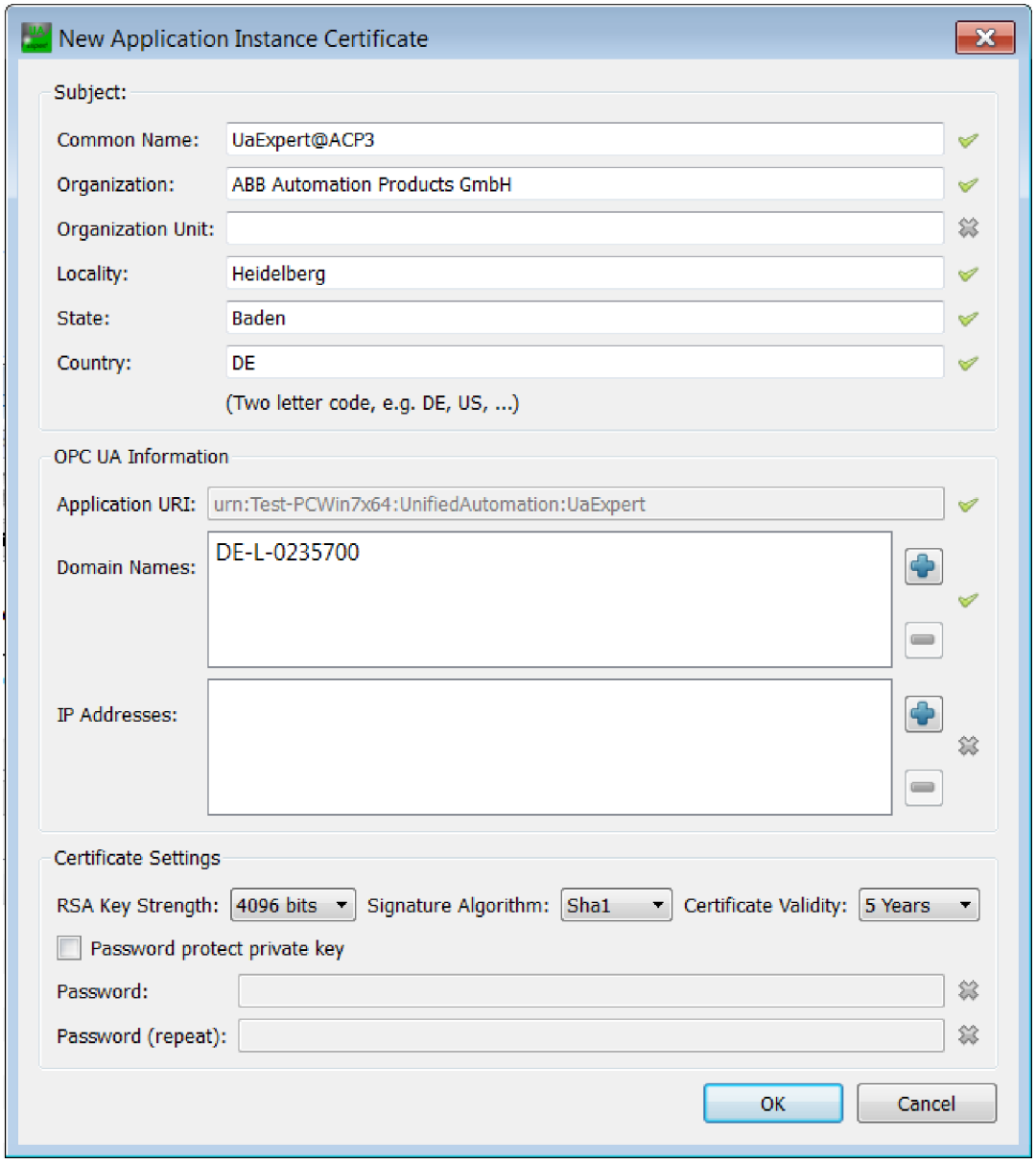
6. Choose option *"Basic256ha256"* of drop-down list *Security Policy* and *"Sign & Encrypt"* of drop-down list *Message Security Mode* and click *[OK]*.



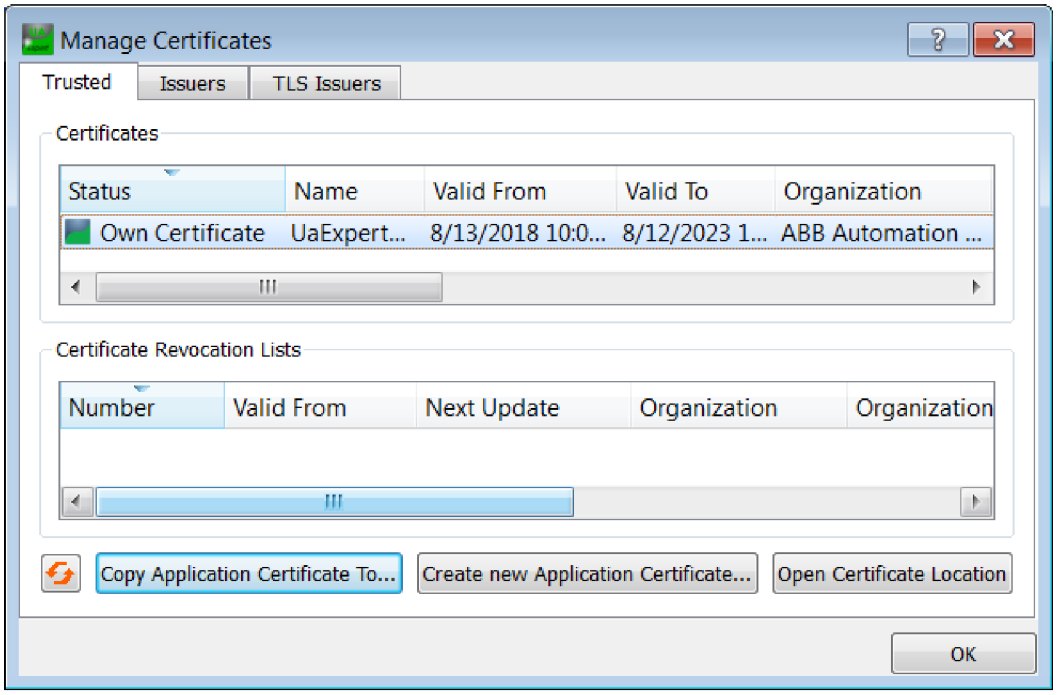
7. Click menu *“Settings”* and *“Manage Certificates”*



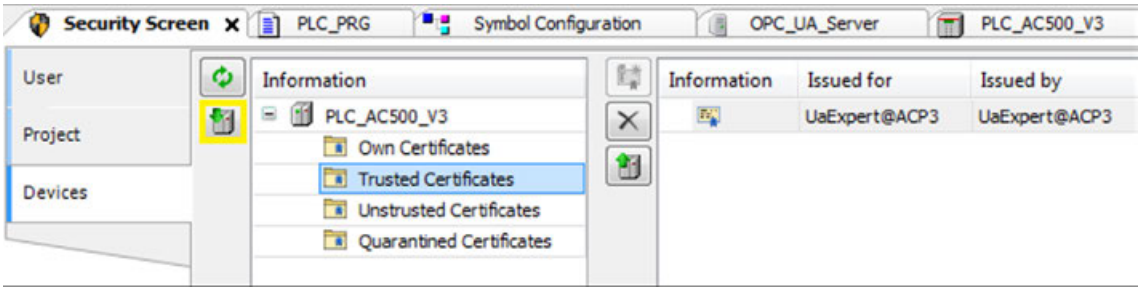
8. Click *[Create new Application Certificate...]*.
   ⇨ Dialog *New Application Instance Certificate* opens.



9. Enter the required informations and click *[OK]*.
   ⇨ Dialog *"Manage Certificates"* opens

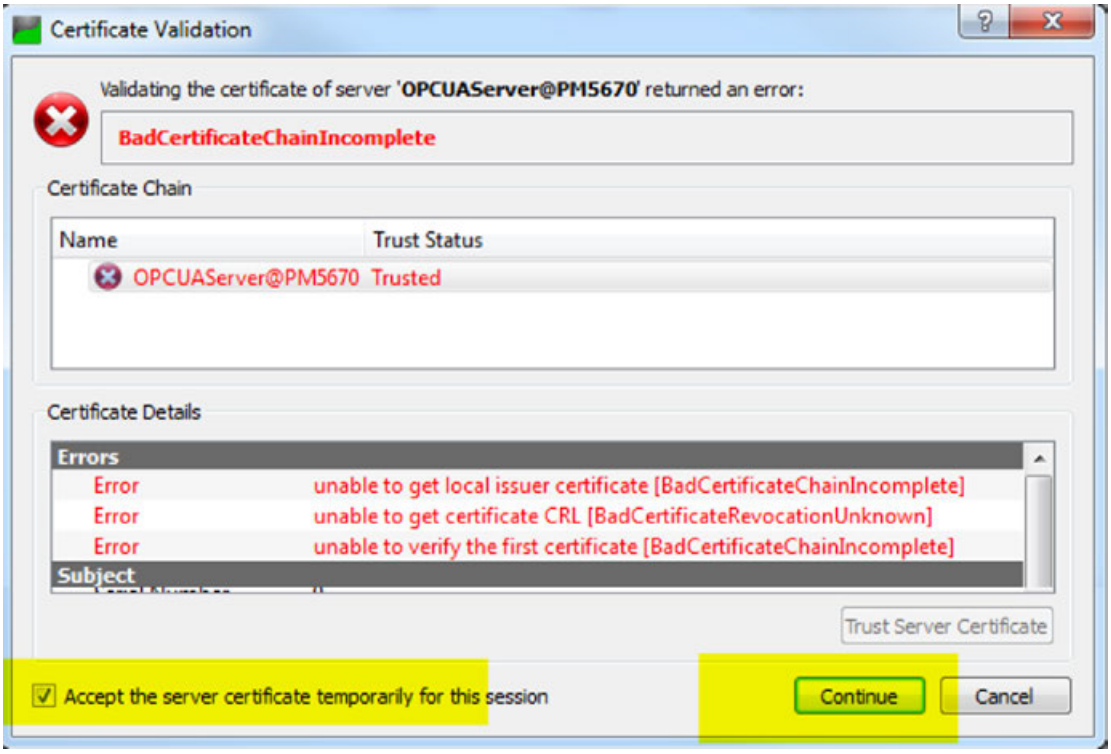


10. Click *[Copy Application Certificate To...]* your PC.



11. Download the certificate to AC500 via the *Security Screen* view.
12. Click *[Connect]* button in the UaExpert client.

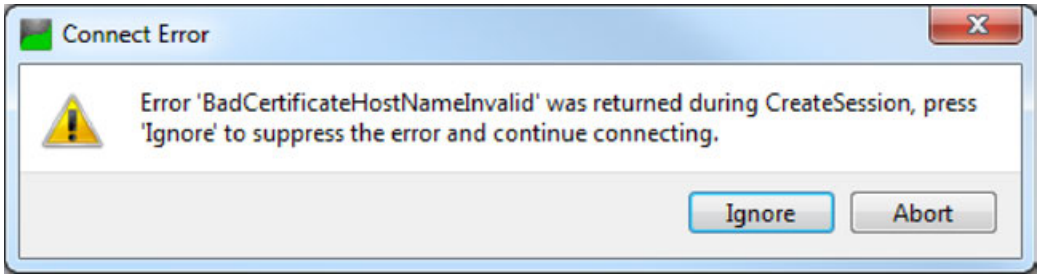
⇨ Dialog *Certificate Validation* opens.



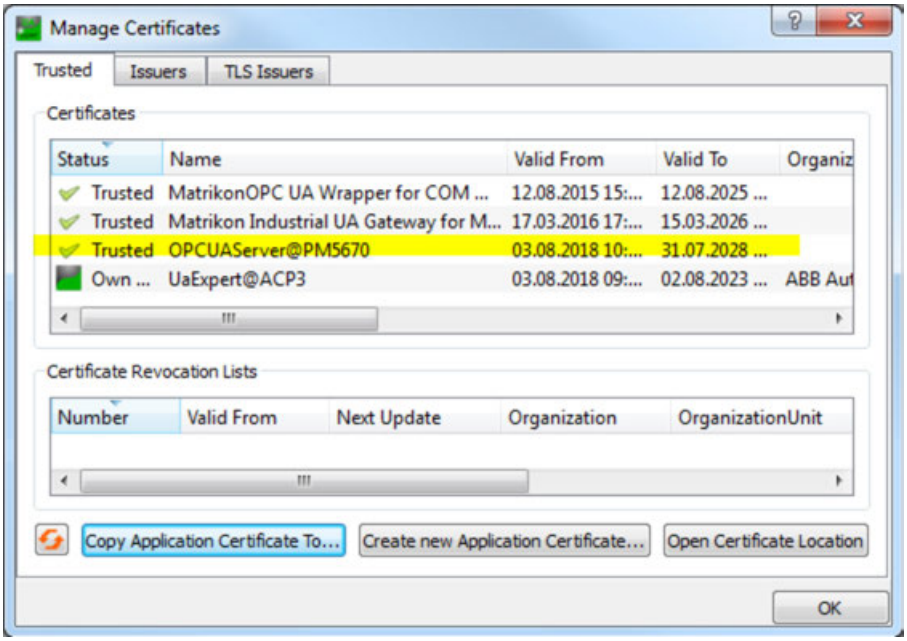
Working with a trusted certificate will avoid this error message.

14. Enable checkbox *Accept the server certificate temporarily for this session* and click *[Continue]*.

⇨ Dialog *Connect Error* opens



15. Click *[Ignore]*

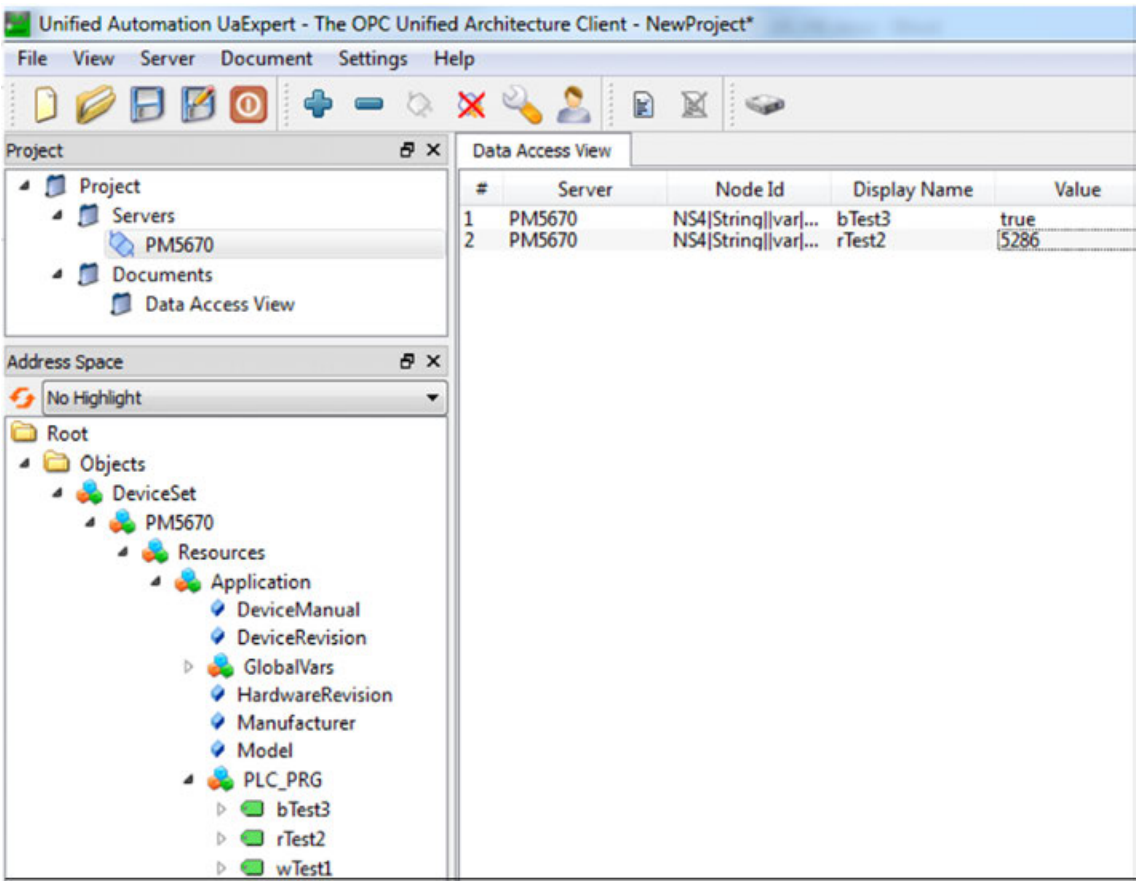


16. Check settings in dialog *Manage Certificates*.

Changing variables via UaExpert client

1. Expand in view *Address Space "Objects ➔ DeviceSet ➔ PM5670 ➔ Resources ➔ Application ➔ PLC_PRG"*.

   ⇨ The variables of the global variable list are visible.



2. Drag and drop the variables to the Data Access View.
3. Change values in the column *Value*.

Configuring OPC UA client

Operating modes

Polling

- Objects will be continuously updated in a defined interval
- Create higher load then Subscription
- Is recommended only for a few Symbols

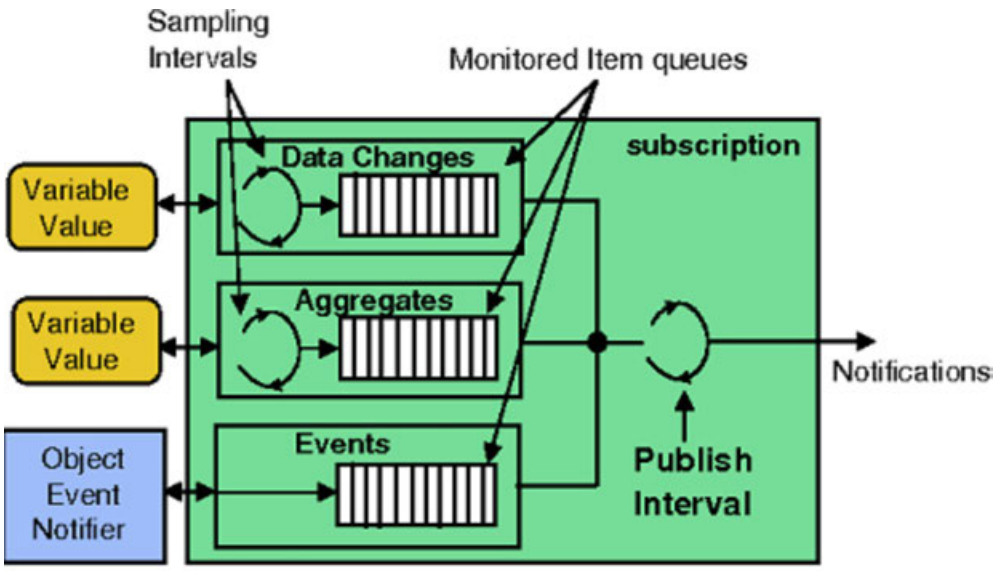
Pub/Sub

Not yet supported

Subscription (recommended mode)

- Updated objects depending on the publishing interval and filters
- Method to reduce load
- Different intervals
- Filter possible (coming in AC500)

Client defines a group of symbols with	Description
Publishing interval	Interval, in which server publish data to client
Sampling interval	Interval for sampling and storing data at server and send in each publishing interval
Queue size	Array of data to save data if sampling Interval is faster than publishing Interval (At AC500 in the moment only 1)
Data change filter	Can be used to reduce traffic from server to client. Criteria: ● Change of data, ● Change of status ● Change of time stamp AC500 is fix configured for change of data and change of status.

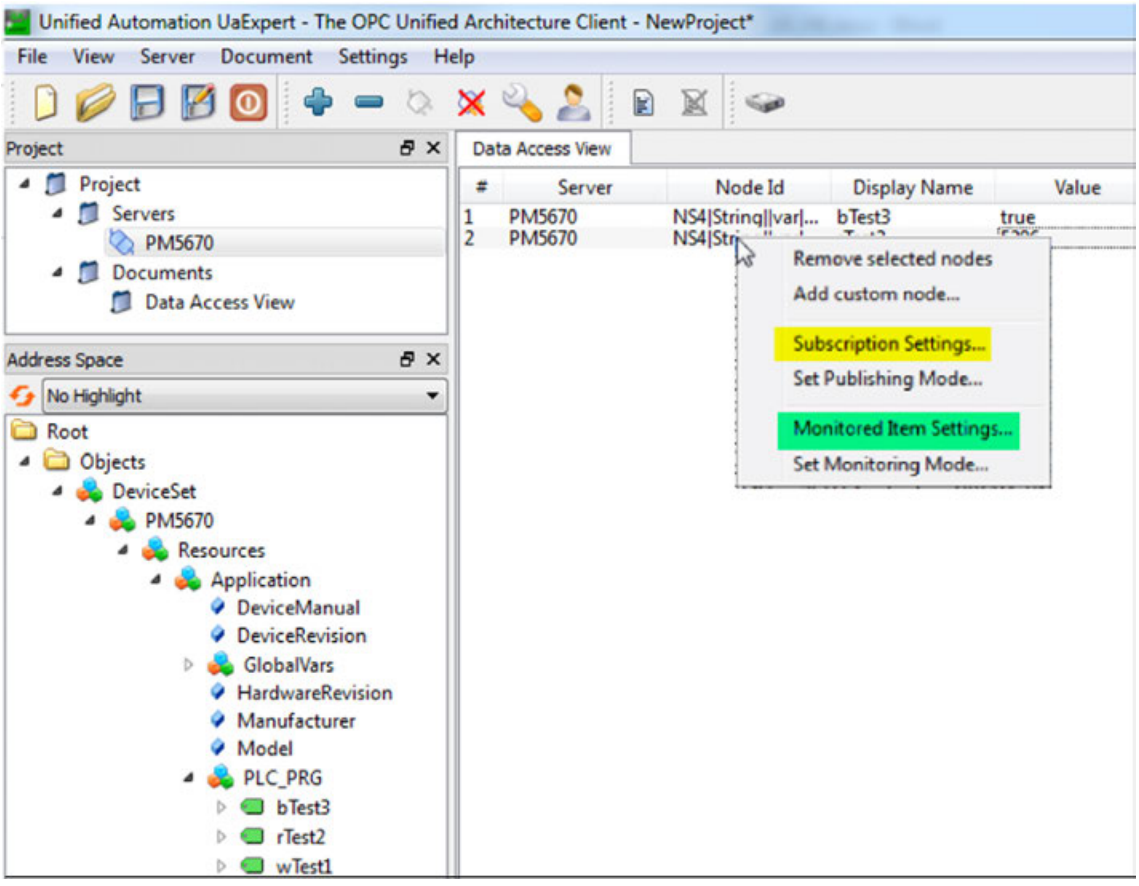


Using OPC UA with subscription mode

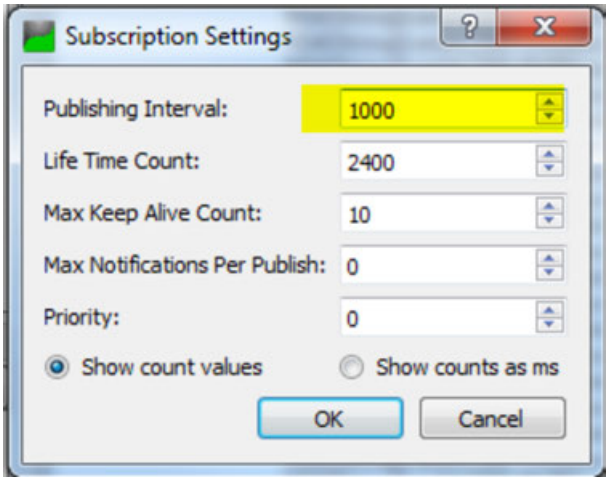
Recommendations:

- *Define only variables you need as symbols*
- *Do not configure publishing Intervals to short (increase load)*
- *Use different subscriptions with different publishing intervals in order to decrease load*
- *Do not use sampling intervals faster then publishing intervals as long as AC500 OPC UA server don‘t support Queue Size different from 1*
- *Be careful: Setting „0“ at sampling Interval at client will be interpreted in server as „as fast as possible“, which is 100ms at AC500 and create a high load.*

Publishing and sampling intervals in UaExpert



1. Right-Click on an Item in *Data Access View* and click *"Subscription Settings"*.



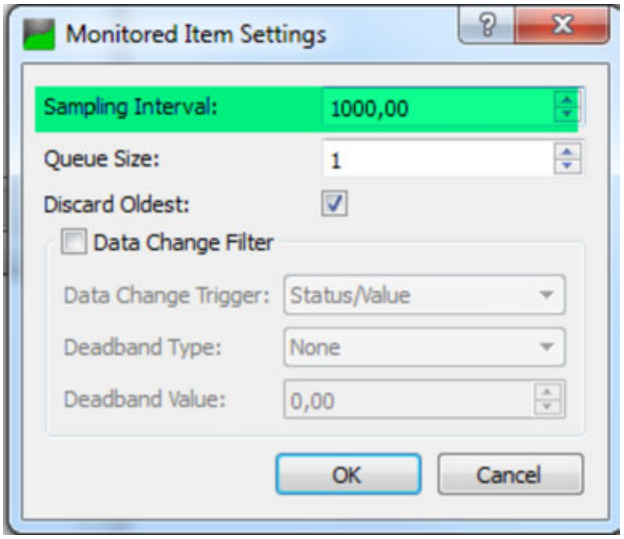
2. Set the recommended values.

   Life Time Count: Number of publishing intervals in which client has to send publish requests to the server. After this period without request from client, subscription in server will be deletet.

   Max Keep Alive Count: If there are no new data to send, server can skip a publishing interval. After the alive count, server has to send, even if there are no new data.

   Click *[OK]*.

3. Right-Click on an Item in *Data Access View* and click *"Monitored Item Settings"*.



4. Set the recommended values.

1.6.6.5.3 Web server

In order to be able to use the PLC as a client for web services, the HTTP function block library can be used. Setup and use are described in an *application example*.

1.6.6.6 Converting an AC500 V2 project to an AC500 V3 project

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

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

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

1.6.7 Storage devices for AC500 V3 products

1.6.7.1 Introduction of AC500 storage devices for AC500 Products

1.6.7.1.1 Overview

AC500 PLCs offer a variety of storage devices. The following table gives a short overview and a description on these storage devices:

IEC access means that the storage device can be accessed by function blocks of an IEC program.

FTP access means that the device can be accessed via FTP server on the PLC (if available).

Component	Description	IEC access	FTP access	CPUs AC500 V3
userdisk home/userdisk (customer data)	User disk for custom data (flash) Internal persistent mass storage placed in the internal flash device Can be used for any application purpose	Yes	Yes	All
PLCLogic home/PLCLogic (customer data)	Internal persistent mass storage placed in the internal flash device Used for configuration data, user application (boot project), WebVisu files, etc.	Yes	Yes	All
SRAM	Battery-buffered device, non-volatile RAM Used for retain/ persistent and ProzM variables	Yes	No	All
system	System RAM disk (Temp directory) for storing the firmware For internal firmware use only!	Yes	No	All

Component	Description	IEC access	FTP access	CPUs AC500 V3
flashdisk	Internal persistent mass storage device Can be used for any application purpose	Yes	Yes	PM5675-2ETH
memory card	memory card (removable) Removable persistent mass storage device Can be used for any application purpose	Yes	Yes	All

1.6.7.1.2 Functionalities

Filesystem Name	As of CPU firmware	Description
userdisk	V3.0.0	Boot project (size deends on PLC type) WebVisu files for web server Symbol file for OPC server and CP600 panels User data via CAA_File_xxx.lib *) Files via Automation Builder file download Files via FTP server
	V3.1.0	Save persistent data
SRAM	V3.1.0	Save retain and persistent data
system	V3.0.0	Load / save boot project
		Firmware update
		Internal system files
flashdisk	V3.1.0	User data via CAA_File_xxx.lib *) Files via Automation Builder file download Files via FTP server
sdcard	V3.0.0	Firmware update, User data via CAA_File_xxx.lib *) Files via Automation Builder file download Files via FTP server
	V3.1.0	Save persistent data Boot project (size depends on PLC type)

*) Examples for the filename with path (sFileName for FILE.Open) specified by the user ('mydir' is optional, but must be an existing directory):

- 'userdisk/myfile.txt'
- 'sdcard/mydir/myfile.txt'
- 'flashdisk/myfile.txt'

The maximum number of files opened at the same time is limited to 1007.

The max. length of the user string (path and filename) is 241 characters.

Unlike the PLC's memory areas like %M or Retain, where 1 byte actually consumes 1 byte, all storage device utilize a file system.

That means there is a difference between a files size and its size on the disk.

On disks the files are stored in so-called clusters which are a group of disk sectors. "Size on disk" refers to the amount of cluster(s) a file is taking up, while "file size" is an actual byte count of the file data. So you will usually find that the size on disk is larger than the file size. This is not an error, but a result of the disk organization via a file system. Since sector and cluster sizes vary depending on a disk's size and the used file system, the ratios between the size on disk and the file size also vary between the various storage devices.

1.6.7.1.3 Memory sizes

AC500-eCo V3 processor modules

PLC type	system RAM disk	userdisk PlcLogic ...	Retain, ProzM area	flash disk	memory card
PM5012-x-ETH	Dynamically /max. 7.6 MB	30 MB	8 kB Retain and persistent 4 kB (of which 88 byte are reserved for allocation table) ProzM 4 kB	None	see *Chapter 1.6.4.6.5.2 “MC5102 - Micro memory card with micro memory card adapter” on page 3432*
PM5032-x-ETH			32 kB Retain and persistent 16 kB (of which 88 byte are reserved for allocation table) ProzM 16 kB		
PM5052-x-ETH					
PM5072-T-2ETH(W)			100 kB Retain and persistent 36 kB (of which 88 byte are reserved for allocation table) ProzM 64 kB		

AC500 V3 processor modules

PLC type	system RAM disk	userdisk PlcLogic ...	SRAM Retain, ProzM area	flash disk	memory card
PM5630-2ETH	Dynamically /max. 7.6 MB	40 MB 30 MB (as of V3.4.0)	256 kB Retain and persistent 128 kB (of which 24 byte are reserved for allocation table) ProzM 128 kB	None	see *Chapter 1.6.4.6.5.1 "MC502 - Memory card" on page 3428* *Chapter 1.6.4.6.5.3 "MC5141 - Memory card" on page 3437* *Chapter 1.6.4.6.5.2 "MC5102 - Micro memory card with micro memory card adapter" on page 3432*
PM5650-2ETH	Dynamically /max. 16 MB	246 MB (as of V3.0.x) 381 MB (as of V3.1) 285.75 (as of V3.4.0)			
PM5670-2ETH	Dynamically /max. 69 MB	858 MB 643.50 MB (as of V3.4.0)	1536 MB 1 MB retain and persistent (of which 24 byte are reserved for allocation table) 512 kB ProzM		
PM5675-2ETH				8 GB	

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

1.6.7.1.4 Storage device details

This section contains some details on each storage device. For further details on specific topics please also refer to the following chapters:

Storage device sizes	*Chapter 1.6.7.1.3 "Memory sizes" on page 3996*
FTP access	*Chapter 1.6.6.3.5 "FTP server" on page 3917*
PLC shell commands	*Chapter 1.6.6.4.4 "PLC shell commands" on page 3950*

SRAM

The SRAM is a battery-buffered, nonvolatile RAM and is used for the retain/persistent and the ProzM variables. If a battery is inserted into the processor module, the data stored in the SRAM will not get lost during a power-down cycle.

During PLC startup, the SRAM will be deleted automatically if no or an empty battery is inserted into the processor module. In this case the information

```
ABBInitSram_SetupMemory : SRAM cleared
```

and the warning

```
Retain size in config changed, or retain area got corrupted
```

are written into the log file.

Further information see *Chapter 1.6.5.1.1 “Handling of remanent variables for AC500 V3 products” on page 3456*.

Memory card

The memory card is a removable persistent mass storage device and can be used for any application purpose. Both firmware updates and boot project updates can be run from the memory card *Chapter 1.6.7.2 “Memory card in AC500 V3” on page 3999*.

Size	Product specific, see table Memory Sizes *Chapter 1.6.7.1.3 “Memory sizes” on page 3996*

Flash disk

The flash disk is an internal persistent mass storage device and can be used for any application purpose.

It has a memory capacity of 8 GB (preformatted).

The flash disk is capable of high data throughput, however, the actual values to be achieved depend on the use cases. If the performance seems to get insufficient, check the following:

- If the PLCs CPU load is high, reduce overall CPU load of the PLC to have more performance for file operations.
- If the device has low free space, cleanup the disk.
  Please consider the cluster size of 4 kB in your application design to achieve optimal usage of the flash disks space and access performance. For example, 10 files with 10 byte each require 10*4 kB disk space, while 1 file with 100 byte requires only 4 kB.

Number of max. write cycles

Technically, the flash chip used in V3 flash disk has 20000 Erase-Cycles (Write cycles).

Due to the produced write overhead, the optimum achievable number of write cycles is 10000 (for typical payload sizes of 256 kB).

Example

The write overhead is indicated by the *write amplification factor* (WAF).

$$\mathrm{WAF} = \frac{\text{Flash Write (in Bytes)}}{\text{Host Write (in Bytes)}}$$

Table 746: Rule of thumb for assessing the flash lifetime for an application:

Typical payload sizes	WAF	Max. write cycles
256 kB	2	10000
128 kB	4	5000
64 kB	8	2500
...	...	...

Typical payload sizes	WAF	Max. write cycles
1024 Byte	512	< 40
512 Byte	1024	< 20

For monitoring the status

It is recommended to use the respective function blocks to monitor the status of the flash disk (see Chapter 1.6.7.4 “Health monitoring” on page 4010).

Since FW version 3.3.0, there is also a diagnosis event supported when the user flash memory reaches the end of its life cycle.

Lifetime of flash disk will also depend on the operating environment.

E.g. high ambient temperatures will impose stress on the user flash memory and reduce the total overwrites achievable.

- Max. write speed is 20 MB/s (continuous write of sequential data)
- Read cycles are unlimited.

1.6.7.2 Memory card in AC500 V3

The memory card is a removable persistent mass storage device and can be used for any application purpose. Both firmware updates and boot project updates can be run from the memory card.

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working. For memory card activity the black square (■) is shown on PLC display as long as a file is open on the memory card. Remove the memory card only when no black square (■) is shown next to memory card in the display. Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1.6.7.2.1 Firmware and/or application update with memory card

ABB recommends that users carry out the firmware update via Automation Builder. Chapter 1.6.6.1.4 “Firmware identification and update” on page 3652

Not every user has an Automation Builder at his disposal with which a firmware or boot project update can be easily realized. In this case, the user must be provided with a prepared memory card from his client.

It is not possible to update the communication interface modules with a memory card. The firmware of the communication interface modules can only be updated with the IP configuration tool. Chapter 1.6.6.2.9.1.2.3 “Firmware update” on page 3728

Preparation of memory card

Memory cards contain firmware and application. Visualizations and all related objects (like text lists) are also added to the memory card.

The command to create boot project and/or firmware files is available in the context menu of AC500 application nodes *"Export"* in the device tree. In both cases a folder location can be chosen by the user to which the content shall be exported (any location is fine: file system, memory card, etc.). The created folders can also later be copied to a memory card.

For more information, see also

- MC502 - memory card *Chapter 1.6.4.6.5.1 "MC502 - Memory card" on page 3428*
- MC5141 - memory card *Chapter 1.6.4.6.5.3 "MC5141 - Memory card" on page 3437*
- MC5102 - micro memory card with micro memory card adapter *Chapter 1.6.4.6.5.2 "MC5102 - Micro memory card with micro memory card adapter" on page 3432*

Export boot project and firmware

When selecting *"Boot project and firmware (SD card)"* the boot project is additionally exported to the given file location. If not yet existing it is created automatically for V3. On V2 it has to be created before executing the export command. A corresponding error message is then shown with instructions.

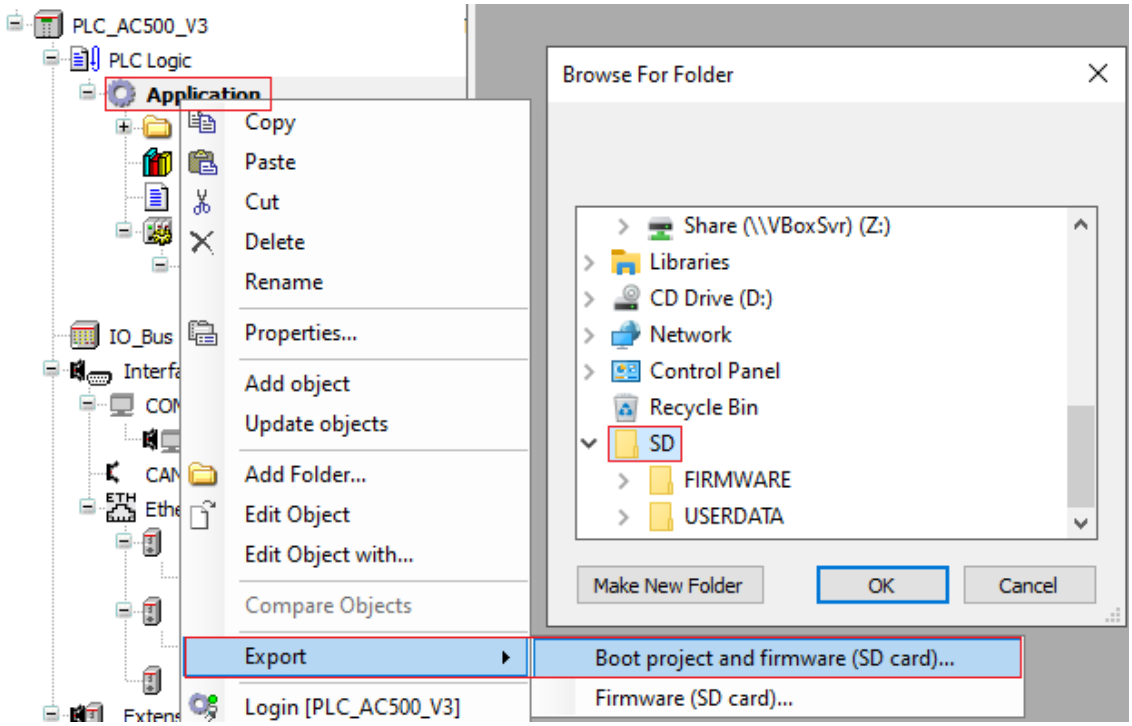
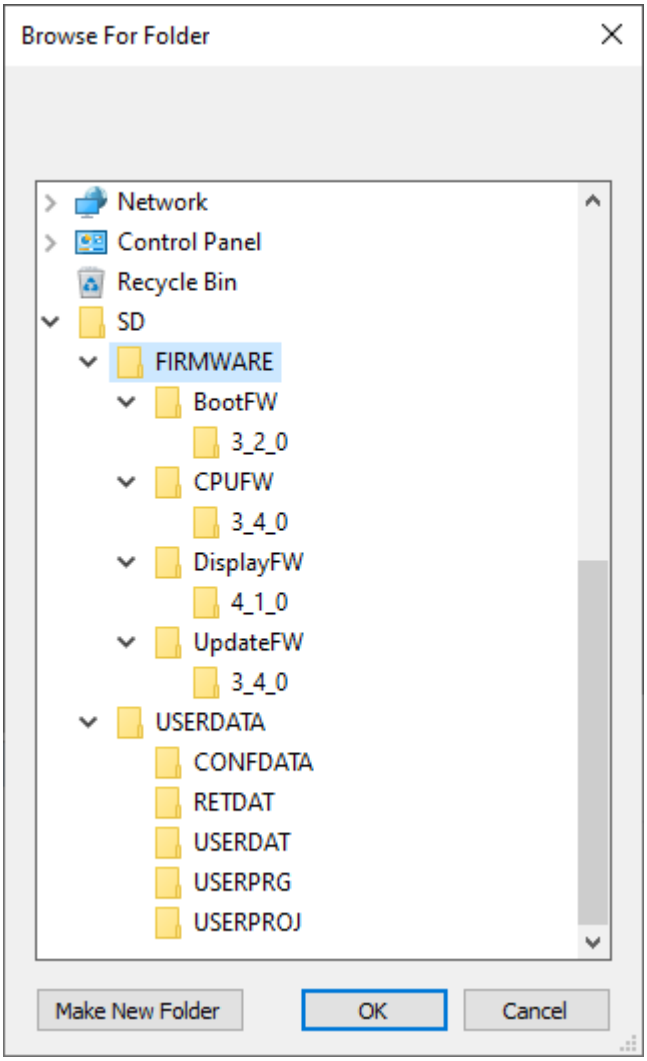


Fig. 342: Example for AC500 V3

1. Right-click *"Application"* in the device tree.
2. Select *"Export ➔ Boot project and firmware (SD card)..."* .
3. Click *[Make New Folder]* and type in "SD".
4. Select *[OK]* to add the folder.

5. Select folder *"SD"*



⇨ The SD structure has been created and the firmware and application have been exported.

Mark all subfolders and files of the SD folder and copy them to a memory card. Do not copy the SD folder, only the subfolders and files!

> *The created SD folder does not contain user data, remanent data, config data, safety PLC power dip data and safety PLC password!*
>
> *Please add this data if required by the used application.*

Export firmware (only)

When selecting *"Firmware (SD card)"* the firmware of the PLC as well as the communication module is exported.

A confirmation message shows additional information on the export including the exported firmware.

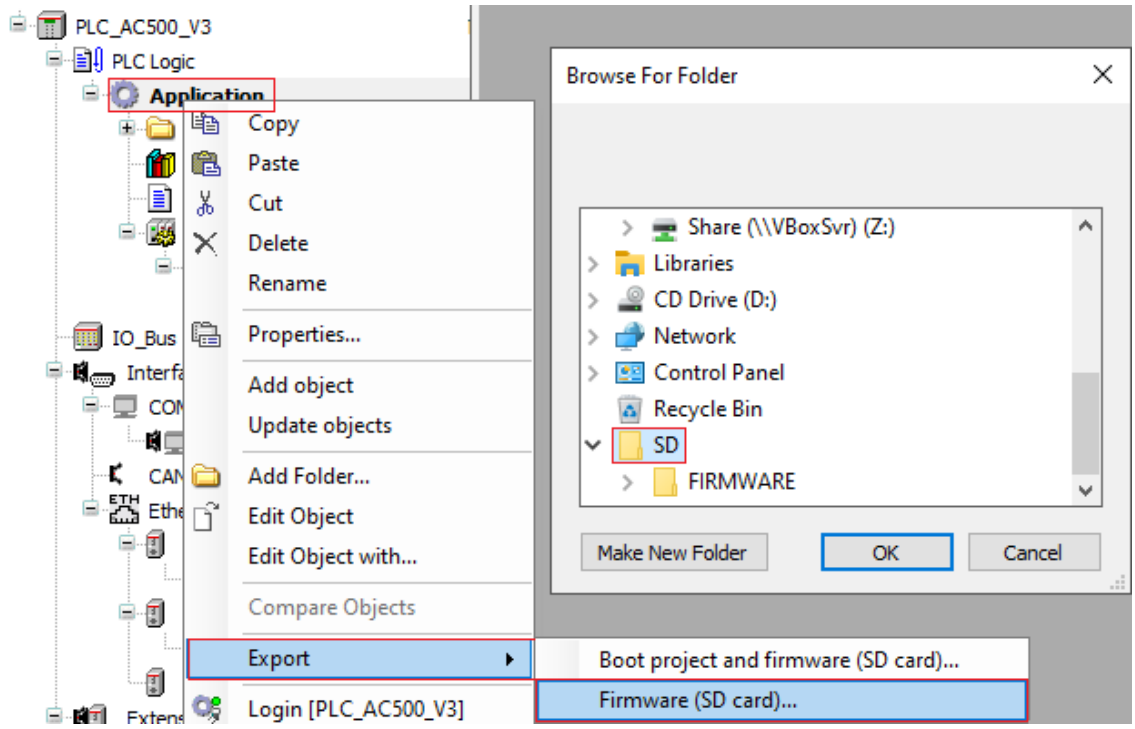
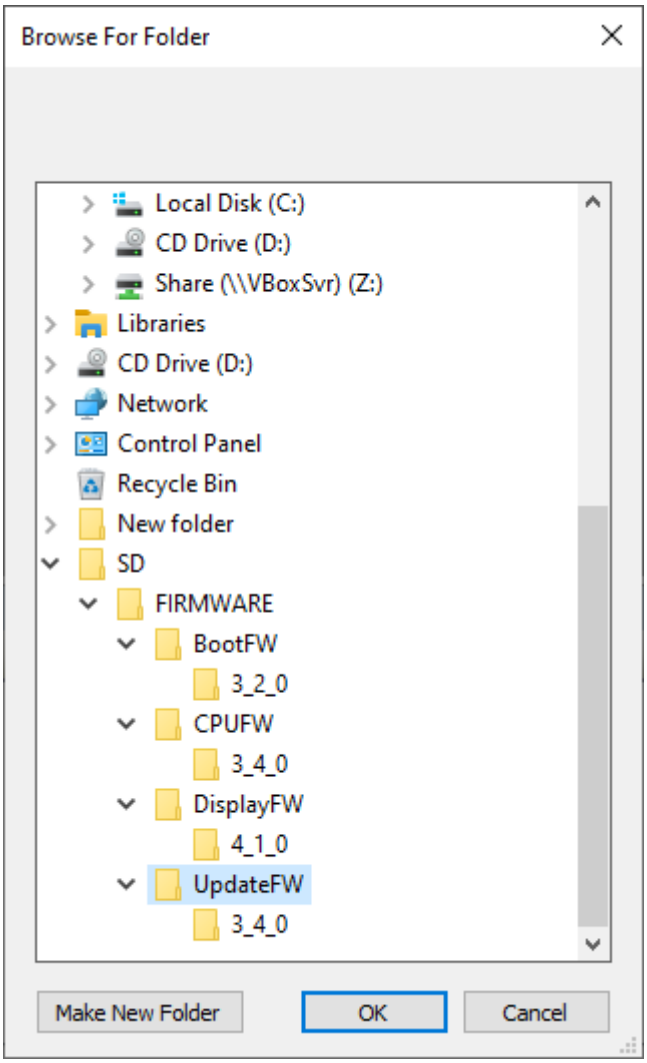


Fig. 343: Example for AC500 V3

1. Right-click *"Application"* in the device tree.
2. Select *"Export ➔ Firmware (SD card)..."* .
3. Click *[Make New Folder]* and type in "SD".
4. Select *[OK]* to add the folder.
5. Select folder *"SD"*



⇨ The SD structure has been created and the firmware has been exported.

Mark all subfolders and files of the SD folder and copy them to a memory card. Do not copy the SD folder, only the subfolders and files!

Execution of update via memory card

The following steps describe the procedure for updating the firmware or the boot project using a memory card. Prerequisite is the previous download of the current firmware to the memory card either as export from the Automation Builder as described in the previous chapter or as online download from ABB.

Direct from *https://share.library.abb.com/api/v4?cid=9AAC177288&dk=Software*.

Click this link and on the next web page find the relevant firmware package and download it.

- Unpack this .zip archive file at any location of your hard disc
- Insert empty formatted (FAT16 / FAT32) memory card in the PC card reader
- Execute the unpacked *.exe file
- Select PC card reader as the final destination and confirm.

All directories, files and SDCARD.INI file will be automatically created on memory card and properly configured. After the process is complete, one has the prepared memory card with relevant updates.

Boot project and firmware update

1. Switch off the device.
2. Insert the memory card.
3. Switch on the device.
   ⇨ The alternate flashing of the RUN and the ERR LED indicates the running update process.

   At the end of the update process a reboot is executed and the boot project and system firmware is started for the finishing of the update process.

   If RUN LED blinks (ERR LED is off), the update was successful and the display shows *“done”*.

   If ERR LED blinks (RUN LED is off), the update failed and the display shows *“FAIL”*.

   The text file “SDCARD.RDY” includes the results of the different updates. If the update fails, the file contains the reasons for the abort. Based on this, further steps can be taken to fix the problem.
4. Switch off the device.
5. Remove the memory card.
6. Switch on the device.
   ⇨ The system starts with the new boot project and firmware on the CPU.

Firmware update

1. Switch off the device.
2. Insert the memory card.
3. Switch on the device.
   ⇨ The alternate flashing of the RUN and the ERR LED indicates the running update process.

   At the end of the update process a reboot is executed and the system firmware is started for the finishing of the update process.

   If RUN LED blinks (ERR LED is off), the update was successful and the display shows *done*.

   If ERR LED blinks (RUN LED is off), the update failed and the display shows *FAIL*.

   The text file “SDCARD.RDY” includes the results of the different updates. If the update fails, the file contains the reasons for the abort. Based on this, further steps can be taken to fix the problem.
4. Switch off the device.
5. Remove the memory card.
6. Switch on the device.
   ⇨ The system starts with the new firmware on the CPU.

Description of LEDs

The LEDs below the display indicate the status of the processor module:

LED	State	Color	LED = ON	LED = OFF	LED flashes
Power LED (PWR)	Denotes the power supply state of the processor module	Green	Voltage is present (24 V DC)	Voltage is missing	-
Run LED (RUN)	Denotes the activity state of the processor module	Green	Processor module is in RUN mode	Processor module is in STOP mode	If the LED flashes fast (4 Hz) a firmware update is finished with no errors. If the Run LED flashes fast (4 Hz), alternating with a flashing Run LED the firmware is updated. To enforce boot mode 1, keep the RUN function key pressed during the boot procedure. In this case, the Run LED flashes slowly (1 Hz). A subsequent project download (from within Automation Builder) cancels the blinking.
Error LED (ERR)	Denotes an error	Red	An error has occured.	No errors or only warnings have occurred.	If the Error LED flashes slowly (1 Hz) a firmware update from the memory card is finished with errors. If the Error LED flashes fast with AC500 on display a fatal system error has occurred. If the Error LED flashes fast (4 Hz) alternating with a flashing Run LED the firmware is updated.

A running processor module is indicated with the state RUN on the display, a deactivated processor module is indicated with the state STOP. In both cases the display's backlight is off.

1.6.7.2.2 Content of the memory card for firmware/application update

Only advanced users should apply the instructions in this chapter.

Memory card file content: Firmware version V3.x

Only advanced users should apply the instructions in this chapter.

General update process

Information on the firmware: *Chapter 1.6.6.1.4.2 “ AC500 V3 firmware installation and update” on page 3653*

The main components of the V3 CPU firmware are:

- BootFW (boot firmware): responsible for the starting of the UpdateFW or the SystemFW
- UpdateFW (update firmware): responsible for the update of BootFW, UpdateFW, SystemFW, UpdateHook and boot project
- SystemFW (system firmware, CPUFW): Runtime system of the PLC, additionally responsible for the update of the DisplayFW (display firmware) and the firmware of the communication module
- DisplayFW (display firmware): Firmware of the display

Additionally the update process includes the following parts:

- Communication module (communication module firmware): Firmware of the different communication module
- UpdateHook: Specific patches for the PLC
- UserProgram: Boot project of the application
- License features: Import and export of license files. The license file for the "ImportLicense" is a Wbb or a WibuCmRaU file. The license file of the "ExportLicense" is a WibuCmRaC file.

The firmware updates are triggered by the command file *SDCARD.INI*. In addition a result file of the firmware update is generated (*SDCARD.RDY*, identical path as *SDCARD.INI*). For the group [FirmwareUpdate] the parameters 0, 11, 12 and 13 are defined. For each firmware update two files are necessary. The firmware file and the corresponding signature file.

For example:

AC500_V3_SystemFirmware_V3.0.1.73.tar.bz2

AC500_V3_SystemFirmware_V3.0.1.73.tar.bz2.sig

AC500_V3_DisplayFirmware_V3.0.0.0.app

AC500_V3_DisplayFirmware_V3.0.0.0.app.sig

For the user program the application file and the application *CRC* are necessary. For example:

Application.app

Application.crc

If the signature file and the firmware file do not match, no update is performed and the correspondent error result is written to the file *SDCARD.RDY*.

If the update firmware is running the display shows the text *update*. The blinking of the *RUN* and the *ERR* LED’s indicates the update process.

The file “SDCARD.RDY” includes the results of the different updates. After an update of a communication module *CODESYS Control* is started in safe mode (no download or starting of the application is possible) and the PLC needs a reboot (power down/up; the display shows *please* and *reboot* alternately).

As of system firmware 3.2 the compatibility file "Version.txt" (with the corresponding signature file "Version.txt.sig", identical path as "SDCARD.INI") is necessary for the update process. The update firmware checked the compatibility of the following parts:

- CPUFW (system firmware)
- BootFW (boot firmware

- UpdateFW (update firmware)
- DisplayFW (display firmware)

A missing "Version.txt" or a missing/corrupt "Version.txt.sig" file is signalled at the component "CPUFW" (file "SDCARD.RDY").

If the update process would result in incompatible parts of firmware no update is performed in the update firmware. After starting of the system firmware the compatibility of the communication module firmware is checked additionally. The check of the compatibility of the firmware is executed always (independent of the parameter for the component). Incompatibility is signalled at the corresponding component (file SDCARD.RDY").

Command file *SDCARD.INI* for AC500 V3 Products

Only advanced users should apply the instructions in this chapter.

[FirmwareUpdate]	0 = No update
CPUFW=x x= 0, 11, 12, 13	11 = Update system firmware always with the file specified in module's section [CPU] and component's path key “CPUFW”. 12 = Update with different version, the update is only performed if the version of the file specified by the component path key “CPUFW” in module’s section [CPU] differs from the current version of the CPU. 13 = Update with newer version, the update is only performed if the version of the file specified by the component path key “CPUFW” in module’s section [CPU] is newer than the current version of the CPU.
BootFW=x x= 0, 11, 12, 13	See description CPUFW. The component's path key for the boot firmware in module’s section [CPU] is “BootFW”.
UpdateFW=x x= 0, 11, 12, 13	See description CPUFW. The component's path key for the update firmware in module’s section [CPU] is “UpdateFW”.
DisplayFW=x x= 0, 11, 12, 13	See description CPUFW. The component's path key for the display firmware in module’s section [CPU] is “DisplayFW”.
UpdateHook=x x= 0, 11	11 = Execute UpdateHook always with the file specified in module's section [CPU] and component's path key “UpdateHook”.
ImportLicense=x x= 0, 12	12 = Import the license always with the file specified in module's section [CPU] and component's path key “ImportLicense”. The license file is a Wbb or a WibuCmRaU file. The update process imports this file into the plc. Note: Do not use parameter 11 for license import.
ExportLicense=x x= 0, 12	12 = Export the license always to the file specified in module's section [CPU] and component's path key “ExportLicense”. The exported license file is a WibuCmRaC file. Note: Do not use parameter 11 for license export.

Coupler0=x x= 0, 11, 12, 13	0 = No update. 11 = Update firmware always with the file specified in module's section [Coupler0] and component's path key “Boot” and/or “Firmware”. 12 = Update with different version, the update is only performed if the version of the file specified by the component path key “Boot” and/or “Firmware” in module’s section [Coupler0] differs from the current version of the coupler. 13 = Update with newer version, the update is only performed if the version of the file specified by the component key “Boot” and/or “Firmware” in module’s section [Coupler0] is newer than the current version of the coupler.
Coupler1=x x= 0, 11, 12, 13	Update module slot 1; see description Coupler0, module section is [Coupler1]*.
Coupler2=x x= 0, 11, 12, 13	Update module slot 2; see description Coupler0, module section is [Coupler2]*.
Coupler3=x x= 0, 11, 12, 13	Update module slot 3; see description Coupler0, module section is [Coupler3]*.
Coupler4=x x= 0, 11, 12, 13	Update module slot 4; see description Coupler0, module section is [Coupler4]*.
Coupler5=x x= 0, 11, 12, 13	Update module slot 5; see description Coupler0, module section is [Coupler5]*.
Coupler6=x x= 0, 11, 12, 13	Update module slot 6; see description Coupler0, module section is [Coupler6]*.
[UserProg]	0 = No update.
UserProgram=x x= 0, 11	11 = Update user program always with the file specified in module's section [CPU] and component's path key “UserProgram”.

Example: SDCARD.INI as of CPU firmware V3.x

Only advanced users should apply the instructions in this chapter.

```
 [Status]
;FunctionOfCard
;0 = Perform no function when inserting the card or voltage ON
;1 = Load user program according to entry in group |UserProg|
;2 = Start firmware update according to entry in group |
FirmwareUpdate|
;3 = Update firmware according to entry in group |FirmwareUpdate|
;    and load user program according to entry in |UserProg|
FunctionOfCard=0

 [FirmwareUpdate]
; 0 = No update
;11 = Update with file specified in module's section <modsec>,
component's path key <pathkey>
;12 = Like 11, but check version of file to be updated differs from
current one.
```

```
;13 = Like 11, but check version of file to be updated is newer than
current one.
CPUPFW=0          ;<modsec>=|CPU|, <pathkey>= CPUPFW
BootFW=0          ;<modsec>=|CPU|, <pathkey>= BootFW
UpdateFW=0        ;<modsec>=|CPU|, <pathkey>= UpdateFW
DisplayFW=0       ;<modsec>=|CPU|, <pathkey>=DisplayFW
UpdateHook=0      ;<modsec>=|CPU|, <pathkey>=UpdateHook
ImportLicense=0   ;<modsec>=|CPU|, <pathkey>=ImportLicense
ExportLicense=0   ;<modsec>=|CPU|, <pathkey>=ExportLicense
Coupler0=0        ;<modsec>=|Coupler0|, <pathkey>=Firmware
Coupler1=0        ;<modsec>=|Coupler1|, <pathkey>=Firmware
Coupler2=0        ;<modsec>=|Coupler2|, <pathkey>=Firmware
Coupler3=0        ;<modsec>=|Coupler3|, <pathkey>=Firmware
Coupler4=0        ;<modsec>=|Coupler4|, <pathkey>=Firmware
Coupler5=0        ;<modsec>=|Coupler5|, <pathkey>=Firmware
Coupler6=0        ;<modsec>=|Coupler6|, <pathkey>=Firmware

[UserProg]
; 0 = No update
;11 = Update with file specified in module's section <modsec>,
component's path key <pathkey>
UserProgram=0     ;Update user program. <modsec>=[CPU], <pathkey>=
UserProgram

[CPU];
CPUFW=            ;Path/file of CPU's system firmware to update
BootFW=           ;Path/file of CPU's boot firmware to update
UpdateFW=         ;Path/file of CPU's update firmware to update
UpdateHook =      ;Path/file of UpdateHook to update
DisplayFW=        ;Path/file of Display's firmware to update
ImportLicense=    ;Path/file of import license file
ExportLicense=    ;Path/file for export license file
UserProgram=      ;Path/File of user program to update


[Coupler0]
Firmware=         ;Path/file of internal coupler's firmware to update

[Coupler1]
Firmware=         ;Path/file of external coupler's firmware slot 1 to
update

[Coupler2]
Firmware=         ;Path/file of external coupler's firmware slot 2 to
update

[Coupler3]
Firmware=         ;Path/file of external coupler's firmware slot 3 to
update

[Coupler4]
Firmware=         ;Path/file of external coupler's firmware slot 4 to
update

[Coupler5]
Firmware=         ;Path/file of external coupler's firmware slot 5 to
update

[Coupler6]
Firmware=         ;Path/file of external coupler's firmware slot 6 to
update
```

Example content and description of the SDCARD.INI folder

SDCARD.INI for memory card for update only the system firmware (SystemFW):

[FirmwareUpdate]

CPUPFW=11

[CPU]

CPUFW=/SystemFirmware/ AC500_V3_SystemFirmware_V3.1.3.zzz.tar.bz2

1.6.7.3 Flash memory for AC500 V3 products

AC500 processor modules for V3 products (PM56xx) are equipped with non-removable and non-volatile onboard user flash memory for program and data storage. The integrated flash management, including a wear levelling algorithm and a power-fail protected file system, is designed for robustness and operation in industrial environments and applications. The user flash memory can be accessed from the user program using the CAA_File library.

NOTICE!
The user flash memory has a finite number of write cycles.

Important: Programmers should keep the amount of cyclic written data low to ensure long availability.

1.6.7.4 Health monitoring

AC500 V3 products are equipped with non-removable and non-volatile onboard user flash memory for program and data storage. The integrated flash management, including a wear levelling algorithm and a power-fail protected file system, is designed for robustness and operation in industrial environments and applications.

Keep the amount of cyclic written data low to assure long availability of the user flash memory. The spent/remaining lifetime information of the user flash memory can be acquired with the function block PmDiskStatus and PmDiskLifetimeUsed.

Further information is provided in the documentation of the *AC500_Pm* library. *Chapter 1.10 “Reference, function blocks” on page 4292*

Since FW version 3.3.0, there is also a diagnosis message issued when the user flash memory reaches the end of its lifecycle. Please refer to the diagnosis documentation for more info.

1.7 Diagnosis and debugging for AC500 V3 products

1.7.1 The diagnosis system

The diagnosis system enables uniform diagnosis of the CPU and its local interfaces, of the local I/O bus with the connected S500 I/O devices and of the fieldbuses connected via communication modules, considering the special features of the various fieldbuses. The safety CPU is also integrated into the diagnosis system.

Diagnosis data of the devices can be accessed by

- CPU display
- Automation Builder
- IEC application

To forward the information to notify them by, e.g., webserver or OPC UA server, the data retrieved in IEC application can be stored in variables.

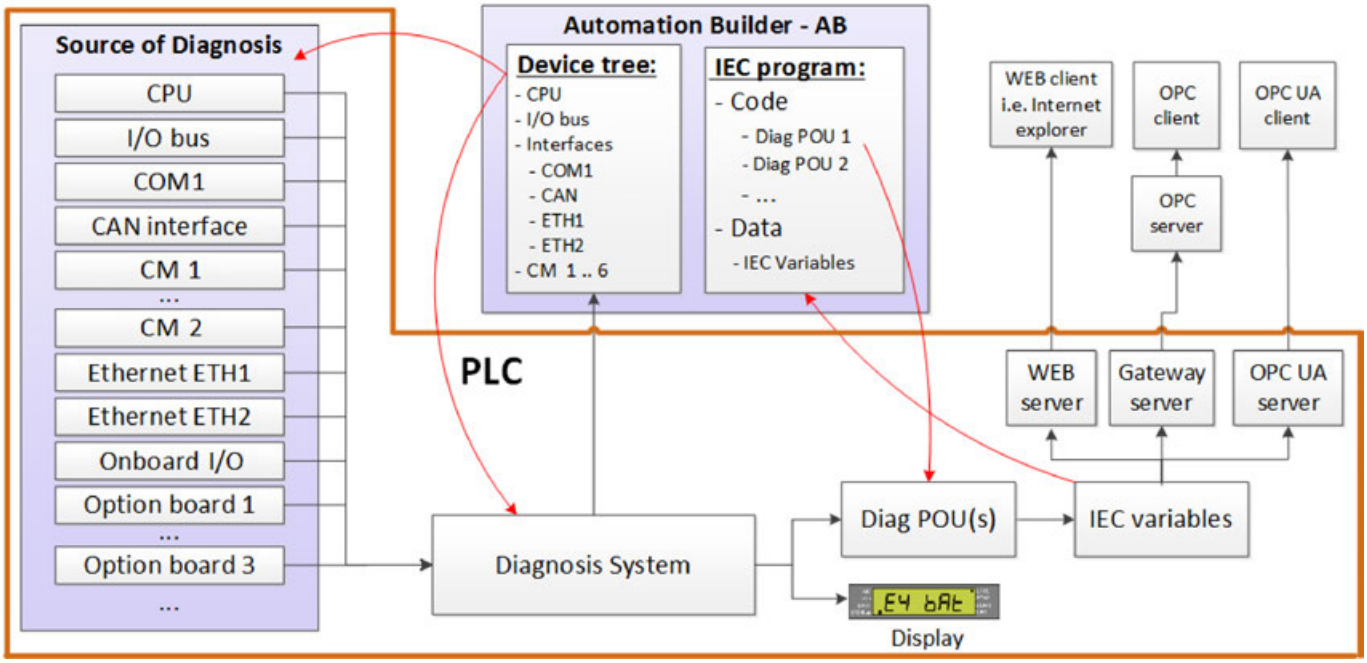


Fig. 344: Overview of the diagnosis system

All diagnosis data is assigned to a device. System diagnosis (e.g., battery low) is coming from the CPU device or one of the child objects (e.g., watchdog diagnosis of a task object).

Diagnosis is available for devices with representation in the Automation Builder device tree.

Diagnosis messages include the severity of an error. Error severity can be used for defining system behavior, e.g., activating the error LED or stop the PLC ↳ *Chapter 1.7.1.5.1 “Error severity” on page 4044*.

Types of diagnosis messages

- An **event** describes the current state of the device. It does not have to be acknowledged.
- An **alarm** describes that at a certain point of time, there was a diagnosis message. It does not say anything about the current state of the device.
  Alarms must be acknowledged by the user. After acknowledging, the alarm disappears for all consumers.

Every diagnosis message has a come time.

Device state

With reference to diagnosis, there are different device states:

- Device without events and without unacknowledged alarms.
- Device with events or unacknowledged alarms.
- Device does not respond and is not available for online connection.

Diagnosis descriptions

Diagnosis messages are always available for all consumers.

- ABB AC500 V3 devices:
  - Events and unacknowledged alarms.
  - Every diagnosis message with come time, location, error number and text.
- 3rd party devices:
  - Events and unacknowledged alarms.
  - Every diagnosis message with come time, location and error number.
  - Clear text information if available either from a standard or from the device description.
  - If available: Extended diagnosis: Additional data coming from the device for manual analysis.

Extended diagnosis

Some devices are able to provide extended diagnosis. This additional device-dependant diagnosis will only be collected on request and will be device type specific (e.g. bus scan request on PROFINET I/O controller). Main intention is to cover commissioning use cases, when very specific information is required that typically cannot be stored in error numbers in a reasonable way.

1.7.1.1 Access to diagnosis data

Access to device state

- Error LED on CPU *Chapter 1.7.1.2 “Diagnosis in CPU display” on page 4013*
- Automation Builder device tree *Chapter 1.7.1.3 “Diagnosis in Automation Builder” on page 4017*
- IEC application via device name *Chapter 1.7.1.4.3 “Device diagnosis” on page 4034*
- IEC application via list of all available diagnosis *Chapter 1.7.1.4.2 “System diagnosis” on page 4025*
- External access via global IEC variables Fig. 344

Access to diagnosis descriptions

- CPU display *Chapter 1.7.1.2.2 “Diagnosis descriptions” on page 4013*
  (for CPU, local I/O bus and connected S500 I/O modules, not for communication modules and field buses)
- Automation Builder via *“All messages”* window *Chapter 1.7.1.3.2 “Diagnosis descriptions” on page 4017*:
  - Support for (bulk) acknowledgement of alarm
  - Access to extended diagnosis data of 3rd party devices devices (if available), without any interpretation
- IEC application via a list of all current diagnosis either of a device (device object from the Automation Builder tree) *Chapter 1.7.1.4.3 “Device diagnosis” on page 4034* or of the complete PLC *Chapter 1.7.1.4.2 “System diagnosis” on page 4025*:
  - Navigation chronologically in both directions (starting either from the oldest or newest diagnosis)
  - Access to the diagnosis numerically (evaluation by IEC application), textual (use on HMI) or to extended diagnosis data of 3rd party devices (if available)
  - Acknowledgement of alarms
- External access via global IEC variables by using the IEC application features for getting all relevant information

Access to extended diagnosis

Extended diagnosis data will be displayed in Automation Builder via *“All messages”* window. The data is displayed as it is provided from the device without any interpretation. Refer to, e.g., the manual of the device to get information about the extended diagnosis data.

1.7.1.2 Diagnosis in CPU display

1.7.1.2.1 Device state

If there is at least one active diagnosis message, the error LED ERR is on.

The behavior of the error LED depends on the setting of CPU parameter *“Error LED” ↳ Table on page 3693*.

> *Diagnosis of AC500-eCo CPUs can only be shown by LED ERR at CPU. No display is available.*

General information on the LEDs, the display and the function keys can be found in chapter *↳ Chapter 1.6.5.1.6 “LEDs, display and function keys on the front panel” on page 3486*.

1.7.1.2.2 Diagnosis descriptions

↳ Chapter 1.7.1.2.3 “Reading out diagnosis messages on the CPU” on page 4015.

Entry	Length [byte]	Values	Description	Display
Error severity	1	0 .. 255	Used values: 1, 2, 3, 4, 11 *↳ Chapter 1.7.1.5.1 “Error severity” on page 4044*	**Ex** abc
Hardware ID (HwId)	1	0 .. 255	Location of diagnosis, e.g., subdevice, as three-letter word *↳ Further information on page 4013*	Ex **abc**
Error code	2	1 .. 65535	Error number (low word)	12345
SubSysteminfo byte 1	1	0 .. 255	Depends on hardware ID	d1 123
SubSysteminfo byte 2	1	0 .. 255	Depends on hardware ID	d2 123
SubSysteminfo byte 3	1	0 .. 255	Depends on hardware ID	d3 123
SubSysteminfo byte 4	1	0 .. 255	Depends on hardware ID	d4 123

> *CPU display does not show any communication modules or fieldbus diagnosis. To view these diagnosis messages use Automation Builder ↳ Chapter 1.7.1.3 “Diagnosis in Automation Builder” on page 4017 or IEC application ↳ Chapter 1.7.1.4 “Diagnosis in IEC application” on page 4020.*
>
> *This is valid for:*
>
> – *all external communication modules incl. safety CPUs, CM574-RS, FM502-CMS*
> – *CANopen on onboard CAN interface*
> – *fieldbuses on Ethernet interfaces ETH1/ETH2 like PROFINET IO controller, EtherCAT master, etc.*

For identification of the location of a diagnosis the hardware ID and in addition for CPU diagnosis the SubSysteminfo byte 1 is used.

The location is displayed with 3 characters.

Hardware ID	Value	SubSysteminfo byte 1	Value	Display
CPU	0	CPU itself	0	
CPU	0	RAM	17	
CPU	0	Flash	18	
CPU	0	Flashdisk	19	
CPU	0	SD memory card	20	
CPU	0	Display	21	
CPU	0	Battery	22	
CPU	0	RTC (real-time clock)	23	
CPU	0	FPU (floating point unit)	24	
CPU	0	Power supply	25	
Communication module 1	1			
Communication module 2	2			
Communication module 3	3			
Communication module 4	4			
Communication module 5	5			
Communication module 6	6			
COM1 serial interface 1	7			
COM2 reserved for serial interface 2	8			
CAN interface	9			
Onboard I/O (eCo)	10			No display
Option board 1 (eCo)	11			No display
Option board 2 (eCo)	12			No display
Option board 3 (eCo)	13			No display

Hardware ID	Value	SubSysteminfo byte 1	Value	Display
I/O bus	14			
Ethernet ETH1	15			
Ethernet ETH2	16			

1.7.1.2.3 Reading out diagnosis messages on the CPU

Table 747: Example: no diagnosis message in status list

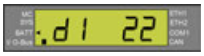
State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed	-	-	-	-
1			No action	No action	Return into RUN/STOP mode.	

Table 748: Example: diagnosis messages in status list

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed	-	-	-	-
1	 Number of diagnosis messages; here 4		Go to first/ next diagnosis message in status list (e.g., state 2)	Go to last/ previous diagnosis message in status list	Return into RUN/STOP mode.	Return into RUN/STOP mode.
2	 Diagnosis message example: *Error battery empty or missing* Toggling between state 2 and 3	Selects displayed diagnosis message and shows details *Table 749 "Example: error battery empty or missing" on page 4016*	Go to first/ next diagnosis message in status list	Go to last/ previous diagnosis message in status list	Return into RUN/STOP mode.	Acknowledge and return into RUN/STOP mode.

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
3	 Error ID example Toggling between state 2 and 3					

Table 749: Example: error battery empty or missing

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	 E4 = error severity 4 bAt = subdevice battery Toggling between state 0 and 1	State 2 is displayed	State 2 is displayed	State 6 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
1	 Error ID example Toggling between state 0 and 1					
2	 Error number 8 Battery is missing or empty		State 3 is displayed	State 0 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Displays state 0 Return to diagnosis status list
3	 Detail 1 Subdevice 22: battery		State 4 is displayed	State 2 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
4	 Detail 2 Error type 0: device		State 5 is displayed	State 3 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
5	 Detail 3 Error type number 0: device itself		State 6 is displayed	State 4 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
6	 Detail 4 Additional information 0: none		State 1 is displayed	State 5 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list

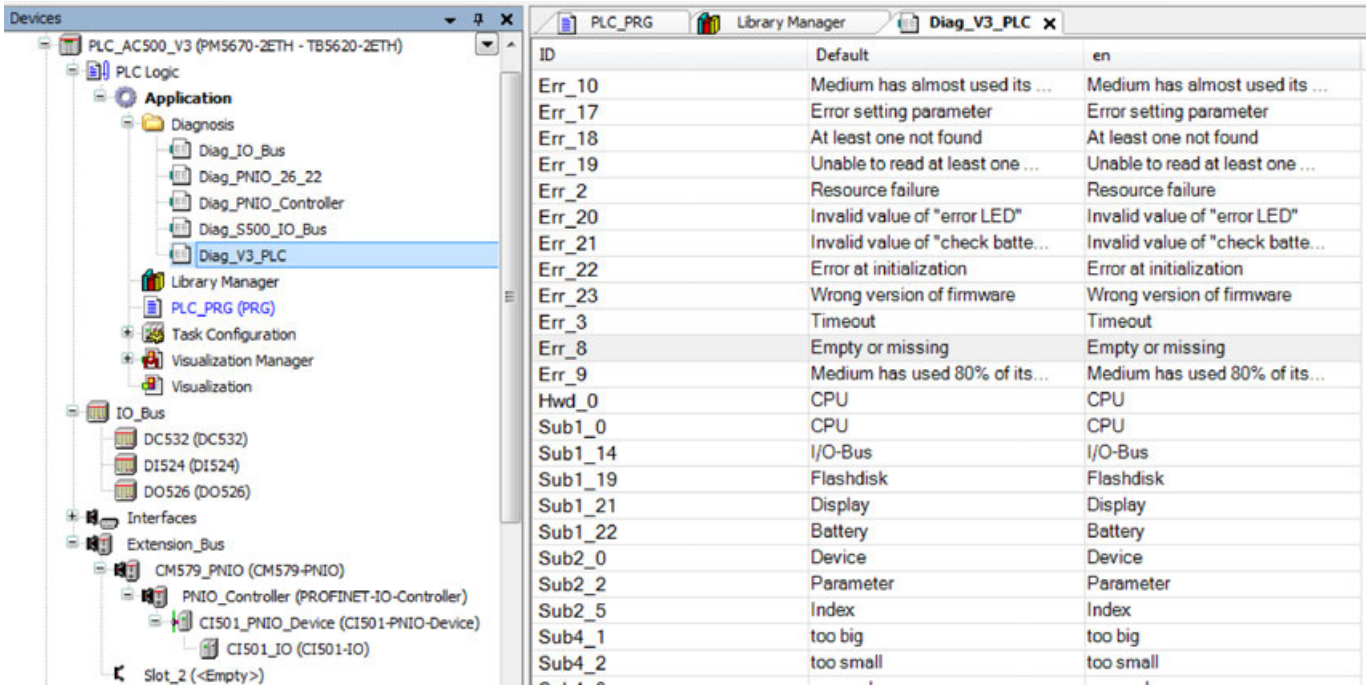
1.7.1.3 Diagnosis in Automation Builder

1.7.1.3.1 Device state

In Automation Builder, colored icons next to the devices' nodes in the device tree indicate the device state of each single device ↳ *Chapter 1.7.2.3 "Project tree in online mode" on page 4047*.

1.7.1.3.2 Diagnosis descriptions

Displayed text For output of diagnosis messages in textual format Automation Builder and IEC application use text lists. Both application use the same text lists. The text lists are part of the device description. When inserting a new device in device tree of project, the corresponding text list is loaded. This text lists are part of PLC program and will be downloaded into the PLC.



It is necessary to include a visualization, even if visualization will not be used.
Chapter 1.4.5 “CODESYS Visualization” on page 1249

Without visualization the text lists will not be included.

The text lists are generated automatically. We recommend that you do not change them manually because the changes can be overwritten automatically and without prompting.

Displayed text for 3rd party devices

The text lists for 3rd party devices are created during reading of the device description sheets, e.g., GSDML files for PROFINET I/O devices.

The name of a text list for a PROFINET I/O device is: Diag_PNIO_Vendor ID_Device ID

Example

CI501-PNIO: Diag_PNIO_26_22

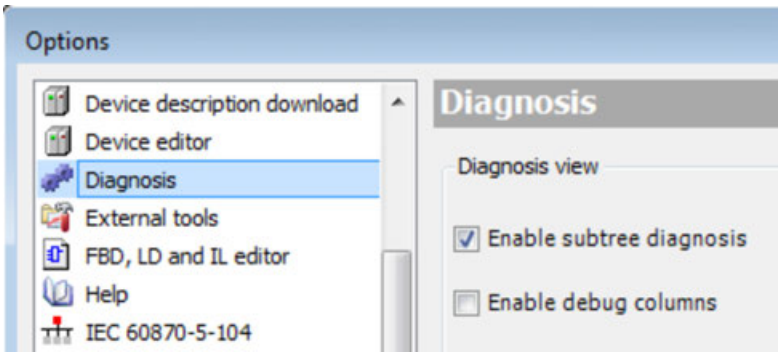
26 = vendor ID ABB, 22 = device ID CI501-PNIO

The text list for the AC500 PROFINET I/O modules contains all text needed for PROFINET standard diagnosis and AC500 process alarm handling.

Which texts are used, depends on parameter *“Selection of diagnosis method”*: Double-click on a PROFINET I/O module and open tab *“General”*.

1.7.1.3.3 System diagnosis

In Automation Builder the system diagnosis is activated by default and can be deactivated in: *“Tools ➔ Options ➔ Diagnosis ➔ Enable subtree diagnosis”*



1.7.1.3.4 Device diagnosis

Each node in the device tree has a diagnosis view, which displays the diagnosis messages for this device only.

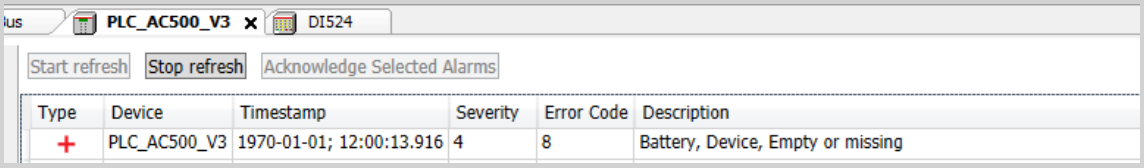
The message consists of:

- Type *Chapter 1.7.1.3.5 “Diagnosis history” on page 4019*
- Timestamp in date and time YYYY-MM-DD hh:mm:ss.ms
- Error severity
- Error code
- Diagnosis description
- Additional data

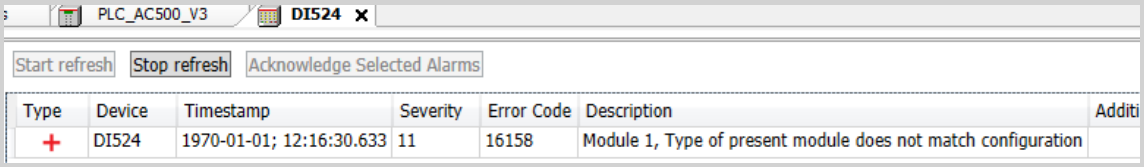
To view the diagnosis message:

1. Double-click on a device.
2. Select the tab *"Diagnosis"*.

Example Battery empty or missing.



Example Wrong module configured on I/O bus.



Device diagnosis is disabled by default.

To enable/disable device diagnosis:

1. Double-click on the PLC.
2. Select the tab *"PLC Settings"*.
3. Under *"Additional Settings"* enable/disable *"Diagnosis for devices"*.
   ⇨ When the device diganosis is disabled, this symbol will be displayed in the device tree and no diagnosis messages will be shown.

1.7.1.3.5 Diagnosis history

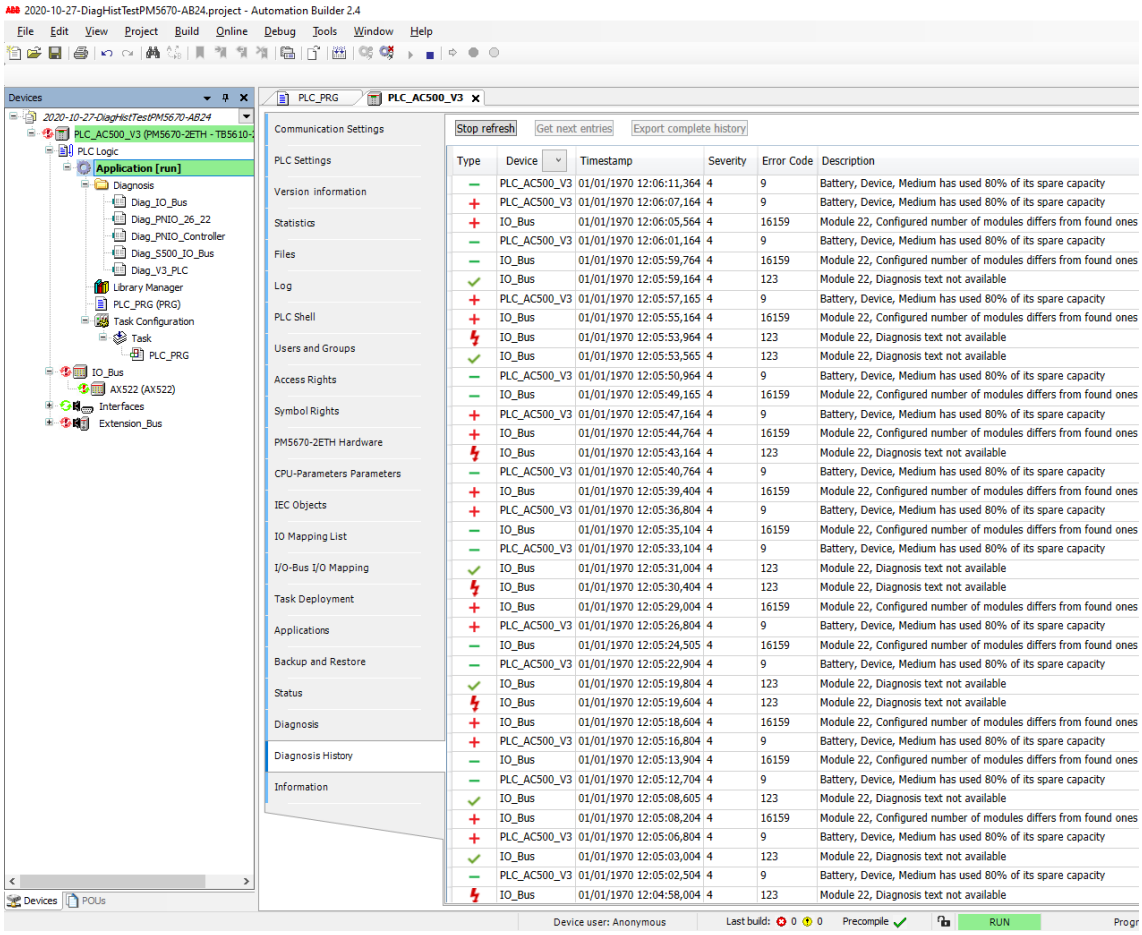
Diagnosis history is available as of Automation Builder 2.4.0 / System FW 3.4.0 the diagnosis system has been extended with diagnosis history.

The 'Diagnosis History' view provides an overview of the current and past system events that resulted in a diagnosis event.

- Incoming diagnosis events are indicated with +.
  After the problem that causes a diagnosis event has been resolved, this diagnosis event is indicated automatically with −.
- Alarm events, e.g. PROFINET alarms are indicated with .
  In the 'Diagnosis' view the user can acknowledge an alarm. Note that an alarm event can be acknowledged though the problem that causes the alarm still persists.
  The acknowledge action is indicated with on the concerning event entry. If the icon changes to , the acknowledge action has been completed by the PLC.

The following buttons are available in the 'Diagnosis History' view:

- Start/Stop refresh:
  Enables or disables the automatic refresh mode. In refresh mode new diagnosis events will be displayed automatically. Only the last 100 entries are shown in this view, the latest events on top of the list.
- Get next entries:
  Adds the previous (older) 100 diagnosis events at the bottom of the list.
- Export complete history:
  Creates a csv file with all events from the diagnosis history (not only the visible ones).



1.7.1.4 Diagnosis in IEC application

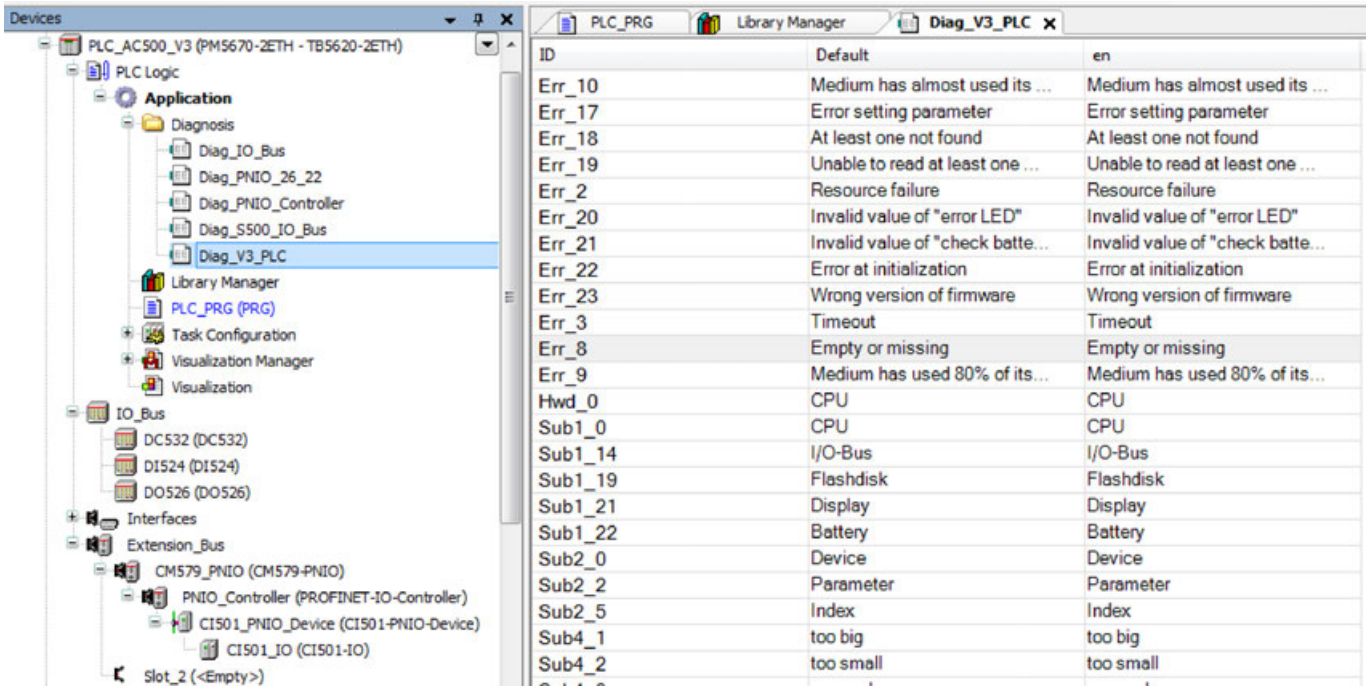
There are two possibilities for accessing the diagnosis messages in the IEC application:

- System diagnosis: Access to diagnosis messages of the whole PLC
- Device diagnosis: Access to the diagnosis messages of a device

For both possibilities common data types (structures and enumerations) are defined in the library AC500_DiagTypes ↬ *Chapter 1.7.1.4.1 “Data types in library AC500_DiagTypes” on page 4021*. The library is automatically included in PLC project.

Displayed text

For output of diagnosis messages in textual format Automation Builder and IEC application use text lists. Both application use the same text lists. The text lists are part of the device description. When inserting a new device in device tree of project, the corresponding text list is loaded. This text lists are part of PLC program and will be downloaded into the PLC.



It is necessary to include a visualization, even if visualization will not be used.
↳ Chapter 1.4.5 “CODESYS Visualization” on page 1249

Without visualization the text lists will not be included.

The text lists are generated automatically. We recommend that you do not change them manually because the changes can be overwritten automatically and without prompting.

Displayed text for 3rd party devices

The text lists for 3rd party devices are created during reading of the device description sheets, e.g., GSDML files for PROFINET I/O devices.

The name of a text list for a PROFINET I/O device is: Diag_PNIO_Vendor ID_Device ID

Example

CI501-PNIO: Diag_PNIO_26_22

26 = vendor ID ABB, 22 = device ID CI501-PNIO

The text list for the AC500 PROFINET I/O modules contains all text needed for PROFINET standard diagnosis and AC500 process alarm handling.

Which texts are used, depends on parameter *“Selection of diagnosis method”*: Double-click on a PROFINET I/O module and open tab *“General”*.

1.7.1.4.1 Data types in library AC500_DiagTypes

All data types regarding diagnosis are defined in the library AC500_DiagTypes.

Structure DIAG_VAL_TYPE

This data type specifies the format of all kinds of diagnosis messages in numeric format. It consists of one element for each detail of a diagnosis message.

Name	Type	Initial	Comment
diTimestamp	DT	DATE_AND_TIME#1970-1-1-0:0	RTC time of eDiagEvent_Occured
uiMs	UINT	0	Milliseconds of event
eClass	teClass	teClass.eDiagClass_4_Warning	Error severity of diagnosis message
szDevice	STRING(80)	""	Name of device, max. 80 characters
eHwInterfaceId	teHwId	teHwId_CPU	Identifier of hardware interface
dwSubSysteminfo	DWORD	0	Any number describing details or location within device, device-specific
dwAdditional	DWORD	0	Additional number describing details or location within device, optional, device-specific
dwErrorCode	DWORD	0	Error code
wSizeExtDiag	DWORD	0	Number of bytes of extended diagnosis data

Structure DIAG_TXT_TYPE

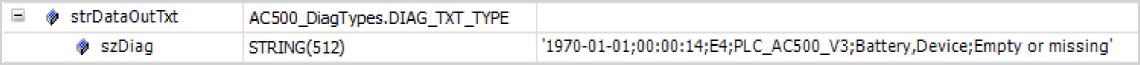
This data type specifies the format of all kinds of diagnosis messages in textual format. It consists of a single string containing all details of a diagnosis message.

Name	Type	Initial	Comment
szDiag	STRING(512)	""	Diagnosis message as text, max. 512 characters

The text consists of the following data, separated by semicolon:

- Timestamp in Date_And_Time (DT) format of eDiagEvent_Occured and added milliseconds
- Error severity
- Device name (max. 80 characters) as defined in Automation Builder device tree
- The error text itself, composed of the interpretation of dwSubSysteminfo and dwAdditional and the error text plus remedy (if available) from Automation Builder text list according dwErrorCode. Displayed as: error text -> remedy.

Example Battery empty or missing.



Enumeration ERROR_ID

Type of the return values of all methods and functions to request information on diagnosis.

Name	Type	Comment	Remedy
NO_ERROR	16#0	Execution successfully completed	
ERR_PARAMETER	16#1	Invalid parameter value in function call	Correct parameter
ERR_NO_SINK	16#2	Failed to register as sink	
ERR_NO_TEXT_LIST	16#3	Failed to get a device text list	Check text lists

Name	Type	Comment	Remedy
ERR_NO_TEXT_ CONTENT	16#4	Failed to get at least one content from text list	
ERR_COMPETING	16#5	Failed due to competing access of other method	Try again
ERR_ASYNC	16#6	Failed to create async process	
ERR_INTERNAL	16#7	Any internal error during execution	
BUSY	16#FFF	Busy	Call again to get final result
NO_ERROR_NO_DATA	16#FFFF	Execution successfully completed, no more diagnosis messages	

All values except "BUSY" are final results. In case "NO_ERROR" is returned, the requested action has been successfully performed. "NO_ERROR_NO_DATA" also indicates a successful completion. The only difference to "NO_ERROR" is the fact, that there is one (more) data to be provided. All other return values (except "BUSY") are final error states. In case a method or function returns "BUSY", it has to be called again in the following cycles until it returns a final result.

Enumeration teClass

Specifies the error severity of diagnosis messages ↳ *Chapter 1.7.1.5.1 “Error severity” on page 4044*.

Name	Error severity
eDiagClass_2_SeriousError	2
eDiagClass_3_Error	3
eDiagClass_4_Warning	4
eDiagClass_Parameter	11

Enumeration teEvent

The enumeration teEvent specifies the severity of diagnosis messages.

Name	Type	Initial	Comment
eDiagEvent_Occured	DINT	1	Error occurred, remains "active" until eDiagEvent_Disappeared
eDiagEvent_Disappeared	DINT	2	Error disappeared, became "active" due to eDiagEvent_Occured earlier on
eDiagEvent_Received	DINT	4	Received a diagnosis message which cannot be analyzed in detail, cannot disappear, needs to be acknowledged
eDiagEvent_Acknowledged	DINT	8	Acknowledge a diagnosis message which has been received by eDiagEvent_Received, removes diagnosis message from diagnosis system although error may still be present

Enumeration teHwId

The enumeration teHwId specifies the hardware component as location of a diagnosis message.

Name	Initial	Comment
eDiagHwId_CPU	0	Any diagnosis message regarding CPU itself, like "battery", "sd card", etc.
eDiagHwId_Coupler1	1	Any diagnosis message regarding communication module at slot 1. May be indicated by any corresponding communication module driver (instance) or a protocol driver (instance)
eDiagHwId_Coupler2	2	Any diagnosis message regarding communication module at slot 2. May be indicated by any corresponding communication module driver (instance) or a protocol driver (instance)
eDiagHwId_Coupler3	3	Any diagnosis message regarding communication module at slot 3. May be indicated by any corresponding communication module driver (instance) or a protocol driver (instance)
eDiagHwId_Coupler4	4	Any diagnosis message regarding communication module at slot 4. May be indicated by any corresponding communication module driver (instance) or a protocol driver (instance)
eDiagHwId_Coupler5	5	Any diagnosis message regarding communication module at slot 5. May be indicated by any corresponding communication module driver (instance) or a protocol driver (instance)
eDiagHwId_Coupler6	6	Any diagnosis message regarding communication module at slot 6. May be indicated by any corresponding communication module driver (instance) or a protocol driver (instance)
eDiagHwId_COM1	7	Any diagnosis message regarding COM1. May be indicated by any corresponding interface driver (instance) or a protocol driver (instance)
eDiagHwId_COM2	8	Any diagnosis message regarding COM2. May be indicated by any corresponding interface driver (instance) or a protocol driver (instance)
eDiagHwId_CAN	9	Any diagnosis message regarding CAN interface. May be indicated by any corresponding interface driver (instance) or a protocol driver (instance)
eDiagHwId_OnboardIO	10	Onboard I/O, AC500-eCo only
eDiagHwId_OptionBoard1	11	Option board 1, AC500-eCo only
eDiagHwId_OptionBoard2	12	Option board 2, AC500-eCo only
eDiagHwId_OptionBoard3	13	Option board 3, AC500-eCo only
eDiagHwId_IOBus	14	I/O bus
eDiagHwId_ETH1	15	Any diagnosis message regarding ETH1. May be indicated by any corresponding interface driver (instance) or a protocol driver (instance)
eDiagHwId_ETH2	16	Any diagnosis message regarding ETH2. May be indicated by any corresponding interface driver (instance) or a protocol driver (instance)

The hardware ID HwId is only used for diagnosis output in CPU display to identify the location of a diagnosis message ↳ *Chapter 1.7.1.2 “Diagnosis in CPU display” on page 4013*.

1.7.1.4.2 System diagnosis

Library AC500_Diag provides several methods and functions to access the diagnosis messages on all devices in the PLC application. It contains also a function to convert numeric diagnosis into a textual format.

The variables and their assigned values can be referred to within the IEC application as well as they can be used to transfer diagnosis messages to any visualization client.

Device state

The library contains a single function block named "Diag", providing several methods to process the device state.

Method	Description
NumTotal	Provides the total number of currently active diagnosis messages
NumClass	Provides the number of currently active diagnosis messages related to the error severity

Method NumTotal

This method provides the total number of currently active diagnosis messages (including parameter errors, etc.).

Scope	Name	Type	Comment
Return	NumTotal	DWORD	Number of diagnosis messages

Example

```
VAR
        fbDiag : AC500_Diag.Diag;   (* instance of FB Diag *)
        dwNumTotal : DWORD;
END_VAR
_____________________________________________________________________________
dwNumTotal := fbDiag.NumTotal();
```

Method NumClass

This method provides the number of currently active diagnosis messages related to the error severity.

Scope	Name	Type	Initial	Comment
Return	NumClass	DWORD		
Input	DataInVal	AC500_DiagTypes.teClass	eClass.eDiagClass_2_SeriousError	Error severity of diagnosis message

Example

```
VAR
        fbDiag : AC500_Diag.Diag;
        eDiagClass_4: AC500_DiagTypes.teClass := AC500_DiagTypes.teClass.eDiagClass_4_Warning; (* 4 *)
        eDiagClass_3: AC500_DiagTypes.teClass := AC500_DiagTypes.teClass.eDiagClass_3_Error; (* 3 *)
        eDiagClass_11: AC500_DiagTypes.teClass := AC500_DiagTypes.teClass.eDiagClass_Parameter; (* 11 *)
        dwNumClass_4 : DWORD;
        dwNumClass_3 : DWORD;
        dwNumClass_11 : DWORD;
END_VAR
```

```
dwNumClass_4 := fbDiag.NumClass(eDiagClass := eDiagClass_4);
dwNumClass_3 := fbDiag.NumClass(eDiagClass := eDiagClass_3);
dwNumClass_11 := fbDiag.NumClass(eDiagClass := eDiagClass_11);
```

Diagnosis descriptions

The library contains a single function block "Diag", providing several methods to process diagnosis descriptions.

All methods for system diagnosis start with "Get...". For device diagnosis the prefix "Diag" is added: "DiagGet...". For better readability, only the method names for system diagnosis is used in the descriptions of the methods.

Table 750: Methods for handling diagnosis entries

Method for system diagnosis	Method for device diagnosis	Description
Ack	DiagAck	Acknowledge a diagnosis alarm previously requested by using any `Get...` / `DiagGet...` method ↳ *Chapter 1.7.1.4.2.2.1 "Method Ack / DiagAck: acknowledgement" on page 4029*
GetFirstVal	DiagGetFirstVal	Get the first (oldest) diagnosis message, numeric values ↳ *Chapter 1.7.1.4.2.2.2 "Methods Get... / DiagGet...: get and sort diagnosis messages" on page 4029*
GetNextVal	DiagGetNextVal	Get the next diagnosis message, numeric values ↳ *Chapter 1.7.1.4.2.2.3 "Method Get-xxx-Val / DiagGet-xxx-Val: numeric values" on page 4030*
GetLastVal	DiagGetLastVal	Get the last (newest) diagnosis message, numeric values ↳ *Chapter 1.7.1.4.2.2.3 "Method Get-xxx-Val / DiagGet-xxx-Val: numeric values" on page 4030*
GetPrevVal	DiagGetPrevVal	Get the previous diagnosis message, numeric values ↳ *Chapter 1.7.1.4.2.2.3 "Method Get-xxx-Val / DiagGet-xxx-Val: numeric values" on page 4030*
GetFirstValExt	DiagGetFirstValExt	Get the first (oldest) diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.2.2.4 "Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values" on page 4030*
GetNextValExt	DiagGetNextValExt	Get the next diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.2.2.4 "Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values" on page 4030*
GetLastValExt	Diag GetLastValExt	Get the last (newest) diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.2.2.4 "Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values" on page 4030*
GetPrevValExt	DiagGetPrevValExt	Get the previous diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.2.2.4 "Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values" on page 4030*
GetFirstValAndTxt	DiagGetFirstValAndTxt	Get the first (oldest) diagnosis message, numeric values and text ↳ *Chapter 1.7.1.4.2.2.5 "Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text" on page 4031*

Method for system diagnosis	Method for device diagnosis	Description
GetNextValAndTxt	DiagGetNextValAndTxt	Get the next diagnosis message, numeric values and text *Chapter 1.7.1.4.2.2.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4031*
GetLastValAndTxt	DiagGetLastValAndTxt	Get the last (newest) diagnosis message, numeric values and text *Chapter 1.7.1.4.2.2.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4031*
GetPrevValAndTxt	DiagGetPrevValAndTxt	Get the previous diagnosis message, numeric values and text *Chapter 1.7.1.4.2.2.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4031*
GetFirstValAndTxtExt	DiagGetFirstValAndTxtExt	Get the first (oldest) diagnosis message, numeric, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*
GetNextValAndTxtExt	DiagGetNextValAndTxtExt	Get the next diagnosis message, numeric, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*
GetLastValAndTxtExt	DiagGetLastValAndTxtExt	Get the last (newest) diagnosis message, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*
GetPrevValAndTxtExt	DiagGetPrevValAndTxtExt	Get the previous diagnosis message, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*

Method Ack / DiagAck: acknowledgement

This method can be used to acknowledge a diagnosis alarm previously requested by using any `Get...` / `DiagGet...` method. Alternatively, you can acknowledge an alarm in Automation Builder.

After acknowledgement, the alarm is deleted from the diagnosis system.

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Ack	DiagAck	AC500_DiagTypes.ERROR_ID	
Input	Data	Data	AC500_DiagTypes.DIAG_VAL_TYPE	Variable containing details of diagnosis alarm to be acknowledged

Example

System diagnosis: acknowledge first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                   (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;  (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;  (* structure of returned values *)
END_VAR
```

```
eErrorID_First := fbDiag.Ack(Data := sFirstVal);
```

Methods Get... / DiagGet...: get and sort diagnosis messages

All these methods can be used to get the first (oldest), next, last (newest) or previous diagnosis message stored in diagnosis system. The only difference are the details the methods provide. While, e.g., `Get-xxx-Val` just provides the basic information in numeric format, `Get-xxx-ValExt` additionally provides this information by the extended diagnosis data of the entry.

The numeric format provided by these methods can be converted into textual format later on if required ↳ *Chapter 1.7.1.4.2.2.7 “Function DiagValToTxt” on page 4033*. Alternatively, the methods `Get-xxx-ValAndTxt` and `Get-xxx-ValAndTxtExt` can be used for numeric and textual format in parallel ↳ *Chapter 1.7.1.4.2.2.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4031* ↳ *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*.

All methods may need multiple cycles to process the request. Therefore, they must be called in successive cycles until they return a final result ↳ *Chapter 1.7.1.4.1.3 “Enumeration ERROR_ID” on page 4022*.

All diagnosis messages sorted by time, ascending

1. Call any `GetFirst...` method until it indicates a final result.
2. If the result is not "NO_ERROR_NO_DATA": Call any `GetNext...` method as long as its final result is "NO_ERROR".

All diagnosis messages sorted by time, descending

1. Call any `GetLast...` method until it indicates a final result.
2. If the result is not "NO_ERROR_NO_DATA": Call any `GetPrev...` method as long as its final result is "NO_ERROR".

Method Get-xxx-Val / DiagGet-xxx-Val: numeric values

-xxx- = First, Next, Last, Prev. Example: GetFirstVal, DiagGetLastVal.

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Get-xxx-Val	DiagGet-xxx-Val	AC500_DiagTypes.ERROR_ID	
Inout	Data	Data	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to write data to

Example System diagnosis: get values for first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                   (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;  (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;  (* structure of returned values *)
END_VAR

eErrorID_First := fbDiag.GetFirstVal(Data := sFirstVal);
```

Example Online mode: battery empty or missing

PLC_PRG

PLC_AC500_V3.Application.PLC_PRG

Expression	Type	Value	Prepared value	Address	Comment
fbDiag	AC500_Diag.Diag				instance of FB Diag
eErrorID_First	ERROR_ID	NO_ERROR			return value
sFirstVal	AC500_Diag.AC500...				structure of returend values
dtTimestamp	DATE_AND_TIME	DT#1970-1-1-0:0:15			RTC time of event
uiMs	UINT	356			Milliseconds of event
eClass	TECLASS	eDiagClass_4_Warning			Severity of error event
szDevice	STRING(80)	'PLC_AC500_V3'			Name of device
eEvent	TEEVENT	eDiagEvent_Occurred			Type of event
eHwInterfac...	TEHWID	eDiagHwId_CPU			Identifier of hardware interface
dwSubSyste...	DWORD	369098752			Any number describin...ail/location within d...
dwAdditional	DWORD	0			Additional number des...ng detail/location wi...
udErrorCode	UDINT	8			Error code
uiSizeExtDiag	UINT	0			Number of bytes of extended diagnosis data
hSource	POINTER TO BYTE	16#008703D0			internal reference needed for text convertion
pConn	POINTER TO IoStan...	16#B134A1D4			internal reference needed for text convertion

```
1  eErrorID_First  NO_ERROR  := fbDiag.GetFirstVal(Data := sFirstVal);
```

Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values

-xxx- = First, Next, Last, Prev. Example: GetNextValExt, DiagGetPrevValExt.

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Get-xxx-ValExt	DiagGet-xxx-ValExt	AC500_DiagTypes.ERROR_ID	
Inout	Data	Data	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to write data to
Input	pExt	pExt	POINTER TO BYTE	Address of buffer to copy extended data to
Input	Size	Size	WORD	Size of buffer to copy extended data to
Inout	Length	Length	WORD	Size of extended data copied to buffer

Example System diagnosis: get numeric values and extended numeric values for first diagnosis message

```
VAR
    fbDiag : AC500_Diag.Diag;                    (* instance of FB Diag *)
    eErrorID_First : AC500_DiagTypes.ERROR_ID;   (* return value *)
    sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;   (* structure of returned values *)
    abyExtData : ARRAY[0..1023] OF BYTE;         (* byte array for extended diagnosis data *)
    wSize : WORD := 1024;                        (* Size of buffer to copy extended data to *)
    wLength : WORD;                              (* Size of extended data copied to buffer *)
END_VAR
---------------------------------------------------------------------------------------------
eErrorID_First := fbDiag.GetFirstValExt(Data := sFirstVal, pExt := ADR(abyExtData), Size := wSize, Length := wLength);
```

Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text

-xxx- = First, Next, Last, Prev. Example: GetFirstValAndTxt, DiagGetPrevValAndTxt

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Get-xxx-ValAndTxt	DiagGet-xxx-ValAndTxt	AC500_DiagTypes.ERROR_ID	
Inout	DataVal	DataVal	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to write data to
Inout	DataTxt	DataTxt	AC500_DiagTypes.DIAG_TXT_TYPE	Variable to write text to

Example System diagnosis: get numeric values and text for first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                   (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;  (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;  (* structure of returned values *)
        sFirstTxt : AC500_DiagTypes.DIAG_TXT_TYPE;  (* structure of returned text *)
END_VAR
```

```
eErrorID_First := fbDiag.GetFirstValAndTxt(DataVal := sFirstVal, DataTxt := sFirstTxt);
```

Example Online mode: battery empty or missing

PLC_PRG_1

PLC_AC500_V3.Application.PLC_PRG_1

Expression	Type	Value
fbDiag	AC500_Diag.Diag	
eErrorID_First	ERROR_ID	NO_ERROR
sFirstVal	AC500_Diag.AC500...	
dtTimestamp	DATE_AND_TIME	DT#1970-1-1-0:0:15
uiMs	UINT	356
eClass	TECLASS	eDiagClass_4_Warning
szDevice	STRING(80)	'PLC_AC500_V3'
eEvent	TEEVENT	eDiagEvent_Occurred
eHwInterfac...	TEHWID	eDiagHwId_CPU
dwSubSyste...	DWORD	369098752
dwAdditional	DWORD	0
udErrorCode	UDINT	8
uiSizeExtDiag	UINT	0
hSource	POINTER TO BYTE	16#008703D0
pConn	POINTER TO IoStan...	16#B134A1D4
sFirstTxt	AC500_Diag.AC500...	
szDiag	STRING(512)	'1970-01-01;00:00:15,356;E4;PLC_AC500_V3;8;Battery Device Empty or missing'
szColumn	STRING(1)	';'

```
1  eErrorID_First NO_ERROR := fbDiag.GetFirstValAndTxt(DataVal := sFirstVal, DataTxt := sFirstTxt);
```

Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text

-xxx- = First, Next, Last, Prev. Example: GetLastValAndTxtExt, DiagGetFirstValAndTxtExt

Scope	Name for device diagnosis	Name for device diagnosis	Type	Initial	Comment
Return	Get-xxx-Val-AndTxtExt	DiagGet-xxx-Val-AndTxtExt	AC500_Dia-gTypes.ERROR_ID		
Inout	DataVal	DataVal	AC500_Dia-gTypes.DIAG_VAL_TYPE		Variable to write data to

Scope	Name for device diagnosis	Name for device diagnosis	Type	Initial	Comment
Inout	DataTxt	DataTxt	AC500_Dia-gTypes.DIAG_TXT_TYPE		Variable to write text to
Input	pExt	pExt	POINTER TO BYTE	0	Address of buffer to copy extended data to
Input	Size	Size	WORD	0	Size of buffer to copy extended data to
Inout	Length	Length	WORD	0	Size of extended data copied to buffer

Example

System diagnosis: get numeric values, extended numeric values and text of first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                       (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;      (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;      (* structure of returned values *)
        sFirstTxt : AC500_DiagTypes.DIAG_TXT_TYPE;      (* structure of returned text *)
        abyExtData : ARRAY[0..1023] OF BYTE;            (* byte array for extended diagnosis data *)
        wSize : WORD := 1024;                           (* Size of buffer to copy extended data to *)
        wLength : WORD;                                 (* Size of extended data copied to buffer *)
END_VAR
-------------------------------------------------------------------------------------------------------
eErrorID_First := fbDiag.GetFirstValExt(DataVal := sFirstVal, DataTxt := sFirstTxt,
                                        pExt := ADR(abyExtData), Size := wSize, Length := wLength);
```

Function DiagValToTxt

Call this function to convert a numeric diagnosis message into a textual one at any time, in case this has not yet been done using a method providing both types when requesting this information.

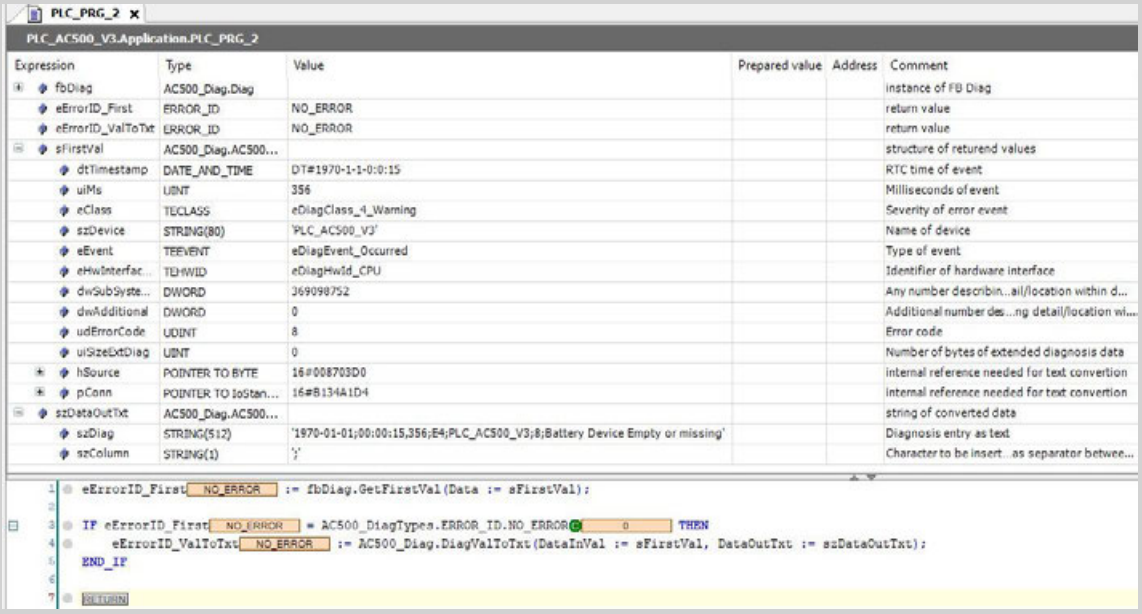
Scope	Name	Type	Comment
Return	DiagValToTxt	AC500_DiagTypes.ERROR_ID	
Inout	DataInVal	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to convert
Inout	DataOutTxt	AC500_DiagTypes.DIAG_TXT_TYPE	Variable to write text to

Example System diagnosis: convert first diagnosis message from numeric value to text

```
VAR
        eErrorID_ValToTxt : AC500_DiagTypes.ERROR_ID;        (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;           (* structure of returned values *)
        szDataOutTxt : AC500_DiagTypes.DIAG_TXT_TYPE;        (* string of converted data *)
END_VAR
```

```
eErrorID_ValToTx := DiagValToTxt(DataInVal := sFirstVal, DataOutTxt := szDataOutTxt);
```

Example Battery empty or missing



1.7.1.4.3 Device diagnosis

Activate device diagnosis

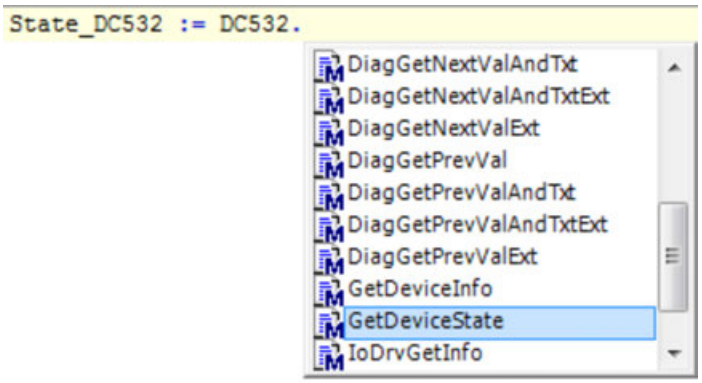
While the notification of diagnosis messages at the display and the Automation Builder is enabled by default, the functionality to access diagnosis messages from within the IEC application needs to be explicitly enabled.

1. Double-click on the CPU in the device tree.
2. Select tab *“PLC Settings”*.
3. Under *“Additional Settings”* select *“Enable Diagnosis for devices”*.
   ⇨ Library CAA Device Diagnosis (namespace DED) is automatically included in the project. This library is needed for displaying and processing the device state.

In case the functionality of diagnosis is no longer needed in IEC application, we recommend to disable this setting.

Device state

1. Open one of the IEC application code editors.
2. Type the device's name as it is written in the device tree, followed by a dot (".").
3. Select the method `GetDeviceState` from the context menu or type the name of the method on yourself.



4. Assign the function's parameters.

```
eError : DED.ERROR;                          (* return value of call *)
bDiagAvailable : BOOL;                       (* at least one diagnosis avilable for this device *)
DeviceState_DC532 : DED.DEVICE_STATE;        (* general device state *)

DeviceState_DC532 := DC532.GetDeviceState(xDiagnosisInfoAvailable => bDiagAvailable, eError => eError);
```

Scope	Name	Type	Comment
Return	GetDeviceState	DEVICE_STATE ↳ *Further information on page 4035*	Current device state
Output	xDiagnosisInfoAvailable	BOOL	If TRUE, diagnosis messages are available regarding the concerning device (= node).
Output	eError	ERROR ↳ *Further information on page 4036*	Type of the return values of all methods and functions of library CAA Device Diagnosis

Table 751: Enumeration DEVICE_STATE (part of the library CAA Device Diagnosis (DED))

Name	Type	Initial	Icon in AB	Comment
UNKNOWN	INT	0		The device is in state unknown. Example: No supervision mechanism active
STOPPED	INT	1		The device is stopped.
RUNNING	INT	2		The device is running.
ERROR	INT	3		The device is in error state.
DISABLED	INT	4		The device is disabled in device tree.

Name	Type	Initial	Icon in AB	Comment
NOT_CONFIG-URED	INT	5		The device has not been yet configured by the stack. Example: Configuration phase not yet started
CONFIGURED	INT	6		The device has been configured by the stack. Example: Configuration phase finished but the device is not in running state
NOT_FOUND	INT	7		The device was not found on bus.

Table 752: Enumeration ERROR (part of the library CAA Device Diagnosis (DED))

Name	Type	Initial	Comment
NO_ERROR	INT	0	No error
FIRST_ERROR	INT	1300	First library-specific error
TIME_OUT	INT	1301	Timeout occured.
ABORT	INT	1302	Operation was aborted.
REF_INVALID	INT	1303	The interface reference was invalid.
NOT_SUPPORTED	INT	1304	The function is not supported.
ERROR_IO	INT	1305	A general I/O configuration error occured.
PARAM_INVALID	INT	1306	Invalid parameter
NODE_NOT_EXISTING	INT	1307	The specified node does not exist.
NO_MEMORY	INT	1308	Dynamic memory allocation is disabled, or system is out of memory.
ADR_NOT_FOUND	INT	1309	The specified I/O address is not valid.
INST_NOT_FOUND	INT	1310	There is no associated [Device] instance for the specific I/O address.
NO_DATA	INT	1311	There is no data available.
OPERATION_INVALID	INT	1312	Operation not possible due to the current state
FIRST_MF	INT	1350	First manufacturer-specific error
LAST_ERROR	INT	1399	Last error

Diagnosis descriptions

The library contains a single function block "Diag", providing several methods to process diagnosis descriptions.

All methods for system diagnosis start with "Get...". For device diagnosis the prefix "Diag" is added: "DiagGet...". For better readability, only the method names for system diagnosis is used in the descriptions of the methods.

Table 753: Methods for handling diagnosis entries

Method for system diagnosis	Method for device diagnosis	Description
Ack	DiagAck	Acknowledge a diagnosis alarm previously requested by using any `Get...` / `DiagGet...` method ↳ *Chapter 1.7.1.4.3.3.1 “Method Ack / DiagAck: acknowledgement” on page 4039*
GetFirstVal	DiagGetFirstVal	Get the first (oldest) diagnosis message, numeric values ↳ *Chapter 1.7.1.4.3.3.2 “Methods Get... / DiagGet...: get and sort diagnosis messages” on page 4039*
GetNextVal	DiagGetNextVal	Get the next diagnosis message, numeric values ↳ *Chapter 1.7.1.4.3.3.3 “Method Get-xxx-Val / DiagGet-xxx-Val: numeric values” on page 4040*
GetLastVal	DiagGetLastVal	Get the last (newest) diagnosis message, numeric values ↳ *Chapter 1.7.1.4.3.3.3 “Method Get-xxx-Val / DiagGet-xxx-Val: numeric values” on page 4040*
GetPrevVal	DiagGetPrevVal	Get the previous diagnosis message, numeric values ↳ *Chapter 1.7.1.4.3.3.3 “Method Get-xxx-Val / DiagGet-xxx-Val: numeric values” on page 4040*
GetFirstValExt	DiagGetFirstValExt	Get the first (oldest) diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.3.3.4 “Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values” on page 4040*
GetNextValExt	DiagGetNextValExt	Get the next diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.3.3.4 “Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values” on page 4040*
GetLastValExt	Diag GetLastValExt	Get the last (newest) diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.3.3.4 “Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values” on page 4040*
GetPrevValExt	DiagGetPrevValExt	Get the previous diagnosis message, numeric and extended numeric values ↳ *Chapter 1.7.1.4.3.3.4 “Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values” on page 4040*
GetFirstValAndTxt	DiagGetFirstValAndTxt	Get the first (oldest) diagnosis message, numeric values and text ↳ *Chapter 1.7.1.4.3.3.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4041*

Method for system diagnosis	Method for device diagnosis	Description
GetNextValAndTxt	DiagGetNextValAndTxt	Get the next diagnosis message, numeric values and text *Chapter 1.7.1.4.3.3.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4041*
GetLastValAndTxt	DiagGetLastValAndTxt	Get the last (newest) diagnosis message, numeric values and text *Chapter 1.7.1.4.3.3.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4041*
GetPrevValAndTxt	DiagGetPrevValAndTxt	Get the previous diagnosis message, numeric values and text *Chapter 1.7.1.4.3.3.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4041*
GetFirstValAndTxtExt	DiagGetFirstValAndTxtExt	Get the first (oldest) diagnosis message, numeric, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*
GetNextValAndTxtExt	DiagGetNextValAndTxtExt	Get the next diagnosis message, numeric, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*
GetLastValAndTxtExt	DiagGetLastValAndTxtExt	Get the last (newest) diagnosis message, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*
GetPrevValAndTxtExt	DiagGetPrevValAndTxtExt	Get the previous diagnosis message, extended numeric values and text *Chapter 1.7.1.4.2.2.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4032*

Method Ack / DiagAck: acknowledgement

This method can be used to acknowledge a diagnosis alarm previously requested by using any `Get...` / `DiagGet...` method. Alternatively, you can acknowledge an alarm in Automation Builder.

After acknowledgement, the alarm is deleted from the diagnosis system.

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Ack	DiagAck	AC500_DiagTypes.ERROR_ID	
Input	Data	Data	AC500_DiagTypes.DIAG_VAL_TYPE	Variable containing details of diagnosis alarm to be acknowledged

Example

System diagnosis: acknowledge first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                   (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;  (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;  (* structure of returned values *)
END_VAR
```

```
eErrorID_First := fbDiag.Ack(Data := sFirstVal);
```

Methods Get... / DiagGet...: get and sort diagnosis messages

All these methods can be used to get the first (oldest), next, last (newest) or previous diagnosis message stored in diagnosis system. The only difference are the details the methods provide. While, e.g., `Get-xxx-Val` just provides the basic information in numeric format, `Get-xxx-ValExt` additionally provides this information by the extended diagnosis data of the entry.

The numeric format provided by these methods can be converted into textual format later on if required ↳ *Chapter 1.7.1.4.2.2.7 “Function DiagValToTxt” on page 4033*. Alternatively, the methods `Get-xxx-ValAndTxt` and `Get-xxx-ValAndTxtExt` can be used for numeric and textual format in parallel ↳ *Chapter 1.7.1.4.3.3.5 “Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text” on page 4041* ↳ *Chapter 1.7.1.4.3.3.6 “Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text” on page 4042*.

All methods may need multiple cycles to process the request. Therefore, they must be called in successive cycles until they return a final result ↳ *Chapter 1.7.1.4.1.3 “Enumeration ERROR_ID” on page 4022*.

All diagnosis messages sorted by time, ascending

1. Call any `GetFirst...` method until it indicates a final result.
2. If the result is not "NO_ERROR_NO_DATA": Call any `GetNext...` method as long as its final result is "NO_ERROR".

All diagnosis messages sorted by time, descending

1. Call any `GetLast...` method until it indicates a final result.
2. If the result is not "NO_ERROR_NO_DATA": Call any `GetPrev...` method as long as its final result is "NO_ERROR".

Method Get-xxx-Val / DiagGet-xxx-Val: numeric values

-xxx- = First, Next, Last, Prev. Example: GetFirstVal, DiagGetLastVal.

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Get-xxx-Val	DiagGet-xxx-Val	AC500_DiagTypes.ERROR_ID	
Inout	Data	Data	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to write data to

Example System diagnosis: get values for first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                   (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;  (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;  (* structure of returned values *)
END_VAR

eErrorID_First := fbDiag.GetFirstVal(Data := sFirstVal);
```

Example Online mode: battery empty or missing

PLC_PRG

PLC_AC500_V3.Application.PLC_PRG

Expression	Type	Value	Prepared value	Address	Comment
fbDiag	AC500_Diag.Diag				instance of FB Diag
eErrorID_First	ERROR_ID	NO_ERROR			return value
sFirstVal	AC500_Diag.AC500...				structure of returend values
dtTimestamp	DATE_AND_TIME	DT#1970-1-1-0:0:15			RTC time of event
uiMs	UINT	356			Milliseconds of event
eClass	TECLASS	eDiagClass_4_Warning			Severity of error event
szDevice	STRING(80)	'PLC_AC500_V3'			Name of device
eEvent	TEEVENT	eDiagEvent_Occurred			Type of event
eHwInterfac...	TEHWID	eDiagHwId_CPU			Identifier of hardware interface
dwSubSyste...	DWORD	369098752			Any number describin...ail/location within d...
dwAdditional	DWORD	0			Additional number des...ng detail/location wi...
udErrorCode	UDINT	8			Error code
uiSizeExtDiag	UINT	0			Number of bytes of extended diagnosis data
hSource	POINTER TO BYTE	16#008703D0			internal reference needed for text convertion
pConn	POINTER TO IoStan...	16#B134A1D4			internal reference needed for text convertion

```
1  eErrorID_First  NO_ERROR  := fbDiag.GetFirstVal(Data := sFirstVal);
```

Method Get-xxx-ValExt / DiagGet-xxx-ValExt: numeric values and extended numeric values

-xxx- = First, Next, Last, Prev. Example: GetNextValExt, DiagGetPrevValExt.

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Get-xxx-ValExt	DiagGet-xxx-ValExt	AC500_DiagTypes.ERROR_ID	
Inout	Data	Data	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to write data to
Input	pExt	pExt	POINTER TO BYTE	Address of buffer to copy extended data to
Input	Size	Size	WORD	Size of buffer to copy extended data to
Inout	Length	Length	WORD	Size of extended data copied to buffer

Example

System diagnosis: get numeric values and extended numeric values for first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                       (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;      (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;      (* structure of returned values *)
        abyExtData : ARRAY[0..1023] OF BYTE;            (* byte array for extended diagnosis data *)
        wSize : WORD := 1024;                           (* Size of buffer to copy extended data to *)
        wLength : WORD;                                 (* Size of extended data copied to buffer *)
END_VAR
-------------------------------------------------------------------------------------------------------
eErrorID_First := fbDiag.GetFirstValExt(Data := sFirstVal, pExt := ADR(abyExtData), Size := wSize, Length := wLength);
```

Method Get-xxx-ValAndTxt / DiagGet-xxx-ValAndTxt: numeric values and text

-xxx- = First, Next, Last, Prev. Example: GetFirstValAndTxt, DiagGetPrevValAndTxt

Scope	Name for device diagnosis	Name for device diagnosis	Type	Comment
Return	Get-xxx-ValAndTxt	DiagGet-xxx-ValAndTxt	AC500_DiagTypes.ERROR_ID	
Inout	DataVal	DataVal	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to write data to
Inout	DataTxt	DataTxt	AC500_DiagTypes.DIAG_TXT_TYPE	Variable to write text to

Example System diagnosis: get numeric values and text for first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                   (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;  (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;  (* structure of returned values *)
        sFirstTxt : AC500_DiagTypes.DIAG_TXT_TYPE;  (* structure of returned text *)
END_VAR
```

```
eErrorID_First := fbDiag.GetFirstValAndTxt(DataVal := sFirstVal, DataTxt := sFirstTxt);
```

Example Online mode: battery empty or missing

PLC_PRG_1

PLC_AC500_V3.Application.PLC_PRG_1

Expression	Type	Value
fbDiag	AC500_Diag.Diag	
eErrorID_First	ERROR_ID	NO_ERROR
sFirstVal	AC500_Diag.AC500...	
dtTimestamp	DATE_AND_TIME	DT#1970-1-1-0:0:15
uiMs	UINT	356
eClass	TECLASS	eDiagClass_4_Warning
szDevice	STRING(80)	'PLC_AC500_V3'
eEvent	TEEVENT	eDiagEvent_Occurred
eHwInterfac...	TEHWID	eDiagHwId_CPU
dwSubSyste...	DWORD	369098752
dwAdditional	DWORD	0
udErrorCode	UDINT	8
uiSizeExtDiag	UINT	0
hSource	POINTER TO BYTE	16#008703D0
pConn	POINTER TO IoStan...	16#B134A1D4
sFirstTxt	AC500_Diag.AC500...	
szDiag	STRING(512)	'1970-01-01;00:00:15,356;E4;PLC_AC500_V3;8;Battery Device Empty or missing'
szColumn	STRING(1)	';'

```
1  eErrorID_First NO_ERROR := fbDiag.GetFirstValAndTxt(DataVal := sFirstVal, DataTxt := sFirstTxt);
```

Method Get-xxx-ValAndTxtExt / DiagGet-xxx-ValAndTxtExt: numeric values, extended numeric values and text

-xxx- = First, Next, Last, Prev. Example: GetLastValAndTxtExt, DiagGetFirstValAndTxtExt

Scope	Name for device diagnosis	Name for device diagnosis	Type	Initial	Comment
Return	Get-xxx-Val-AndTxtExt	DiagGet-xxx-Val-AndTxtExt	AC500_DiagTypes.ERROR_ID		
Inout	DataVal	DataVal	AC500_DiagTypes.DIAG_VAL_TYPE		Variable to write data to

Scope	Name for device diagnosis	Name for device diagnosis	Type	Initial	Comment
Inout	DataTxt	DataTxt	AC500_Dia-gTypes.DIAG_TXT_TYPE		Variable to write text to
Input	pExt	pExt	POINTER TO BYTE	0	Address of buffer to copy extended data to
Input	Size	Size	WORD	0	Size of buffer to copy extended data to
Inout	Length	Length	WORD	0	Size of extended data copied to buffer

Example

System diagnosis: get numeric values, extended numeric values and text of first diagnosis message

```
VAR
        fbDiag : AC500_Diag.Diag;                       (* instance of FB Diag *)
        eErrorID_First : AC500_DiagTypes.ERROR_ID;      (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;      (* structure of returned values *)
        sFirstTxt : AC500_DiagTypes.DIAG_TXT_TYPE;      (* structure of returned text *)
        abyExtData : ARRAY[0..1023] OF BYTE;            (* byte array for extended diagnosis data *)
        wSize : WORD := 1024;                           (* Size of buffer to copy extended data to *)
        wLength : WORD;                                 (* Size of extended data copied to buffer *)
END_VAR
-------------------------------------------------------------------------------------------------------
eErrorID_First := fbDiag.GetFirstValExt(DataVal := sFirstVal, DataTxt := sFirstTxt,
                                        pExt := ADR(abyExtData), Size := wSize, Length := wLength);
```

Function DiagValToTxt

Call this function to convert a numeric diagnosis message into a textual one at any time, in case this has not yet been done using a method providing both types when requesting this information.

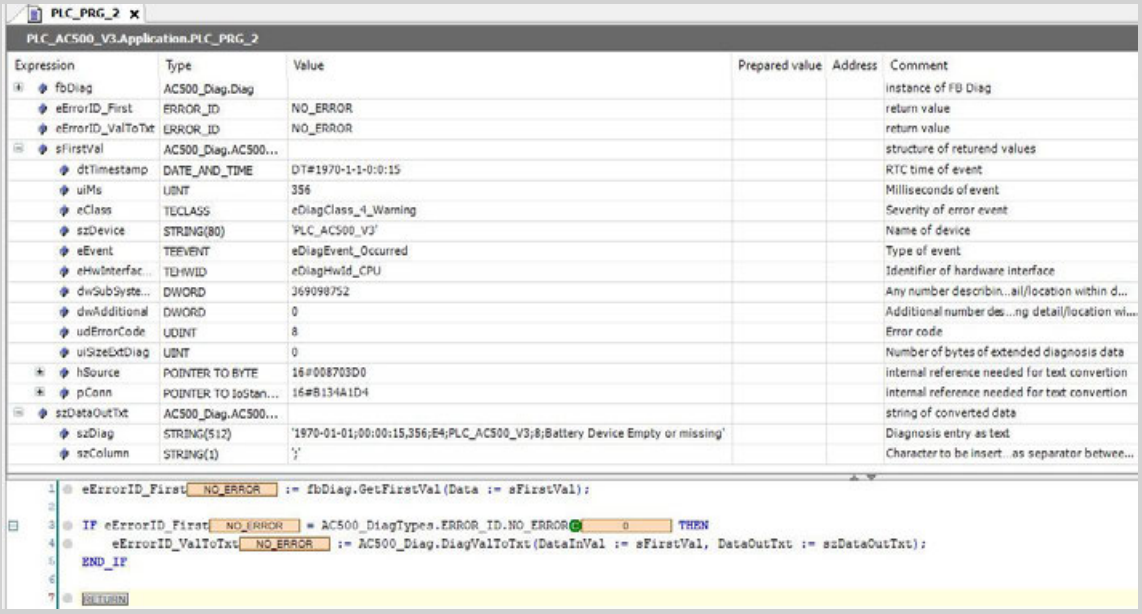
Scope	Name	Type	Comment
Return	DiagValToTxt	AC500_DiagTypes.ERROR_ID	
Inout	DataInVal	AC500_DiagTypes.DIAG_VAL_TYPE	Variable to convert
Inout	DataOutTxt	AC500_DiagTypes.DIAG_TXT_TYPE	Variable to write text to

Example System diagnosis: convert first diagnosis message from numeric value to text

```
VAR
        eErrorID_ValToTxt : AC500_DiagTypes.ERROR_ID;        (* return value *)
        sFirstVal : AC500_DiagTypes.DIAG_VAL_TYPE;           (* structure of returned values *)
        szDataOutTxt : AC500_DiagTypes.DIAG_TXT_TYPE;        (* string of converted data *)
END_VAR
```

```
eErrorID_ValToTx := DiagValToTxt(DataInVal := sFirstVal, DataOutTxt := szDataOutTxt);
```

Example Battery empty or missing



1.7.1.5 Structure of error numbers

1.7.1.5.1 Error severity

Error severity	Type	Description	Example
1	Fatal errors	Safe operation of the operating system is no longer ensured.	Checksum error in system flash, RAM error
2	Severe error	The operating system works correctly, but the error-free execution of the user program is not ensured.	Checksum error in user flash, task cycle times exceeded
3	Minor errors	It depends on the application whether the user program has to be stopped by the operating system or not. The user decides which reaction is to be done.	Flash memory cannot be programmed, I/O module failed

Error severity	Type	Description	Example
4	Warnings	Errors that occur on peripheral devices or that will have an effect only in the future. The user decides which reactions are to be done.	Short circuit in an I/O module, battery empty/not installed
11	Parameter error	Error occurred during parameter setting	Different I/O devices in PLC configuration and hardware installation

Errors with error severitiy 1 - fatal errors

Errors with error severity 1 are not entered in the diagnosis system. These errors do not allow normal operation of the PLC. These errors are detected during PLC start-up and stop the PLC immediately.

Examples are RAM errors or checksum errors when starting the firmware.

Such errors are indicated by rapid flashing of the ERR LED.

1.7.1.6 Diagnosis history file

Diagnosis history is available as of Automation Builder 2.4.0 / System FW 3.4.0 the diagnosis system has been extended with diagnosis history.

Diagnosis history is the entry of all diagnoses into a file according to their time of occurrence.

The diagnosis history file is in the root directory of the user disk and has the name *“DiagHistory.csv”*. The max. number of entries is 2000. When 2000 entries are reached, the oldest entry is overwritten. The max. size of the extended data is 32 bytes.

An entry consists of following data:

Name	Type	Comment	Example
timestamp	ARRAYDT OF BYTE	RTC time of event in milliseconds consists of diTimestamp in DT format and uiMs milliseconds. See *Chapter 1.7.1.4.1.1 “Structure DIAG_VAL_TYPE” on page 4021* STRUCT.	1603371910177
event	BYTE	Event type (1=comes, 2=gone). See *Chapter 1.7.1.4.1.5 “Enumeration teEvent” on page 4023*.	1
class	BYTE	Severity of error event. See *Chapter 1.7.1.4.1.4 “Enumeration teClass” on page 4023*.	4
compID	UDINT	Component ID	270540802
conn	UDINT	Connector	0xb17777ac
connIdx	UDINT	Connector index	0

Name	Type	Comment	Example
sub	DWORD	SubsystemID: Any number describing detail/location within device, device specific	369098752
addl	DWORD	AdditionalID: Additional number describing detail/location within device, optional, device specific	0
error	DWORD	Error code	9
extended data	ARRAYDT OF BYTE	Extended diagnosis data, max. 32 bytes	

As shown in the example data of the diagnosis history file is not easily readable. The entries must be interpreted according to device and/or fieldbus. Therefor the Automation Builder consists a special view for diagnosis history ↬ *Chapter 1.7.1.3.5 “Diagnosis history” on page 4019*.

With the entries CompID, conn and connID, the device generating the event is clearly identified in the device tree.

If the PLC configuration is changed, the values of this entries may be changed also.

Therefore, the diagnosis history will be deleted during each download.

1.7.2 Online diagnosis in Automation Builder

1.7.2.1 Short description and overview

To use the diagnosis system in Automation Builder, login to the online mode is required ↬ *Chapter 1.7.2.2 “Entering/leaving the online mode” on page 4046*. The online diagnosis in Automation Builder consists of a set of partly animated, mostly read only views. They can be invoked by a double-click on a project tree element which shows a circle indicating that this element is able to show diagnosis messages ↬ *Chapter 1.7.2.3 “Project tree in online mode” on page 4047*.

Available online diagnosis and statistics:

- **Diagnosis messages**
  When the Automation Builder is switched to online mode, incoming diagnosis messages are displayed as plain-text ↬ *Chapter 1.7.1.3 “Diagnosis in Automation Builder” on page 4017*.
- **CPU/PLC diagnosis**
  ↬ *Chapter 1.7.2.4 “CPU diagnosis views” on page 4051*.
- **I/O module diagnosis**
  ↬ *Chapter 1.7.2.5 “Live values in views with I/O components” on page 4056*.
- **Communication module and fieldbus diagnosis**
  ↬ *Chapter 1.7.2.6 “Communication module and fieldbus diagnosis” on page 4056*
- **Diagnosis in IEC application**
  ↬ *Chapter 1.7.1.4 “Diagnosis in IEC application” on page 4020*
  For information on the disk status, diagnosis information can be read out with the function blocks PmDiskStatus and PmDiskLifetimeUsed. ↬ *Chapter 1.6.7.4 “Health monitoring” on page 4010*

1.7.2.2 Entering/leaving the online mode

Prerequisite: Set the gateway before entering the online mode. ↬ *Chapter 1.6.6.2.15 “Gateway configuration” on page 3799*

Enter the online mode

Right-click the *“Application”* node and select *“Login”*.

The Automation Builder project login to online mode updates the latest changes of the project.

The online mode can be entered or left for each PLC in the project separately.

Leave the online mode

Right-click the *“Application”* node and select *Logout*.

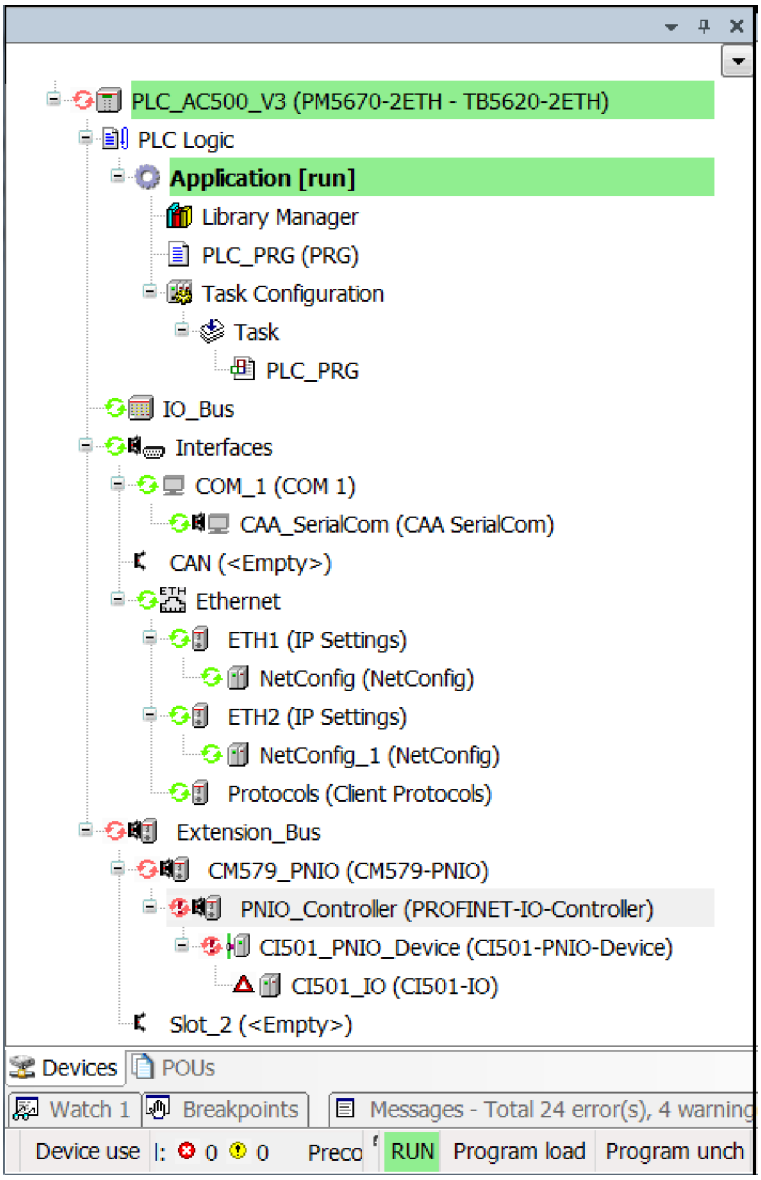
When online mode is active, a thread is running on Automation Builder project which sends cyclically a message to the PLC and expects a response. If the PLC does not respond, the online mode is left programmatically.

1.7.2.3 Project tree in online mode

When Automation Builder enters the online mode internally, it shows the state of all configured communication modules.

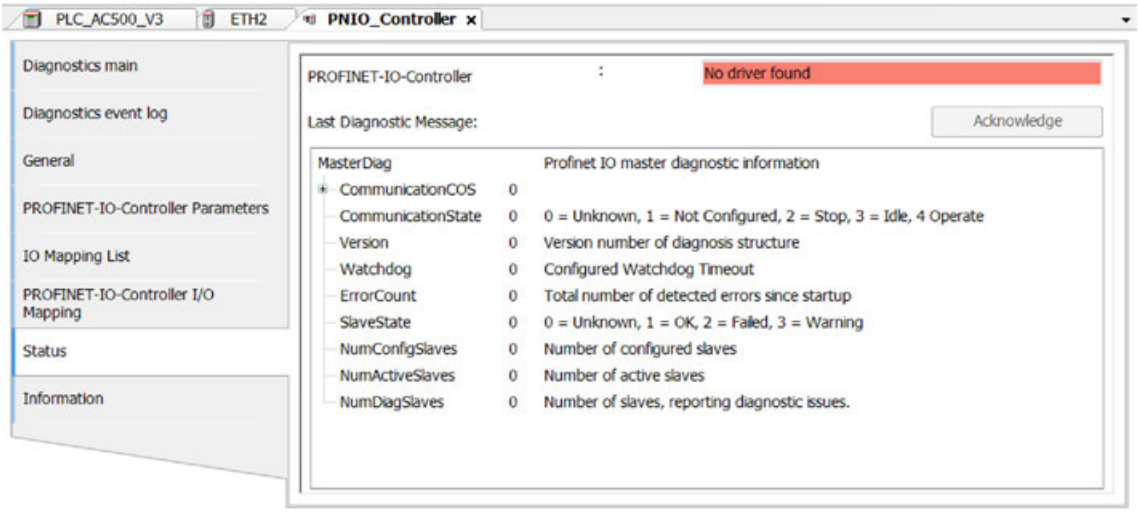
The connection status can be recognized by a symbol in the device tree:

- Device without diagnosis messages
- Device with diagnosis messages or device diagnosis is disabled *Chapter 1.7.2.4.6 “Device diagnosis” on page 4055*
- Device without diagnosis messages, but with diagnosis messages on at least one device in the branch below
- Device with diagnosis messages and with diagnosis messages on at least one device in the branch below
- Device does not respond to identification message and is not available for online connection

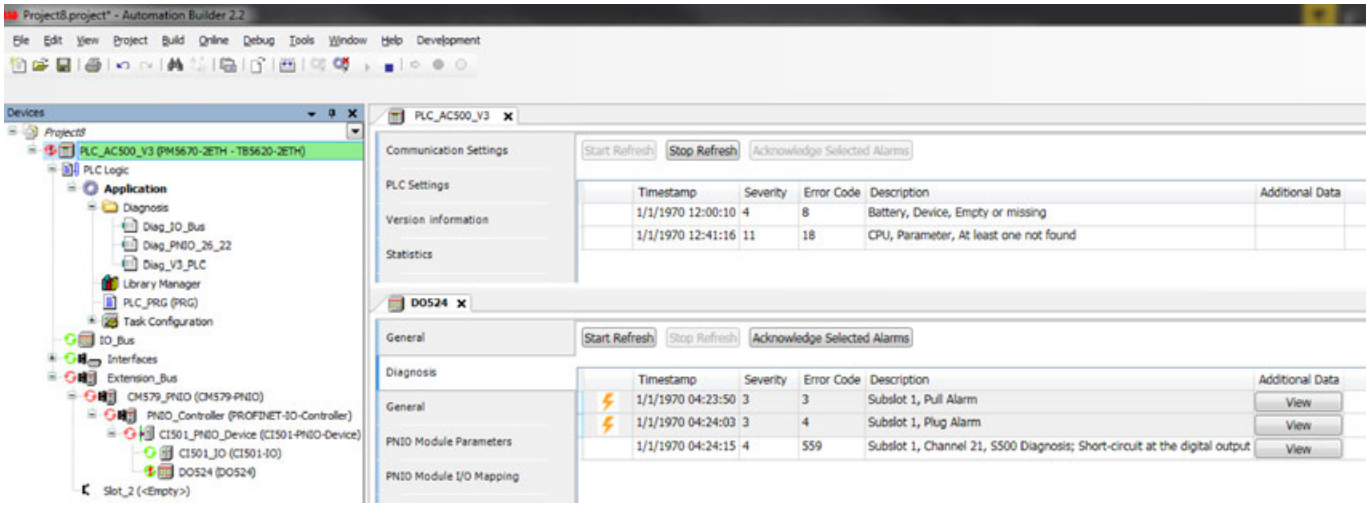


The identification is done in online mode.

- Double-click an element of the device-tree and select *"Status"* tab.
  Diagnosis information will be available.



Diagnosis descriptions

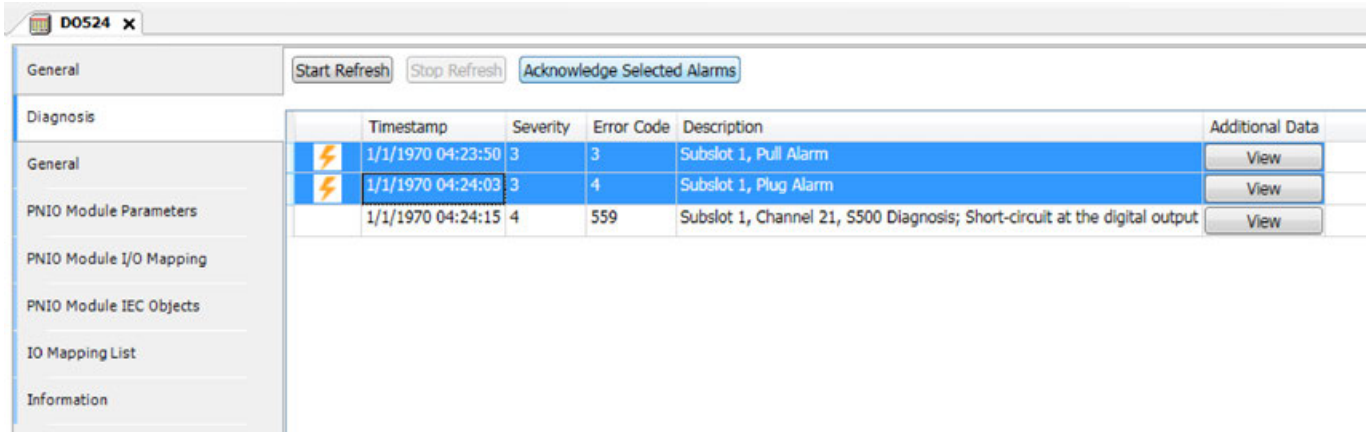


The user will be notified in the device tree with an exclamation mark beside the device having diagnosis messages. The diagnosis messages are provided in the *"Diagnosis"* tab.

Alarms will be presented with a thunderbolt in the first column of the diagnosis grid.

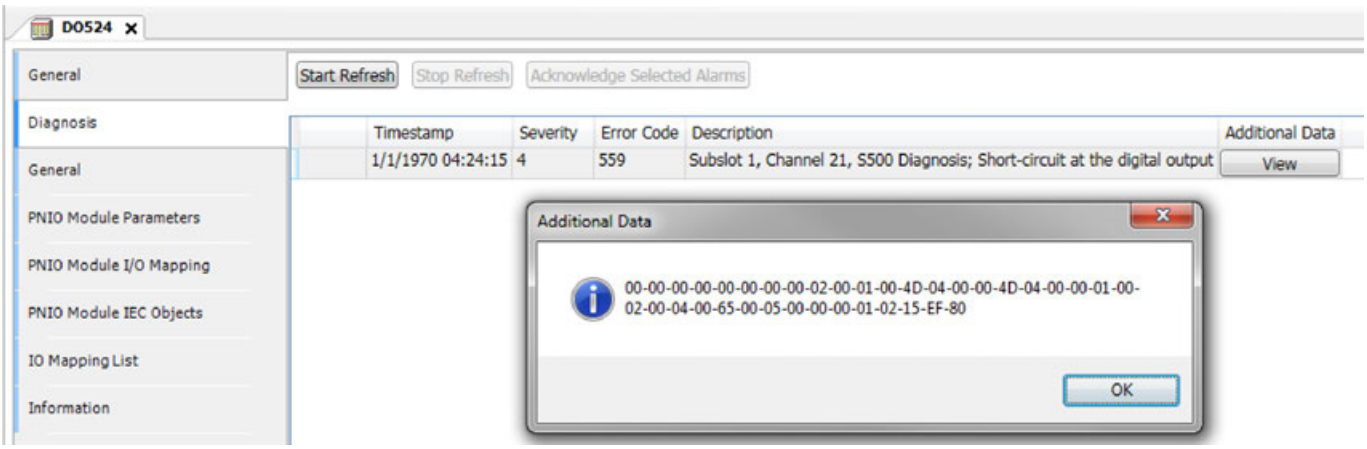
Acknowledging an alarm

1. Stop diagnosis refreshment by clicking *[Stop refresh]*.
2. Select one or more alarms and click *[Acknowledge selected alarms]*.

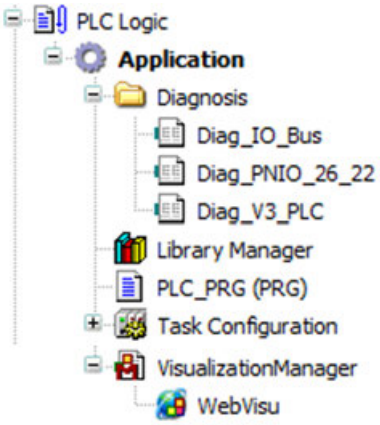


Some diagnosis messages contain additional data. Click *[View]* button to see the additional diagnosis (in hex) for further analysis. If *[View]* button is not available, no additional data is available for this diagnosis message.

You can copy the additional data to the clipboard with *[CTRL] + [C]*.



When building an IEC application in Automation Builder, diagnosis text lists will be generated and added to the device tree below the diagnosis folder. These text lists contain the device type specific diagnosis texts which are used by the diagnosis functions in the PLC application to show corresponding texts for error numbers.

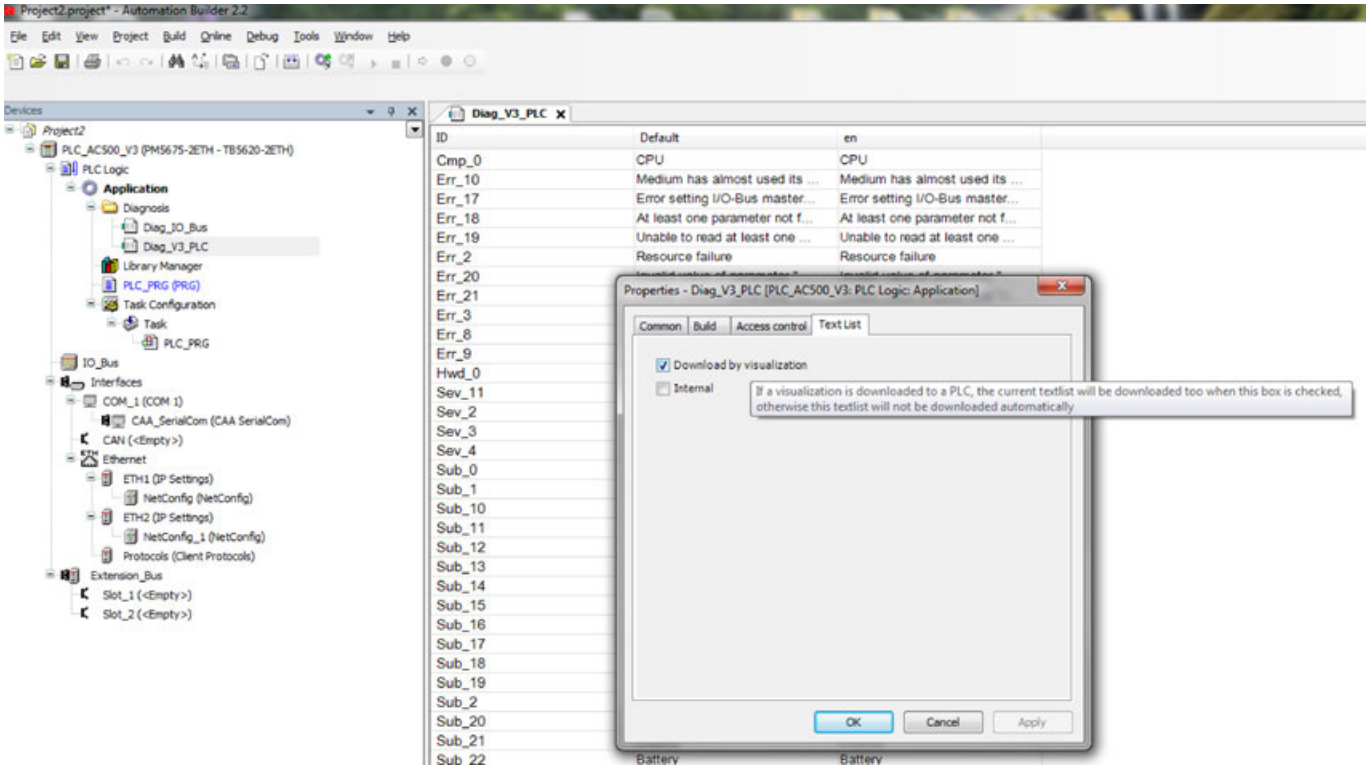


The diagnosis text lists will only be downloaded to the PLC when a visualization is present in the project.

The text lists will be downloaded automatically to the PLC with the visualization.

If there is a problem with downloading the text lists, make sure that the settings are correct:

1. Right-click on a text list and select *“Properties”*.
2. Open the *“Text List”* tab. The check box *“Download by visualization”* has to be selected.



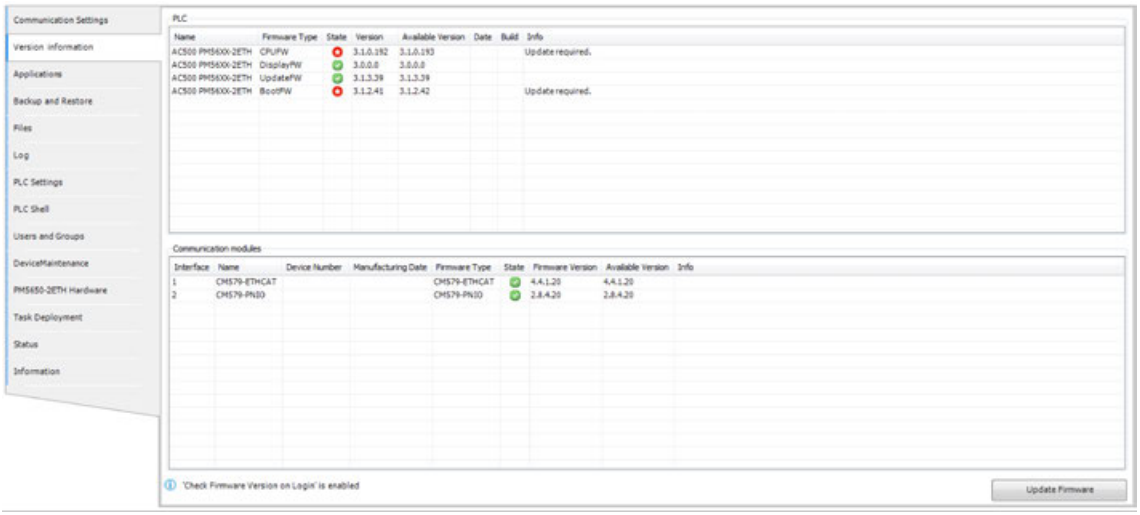
1.7.2.4 CPU diagnosis views

1.7.2.4.1 Version information

Information on the firmware versions of the processor modules or communication modules, is provided on the *“Version information”* tab.

Remarks:

- The *“Version information”* tab displays the version identified on the device and the version provided with Automation Builder.
- The firmware on the devices must match to the Automation Builder version. Upgrade or downgrade to version supplied with Automation Builder is recommended (especially for CPUs) to ensure correct functionality.
- The firmware type can be changed to the type required by the hardware configuration for devices that support changing the firmware type. E.g., the onboard field bus communication modules of PM595 that may be used as PROFINET, Ethernet or EtherCAT communication module.

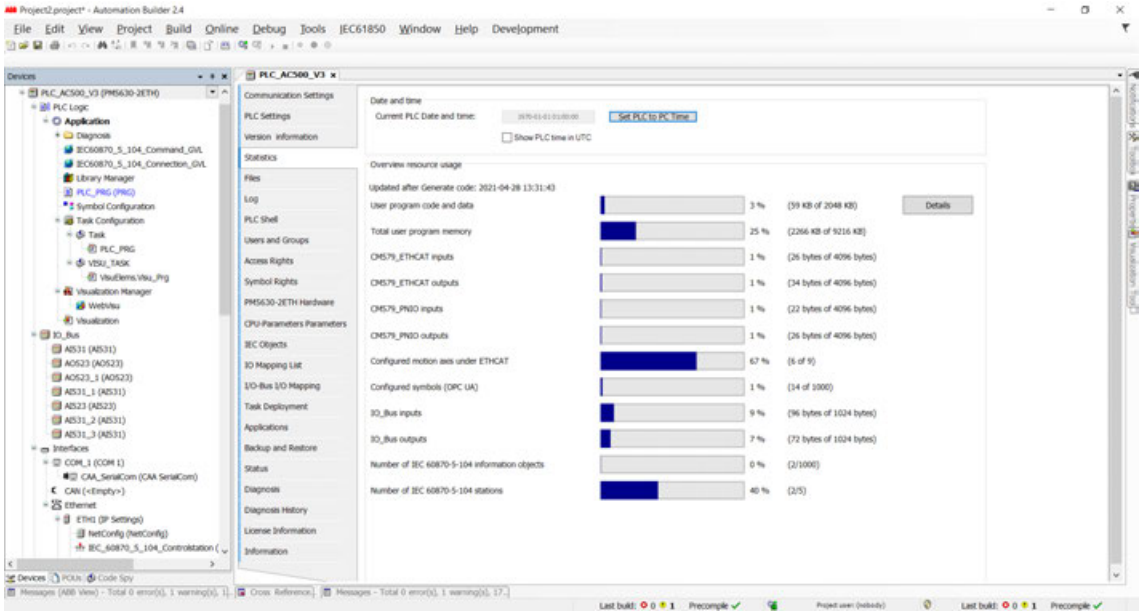


State icons

	Firmware version on device matches version supplied with Automation Builder.
	Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.
	Only for communication modules if CPU firmware must be updated first. This happens when CPU firmware has version below 2.5.0.0. Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.
	Identified device is different from configured device, thus no firmware update is possible. Happens only for Communication Modules.
No icon	Firmware of device is not updateable or no newer firmware than the initial version is available.

The [Update Firmware] button to download the new firmware is only enabled if there is updateable firmware.

1.7.2.4.2 Statistics



The *“Statistics”* tab shows the following information:

- *Date and time:* The actual date and time of the PLC is shown. It can be set or synchronized with the date/time of the PC via *“Set PLC Date & Time”* button.
- *Overview resource usage:* This tab shows all the required information (it is collected at latest when the command *“Generate Code”* is executed, some of the information is not available before then.)

  For the limitation *“User program code and data”* a *[Details]* button will be available. Clicking this button will open a modal window showing a more detailed view of the memory usage.

1.7.2.4.3 Log

You can view the PLC log in this tab. It lists the events that were recorded on the target system. This concerns:

- Events during the startup and shutdown of the system (components loaded, with version)
- Application download and loading of the boot application
- Custom entries
- Log entries from I/O drivers
- Log entries from data sources

Offline logging	☐: Default settings ☑: The PLC also records actions that are not related to the connection with the controller.
UTC time	☐: Standard setting; the time stamp is converted to the local time on the computer as indicated by the time zone of the operating system. ☑: The time stamp of the runtime system is displayed.

Severity	There are four categories for the severity of the event: ● : Message ● : Warning ● : Error ● : Debugging You can show or hide each category with the help of the corresponding button in the bar above the list. Each button shows the number of log entries of the category concerned.
Time stamp	Date and time (example: `12-01-2007 09:48`)
Description	Description of the event, for example `Import function failed of`
Component	Name of the runtime system component concerned, e.g. `CmpApp`
Drop-down list with component names	The log list displays only events that concern the selected component
Logger	Drop-down list with all available recordings. The standard setting is the <Default Logger> specified by the target system, at present identical to 'PlcLog' for the CODESYS runtime system
	Refreshes the log list
	Exports the list contents to an XML file. You can select the file name and storage directory.
	Imports a log list from an XML file. The list is then displayed in a separate window.
	The displayed log list is emptied, i.e. all entries are deleted.

1.7.2.4.4 PLC shell commands

The PLC shell is used for requesting specific information from the controller. By entering a device-specific command the response is returned in a result window. The PLC shell can be issued without login.

Proceed as follows:

1. Ensure the gateway is configured properly and a connection to the controller can be established.
2. In Automation Builder double-click the PLC node and open the tab *"PLC Shell"*.
3. Enter `"?"` in the command line of the tab window. All available PLC commands are listed.

If the gateway is able to establish a connection to the controller, an online connection to the PLC is opened automatically.

The commands listed in online mode can differ from the commands shown when pressing the button [...] as Automation Builder version and firmware version can differ.

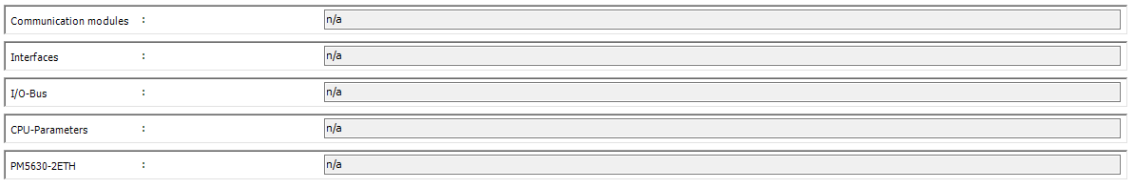
See:

Chapter 1.2.6 "Further information" on page 49

Chapter 1.6.6.1.4 "Firmware identification and update" on page 3652.

1.7.2.4.5 Status

This tab displays status information, for example 'Running' or 'Stopped', and specific diagnosis messages from the respective device, also information about the card used and the internal bus system.



1.7.2.4.6 Device diagnosis

Each node in the device tree has a diagnosis view, which displays the diagnosis messages for this device only.

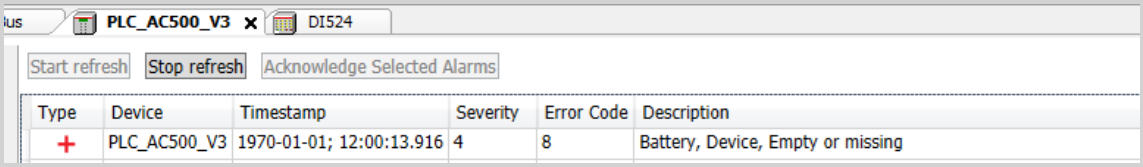
The message consists of:

- Type *Chapter 1.7.1.3.5 “Diagnosis history” on page 4019*
- Timestamp in date and time YYYY-MM-DD hh:mm:ss.ms
- Error severity
- Error code
- Diagnosis description
- Additional data

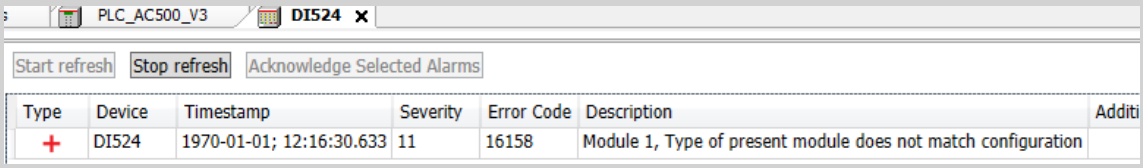
To view the diagnosis message:

1. Double-click on a device.
2. Select the tab *“Diagnosis”*.

Example Battery empty or missing.



Example Wrong module configured on I/O bus.



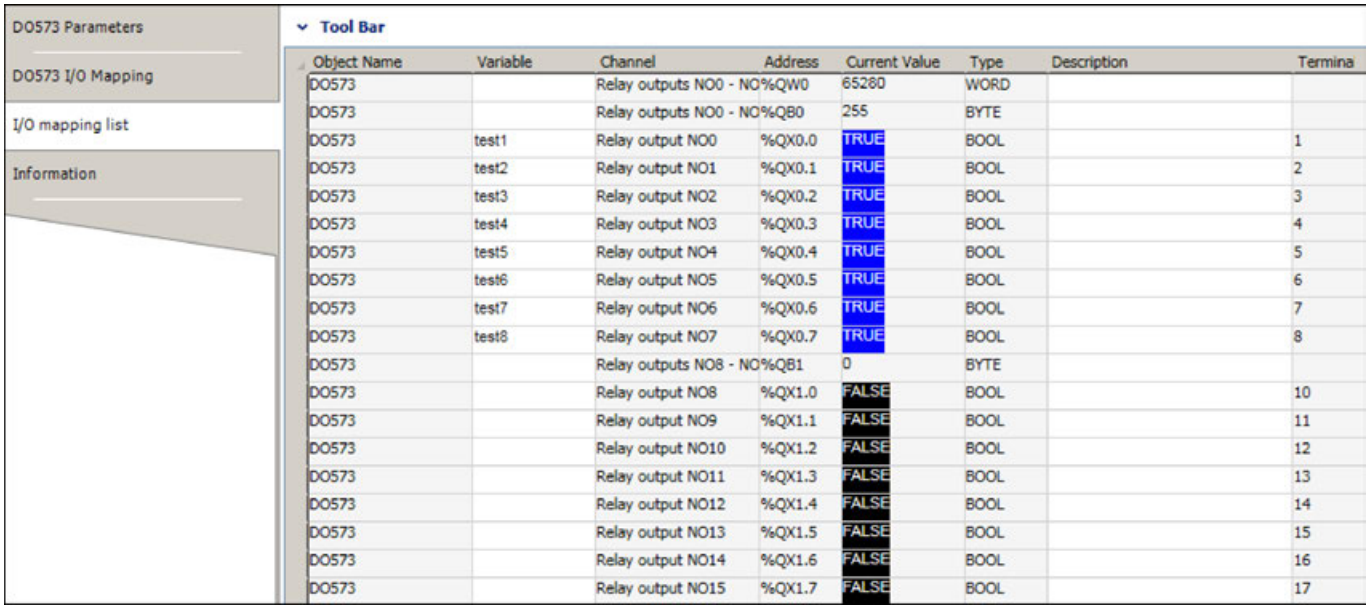
Device diagnosis is disabled by default.

To enable/disable device diagnosis:

1. Double-click on the PLC.
2. Select the tab *“PLC Settings”*.
3. Under *“Additional Settings”* enable/disable *“Diagnosis for devices”*.
   ⇨ When the device diganosis is disabled, this symbol will be displayed in the device tree and no diagnosis messages will be shown.

1.7.2.5 Live values in views with I/O components

"I/O mapping list" tab: In online mode, all Automation Builder views, which contain I/O component mapping tables, show animated live values which are updated every second.



Object Name	Variable	Channel	Address	Current Value	Type	Description	Termina
DO573		Relay outputs NO0 - NO	%QW0	65280	WORD		
DO573		Relay outputs NO0 - NO	%QB0	255	BYTE		
DO573	test1	Relay output NO0	%QX0.0	TRUE	BOOL		1
DO573	test2	Relay output NO1	%QX0.1	TRUE	BOOL		2
DO573	test3	Relay output NO2	%QX0.2	TRUE	BOOL		3
DO573	test4	Relay output NO3	%QX0.3	TRUE	BOOL		4
DO573	test5	Relay output NO4	%QX0.4	TRUE	BOOL		5
DO573	test6	Relay output NO5	%QX0.5	TRUE	BOOL		6
DO573	test7	Relay output NO6	%QX0.6	TRUE	BOOL		7
DO573	test8	Relay output NO7	%QX0.7	TRUE	BOOL		8
DO573		Relay outputs NO8 - NO	%QB1	0	BYTE		
DO573		Relay output NO8	%QX1.0	FALSE	BOOL		10
DO573		Relay output NO9	%QX1.1	FALSE	BOOL		11
DO573		Relay output NO10	%QX1.2	FALSE	BOOL		12
DO573		Relay output NO11	%QX1.3	FALSE	BOOL		13
DO573		Relay output NO12	%QX1.4	FALSE	BOOL		14
DO573		Relay output NO13	%QX1.5	FALSE	BOOL		15
DO573		Relay output NO14	%QX1.6	FALSE	BOOL		16
DO573		Relay output NO15	%QX1.7	FALSE	BOOL		17

1.7.2.6 Communication module and fieldbus diagnosis

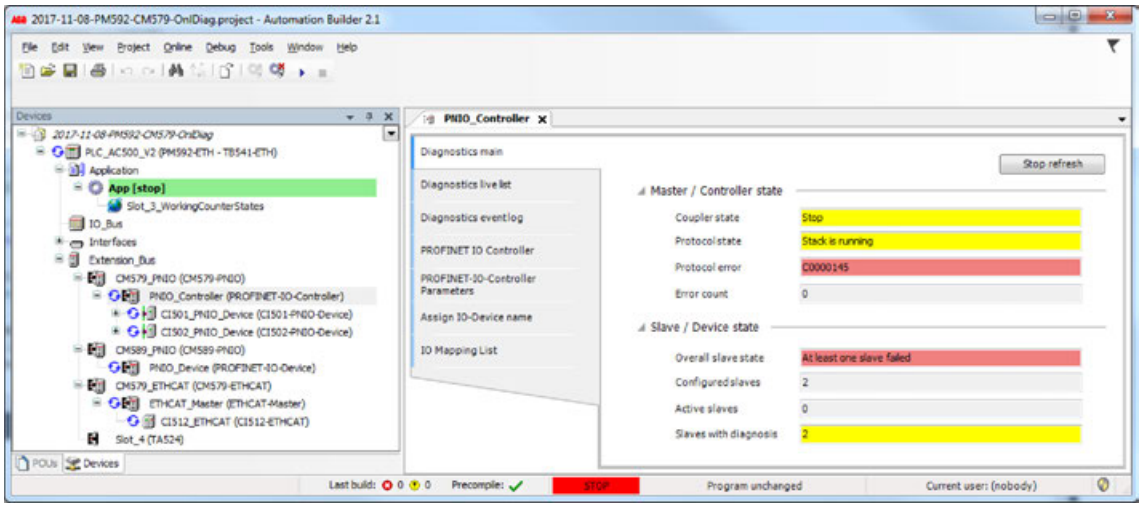
1.7.2.6.1 Fieldbus commissioning

Common online diagnosis views for all netX-based communication modules (e. g. CM579-ETHCAT, CM579-PNIO) can be accessed whenever the related PLC is in online mode
↳ *Chapter 1.7.2.2 "Entering/leaving the online mode" on page 4046*.

Master/controller modules

Master/controller modules like CM579-ETHCAT or CM579-PNIO, provide the following diagnosis views:

- *"Diagnostics main"*: provides diagnosis messages which are common for all protocols (e.g., protocol state and error)
- *"Diagnostics live list"*: provides a list of connected slaves/devices and their state ↳ *Chapter 1.7.2.6.1.1.1 " PROFINET scan and comparison view" on page 4057*
- *"Diagnostics eventlog"*: provides diagnosis messages from the master/controller and its connected slaves/devices



PROFINET scan and comparison view

PNIO_Controller

1. After going online, double-click on *"PNIO_Controller (PROFINET-IO-Controller)"* in the device tree.

   ⇨ The editor *"PNIO_Controller"* is displayed.

2. Select tab *"Diagnostics live list"* and click *[Scan]* to find all hardware devices that exist.

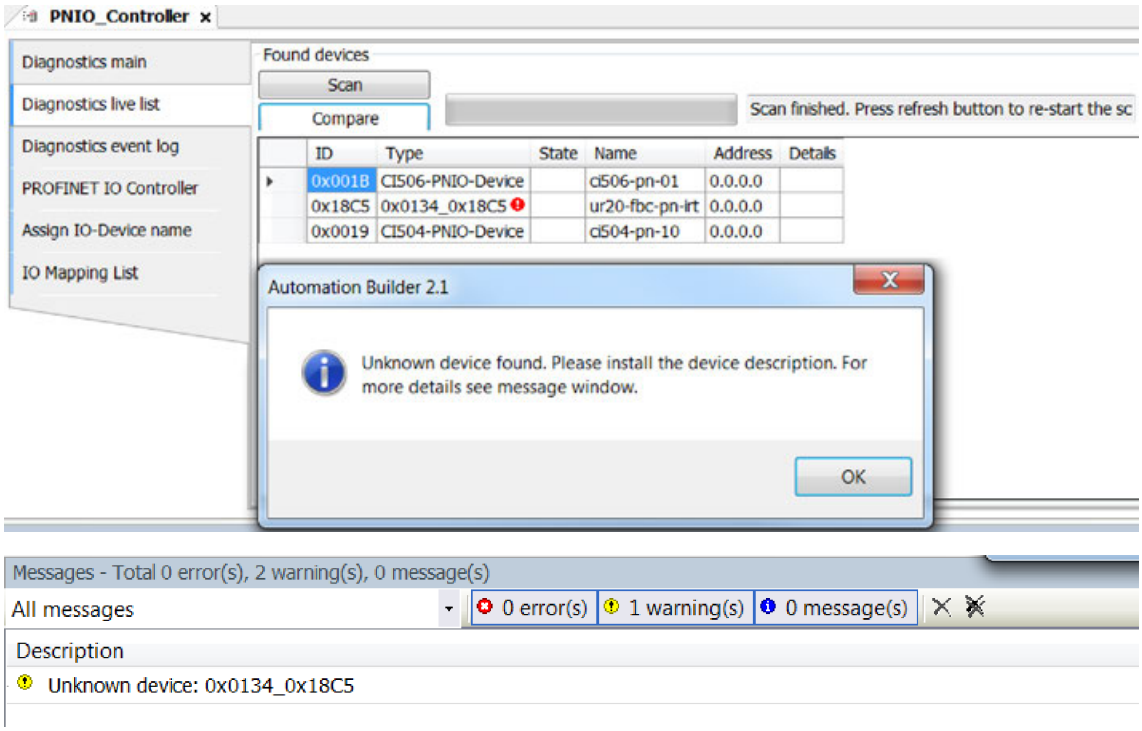
   ⇨ The found devices are listed in a table.

3. Click *[Compare]* to compare the found hardware I/O devices with the current project configuration.

Unknown hardware

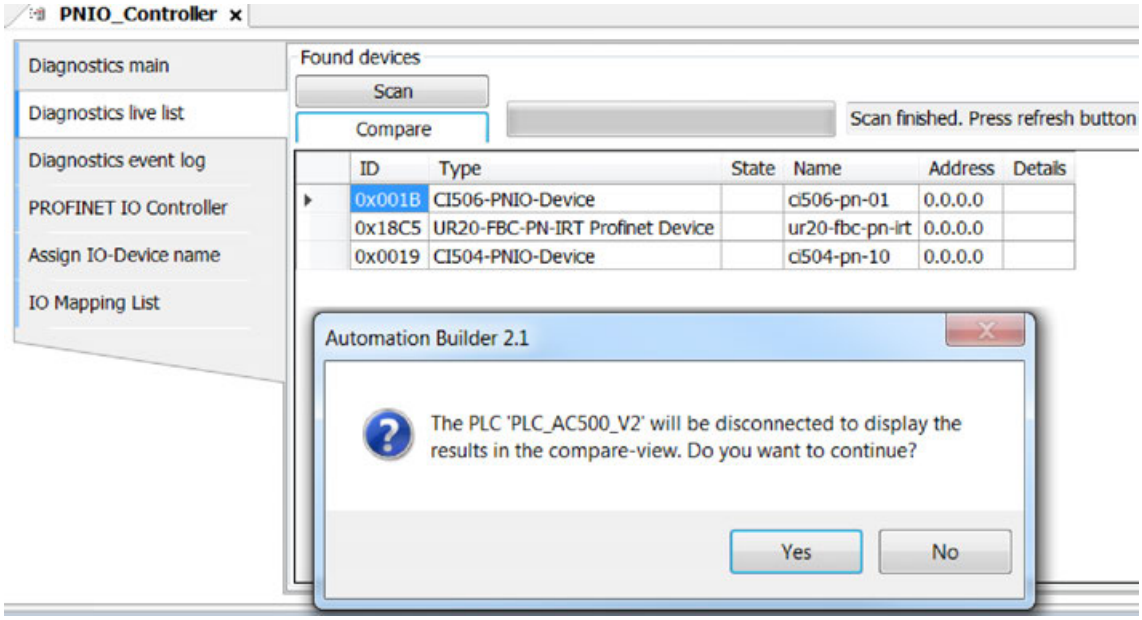
If any I/O hardware device is unknown:

- The devices will be marked with a red exclamation mark.
- A message box will be appear for each unknown device.
- Automation Builder generates a message with information about its vendor ID and device ID.



Comparison view

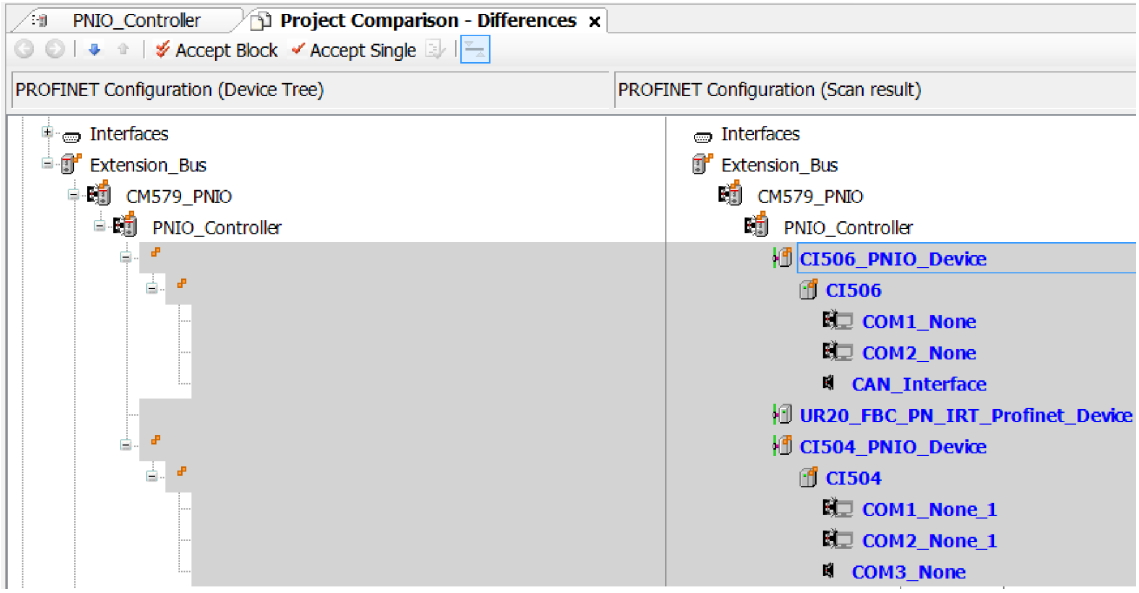
1. To display the comparison view, install the device description for the unknown device.
2. After installing the device description, click *[Scan]* and click *[Compare]*.



⇨ The message box informs you, that the application will go offline to display the comparison view.

3. Click *[Yes]*.

   ⇨ The *"Project Comparison - Differences"* tab displays the difference between the PROFINET configuration in Automation Builder (left side) and the real hardware configuration (right side).

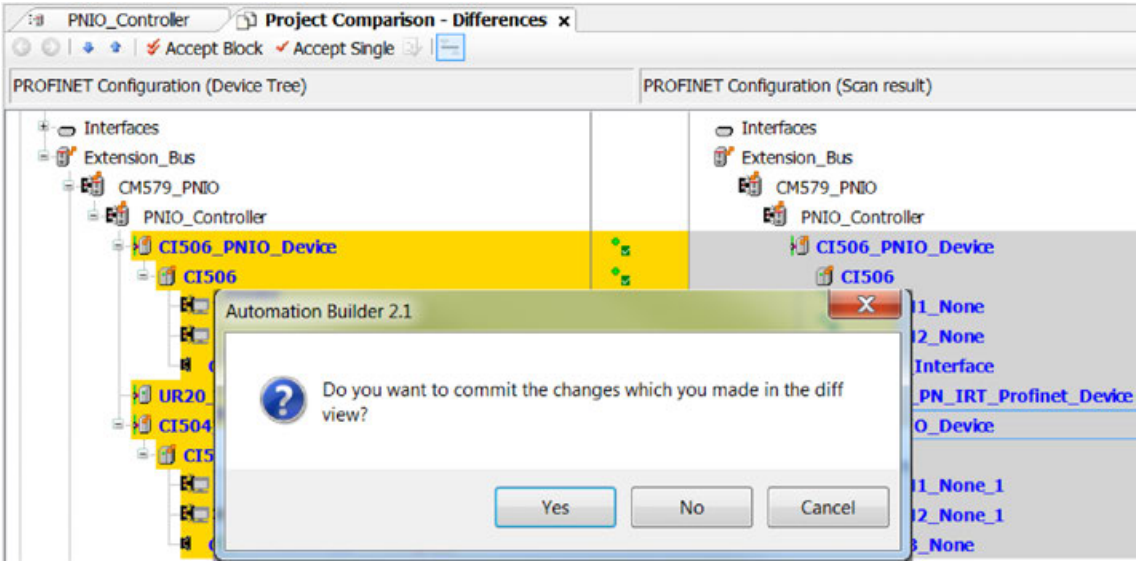


4. Click *[Accept Single]* to accept only a part of the differences or *[Accept Block]* to accept all differences.

   ⇨ After clicking on the Button *[Accept Single]* or *[Accept Block]* the found devices will be moved from the right side to the left side.

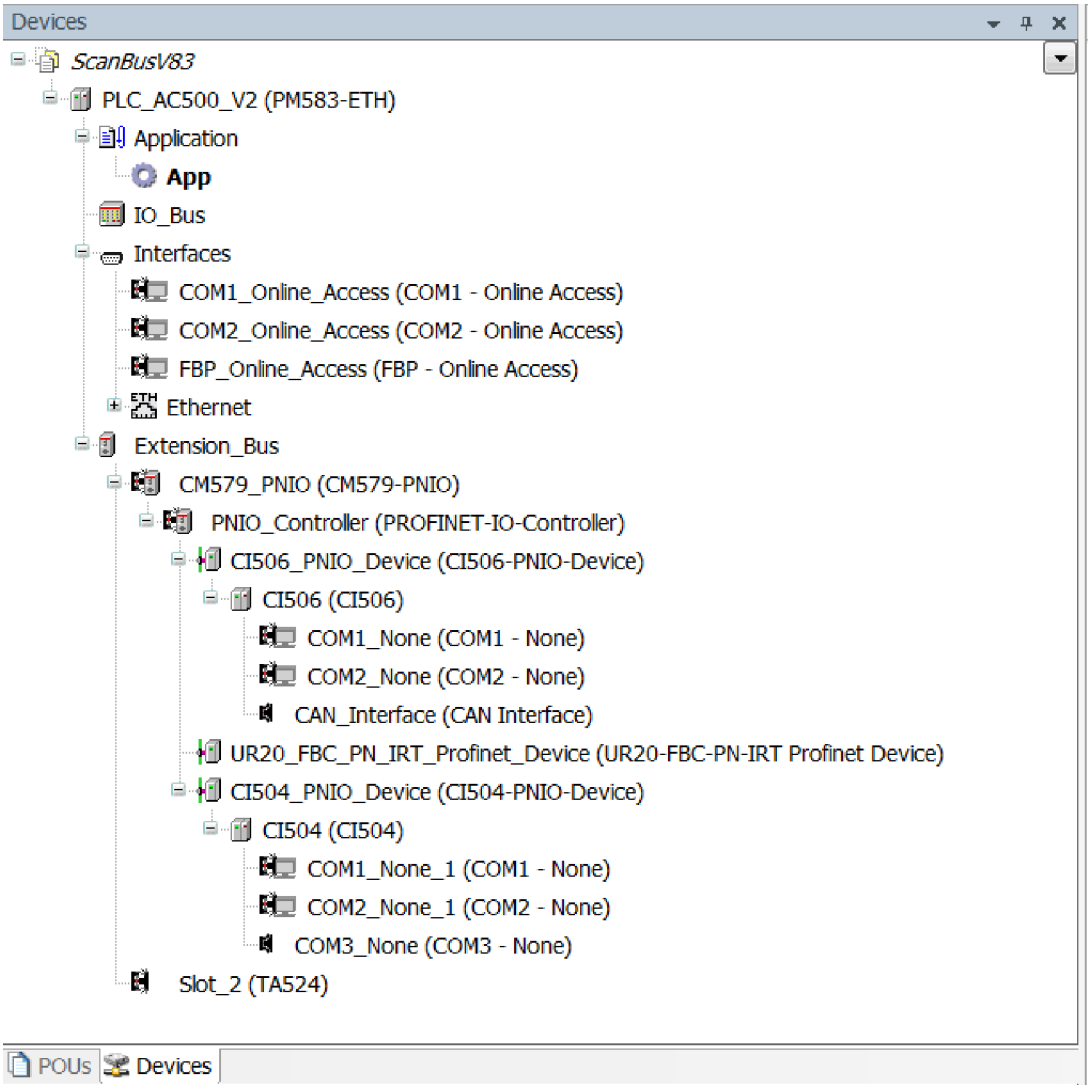
5. Close tab *"Project Comparison - Differences"*.

   ⇨ A message will be displayed to ask if you want to commit the new changes into project.



6. Click *[Yes]*.

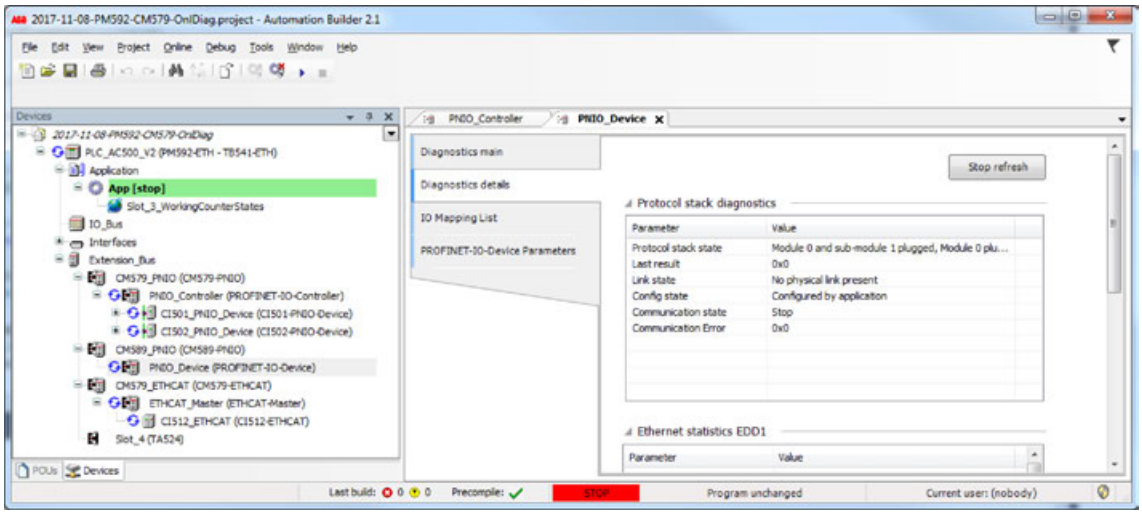
   ⇨ The changes will be saved and the devices will be added to the project.



Slave/device communication modules

Diagnosis views for slave/device communication modules like CM589-PNIO:

- *"Diagnostics main"*: provides diagnosis messages which are common for all protocols
- *"Diagnostics details"*: provides protocol specific diagnosis messages

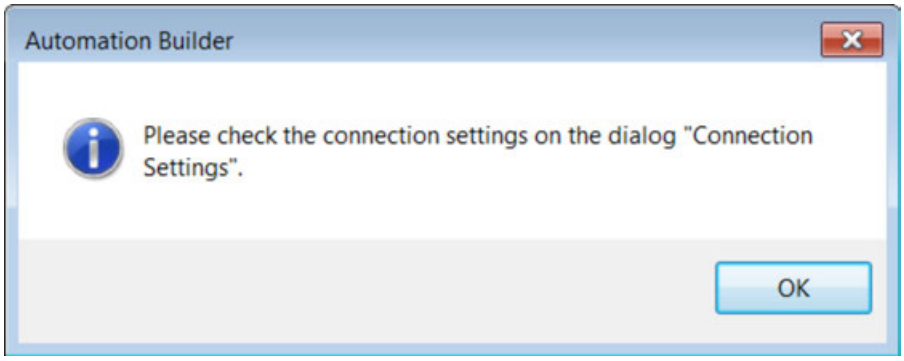


1.7.2.6.2 CI52x Modbus diagnosis

1. Double-click node *"CI52x_MODTCP"* in the device tree.
2. Select *"CI52x Diagnosis"* tab.
   ⇨ The button *[Get Diagnosis]* appears in the tab view.
3. Click on the button *[Get Diagnosis]*.
   ⇨ One of the following use cases will be displayed:
   - Device not connected ↳ *"Device not connected" on page 4061*
   - No Errors on the device ↳ *"No errors on the device" on page 4062*
   - Diagnosis list ↳ *"Diagnosis list" on page 4062*

Device not connected

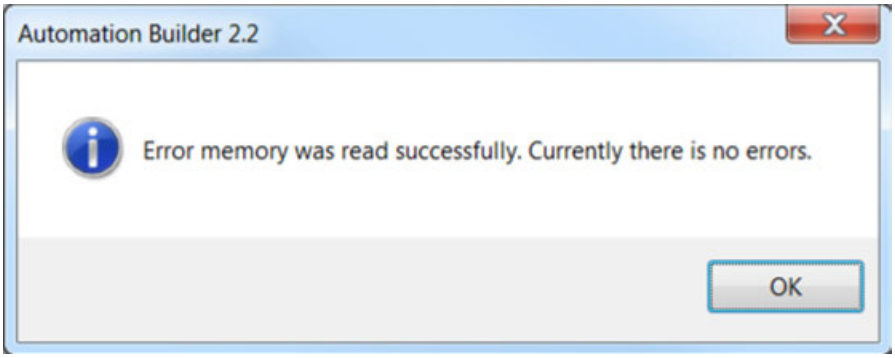
If there is no device connected to the project, the following dialog will be displayed:



1. Select tab *"Connection Settings"* and enter the IP address for the device.
2. Click again button *[Get Diagnosis]*.

No errors on the device

If there are no errors on the device the following dialog will be displayed:



Diagnosis list

If the device is not correctly configured the errors will be displayed with *"Error Code"* and *"Code Description"*.

	Error Code	Code Description
▸	234946507	Process voltage too low
	234946541	Process voltage switched off
	234881072	Module number: 1: Measurement overflow or cut wire at the analog input
*		

1.7.3 Diagnosis messages

1.7.3.1 CPU diagnosis

Diagnosis messages are included in diagnosis text list *"Diag_V3_PLC"*.

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Sub 3_x	Meaning	Sub 4_x	Meaning				
2	0	CPU	2	Parameter	0	-	0	-	0	27	Failed to set parameter "Reboot at powerfail"	
2	14	I/O bus	0	-	0	-	0	-	0	2	Resource failure	
2	14	I/O bus	0	-	0	-	0	-	0	3	Timeout	
2	14	I/O bus	0	-	0	-	0	-	0	17	Error setting I/O bus master parameter	
2	21	Display	0	-	0	-	0	-	0	23	Wrong version of display firmware	
3	21	Display	0	-	0	-	0	-	0	22	Error at initialization of display	
4	22	Battery	0	-	0	-	0	-	0	8	Empty or missing	
4	19	Flash disk	5	Index	0...n	0...n	0	-	0	9	Medium has used 80 % of its spare capacity	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Sub 3_x	Meaning	Sub 4_x	Meaning				
4	19	Flash disk	5	Index	0...n	0...n	0	-	0	10	Medium has almost used its complete spare capacity or is already dead, action required	
11	0	CPU	2	Parameter	0	-	0	-	0	18	At least one parameter not found	
11	0	CPU	2	Parameter	0	-	0	-	0	19	Unable to read at least one parameter value	
11	0	CPU	2	Parameter	0	-	1 or 2	'too big' or 'too small'	0	20	Invalid value of parameter "LED"	
11	0	CPU	2	Parameter	0	-	1 or 2	'too big' or 'too small'	0	21	Invalid value of parameter "Battery"	
11	0	CPU	2	Parameter	0	-	1 or 2	'too big' or 'too small'	0	26	Invalid value of parameter "Reboot at powerfail"	

1.7.3.2 I/O bus diagnosis

Diagnosis messages are included in diagnosis text lists *“Diag_IO_Bus”* and *“Diag_S500_IO_Bus”*.

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
2	0	Master	0	-	0	-	0	-	0	16129	Severe error, see log	
2	0	Master	0	-	0	-	0	-	0	16130	Fatal error, see log	
2	0	Master	0	-	0	-	0	-	0	16194	Fatal error, not running any more	
3	0	Master	0	-	0	-	0	-	0	16128	Failed Max Wait Run	
4	1...20	Module n	0	-	0	-	0	-	0	9480	Module <n>, removed from hot swap terminal unit	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
4	1...20	Module n	0	-	0	-	0	-	0	9526	Module <n>, module on hot swap terminal unit does not support hot swap functionality	
4	1...20	Module n	0	-	0	-	0	-	0	9764	Module <n>, defective hot swap terminal unit	
11	1...20	Module n	0	-	0	-	0	-	0	16133	Module <n>, output data size mismatch	
11	1...20	Module n	0	-	0	-	0	-	0	16134	Module <n>, input data size mismatch	
11	0	Master	0	-	0	-	0	-	0	16145	Error setting I/O bus master parameter	
11	0	Master	0	-	0	-	0	-	0	16146	Failed to start the parameterization of modules	
11	1...20	Module n	0	-	0	-	0	-	0	16147	Module <n>, failed setting parameters	
11	1...20	Module n	0	-	0	-	0	-	0	16149	Module <n>, no module data	
11	1...20	Module n	0	-	0	-	0	-	0	16158	Module <n>, type of present module does not match configuration	
11	0	Master	0	-	0	-	0	-	0	16159	Configured number of modules differs from found ones	
11	0	Master	0	-	0	-	0	-	0	16160	At least one module failed during configuration	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
11	1...20	Module n	0	-	0	-	0	-	0	16248	Module <n>, failed setting expected module	
11	1...20	Module n	0	-	0	-	0	-	0	16254	Module <n>, size of parameters expected by module differs from size provided by configuration	

1.7.3.3 S500 I/O modules diagnosis

Diagnosis messages are included in diagnosis text lists *“Diag_IO_Bus”* and *“Diag_S500_IO_Bus”*.

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
2	1...20	Module n	0	-	-	-	0	-	0	8482	Timeout, while initializing	
2	1...20	Module n	0	-	-	-	0	-	0	8432	Timeout while initializing an I/O module	
2	1...20	Module n	0	-	-	-	0	-	0	9249	Timeout while waiting for Reset	
2	1...20	Module n	0	-	-	-	0	-	0	9258	Breakdown, communication lost	
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	3	Channel <n>, discrepancy time expired	Check discrepancy time value, channel wiring and sensor.

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	12	Channel <n>, test pulse error	Check wiring and sensor
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	13	Channel <n>, test pulse cross-talk error	Check wiring and sensor. If this error persists, replace I/O module. Contact ABB technical support.
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	18	Channel <n>, test error	
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	25	Channel <n>, stuck-at error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	28	Channel <n>, cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	1...20	Module n	1	Channel	0...3	0...3	0	-	0	273	Channel <n>, test error	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
3	1...20	Module n	1	Channel	0...3	0...3	0	-	0	311	Channel <n>, value difference too high	Adjust tolerance window for channels. Check channel wiring and sensor configuration.
3	1...20	Module n	1	Channel	0...7	0...7	0	-	0	524	Channel <n>, stuck-at error	
3	1...20	Module n	1	Channel	0...7	0...7	0	-	0	525	Channel <n>, read-back error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	530	Channel <n>, cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	1...20	Module n	1	Channel	0...15	0...15	0	-	0	540	Channel <n>, test error	
3	1...20	Module n	1	Channel	0...7	0...7	0	-	0	555	Channel <n>, internal error	
3	1...20	Module n	1	Channel	0...7	0...7	0	-	0	1037	Channel <n>, test error	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
3	1...20	Module n	0	-	-	-	0	-	0	8480	Not supported protocol variant	
3	1...20	Module n	0	-	-	-	0	-	0	8707	PROFIsafe watchdog timed out	
3	1...20	Module n	0	-	-	-	0	-	0	8708	Overvoltage	
3	1...20	Module n	0	-	-	-	0	-	0	8711	Undervoltage	
3	1...20	Module n	0	-	-	-	0	-	0	8722	Internal error	
3	1...20	Module n	0	-	-	-	0	-	0	8723	Checksum error has occured in iParameters	
3	1...20	Module n	0	-	-	-	0	-	0	8724	PROFIsafe communication error	
3	1...20	Module n	0	-	-	-	0	-	0	8732	Internal error	
3	1...20	Module n	0	-	-	-	0	-	0	8747	Internal runtime error	
3	1...20	Module n	0	-	-	-	0	-	0	8961	Wrong parameter value, check configuration	
3	1...20	Module n	0	-	-	-	0	-	0	8979	Checksum error has occured in iParameter or F-Parameters	
3	1...20	Module n	0	-	-	-	0	-	0	8986	Invalid configuration	Check modules and parameterization

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
3	1...20	Module n	0	-	-	-	0	-	0	8988	F-Parameter configuration and address switch value do not match.	Check I/O module F-Parameter configuration and module address switch value.
3	1...20	Module n	0	-	-	-	0	-	0	16131	Timeout	Replace I/O module
3	1...20	Module n	0	-	-	-	0	-	0	16137	Overflow diagnosis buffer	Restart
3	1...20	Module n	0	-	-	-	0	-	0	16138	Non-safety I/O: Voltage overflow on outputs (above UP3 level), Safety I/O: Process voltage too high	Check terminals / process voltage
3	1...20	Module n	0	-	-	-	0	-	0	16139	Process voltage UP or UP3 too low	Check process voltage
3	1...20	Module n	0	-	-	-	0	-	0	16146	Plausibility check failed (iParameter)	Check configuration
3	1...20	Module n	0	-	-	-	0	-	0	16147	Checksum error	Non-safety I/O: Replace I/O module Safety I/O: Check safety configuration and CRCs for I- and F-Parameters.

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
3	1...20	Module n	0	-	-	-	0	-	0	16148	PROFIsafe communication error	Restart I/O module. If this error persists, contact ABB technical support.
3	1...20	Module n	0	-	-	-	0	-	0	16153	PROFIsafe watchdog timed out	Restart I/O module. If this error persists, increase PROFIsafe watchdog time.
3	1...20	Module n	0	-	-	-	0	-	0	16154	Parameter error	Check configuration.
3	1...20	Module n	0	-	-	-	0	-	0	16156	F-Parameter configuration and address switch value do not match.	Check I/O module F-Parameter configuration and module address switch value.
3	1...20	Module n	0	-	-	-	0	-	0	16164	Internal data interchange failure	Replace I/O module
3	1...20	Module n	0	-	-	-	0	-	0	16168	Different hard-/firmware versions in the module	Replace I/O module
3	1...20	Module n	0	-	-	-	0	-	0	16171	Internal error	Replace I/O module
3	1...20	Module n	0	-	-	-	0	-	0	16175	Sensor voltage too low	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
4	1...20	Module n	0	-	-	-	0	-	0	8432	Timeout while waiting for ready state	
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	257	Channel <n>, wrong measurement, false temperature at the compensation channel	
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	258	Channel <n>, AI531: wrong measurement, potential difference is too high; CD522: PWM duty cycle out of duty area	
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	260	Channel <n>, measurement overflow	Check channel wiring and sensor power supply.
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	263	Channel <n>, measurement underflow at analog input	Check channel wiring and sensor power supply.
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	266	Channel <n>, short circuit and cut wire or "out of range"	
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	267	Channel <n>, output/ process voltage to small/low.	

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	303	Channel <n>, short circuit at the analog input	Check channel wiring
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	304	Channel <n>, analog value overflow or broken wire at an analog input.	
4	1...20	Module n	1	Channel	0...15	0...15	0	-	0	530	Internal fuse at 0 V is defect. 0 V not connected with GND.	
4	1...20	Module n	1	Channel	0...23	0...23	0	-	0	559	Channel <n>, short circuit at the digital output	Check channel wiring
4	1...20	Module n	1	Channel	0...7	0...7	0	-	0	775	Channel <n>, measurement underflow at analog output	Check channel wiring
4	1...20	Module n	1	Channel	0	0	0	-	0	796	Different configuration	
4	1...20	Module n	0	-	-	-	0	-	0	8482	Timeout while waiting for ready status	
4	1...20	Module n	0	-	-	-	0	-	0	8483	Timeout during parameterization	
4	1...20	Module n	0	-	-	-	0	-	0	9480	I/O module removed from hot swap terminal unit or defective module on hot swap terminal unit.	Plug I/O module, replace I/O module

Error severity	SubSysteminfo								Additional	Error code Err_x	Meaning	Remedy
	Byte0		Byte1		Byte2		Byte3					
	Sub 1_x	Meaning	Sub 2_x	Meaning	Value	Meaning	Value	Meaning				
4	1...20	Module n	0	-	-	-	0	-	0	9500	Wrong I/O module replugged on hot swap terminal unit	Remove wrong I/O module and plug projected I/O module
4	1...20	Module n	0	-	-	-	0	-	0	9514	No communication with I/O module on hot swap terminal unit	Replace I/O module
4	1...20	Module n	0	-	-	-	0	-	0	9526	I/O module does not support hot swap	Power off system and replace I/O module
4	1...20	Module n	0	-	-	-	0	-	0	9736	Hot swap terminal unit required, but not found	
4	1...20	Module n	0	-	-	-	0	-	0	9770	No communication with hot swap terminal unit	Restart, if error persists replace terminal unit
4	1...20	Module n	0	-	-	-	0	-	0	16172	Has not passed factory test	
4	1...20	Module n	0	-	-	-	0	-	0	16173	No process voltage UP or UP3	Check process voltage
11	0	-	0	-	-	-	0	-	0	16159	Configured number of modules differs from found ones	
11	0	-	0	-	-	-	0	-	0	16160	At least one module failed during configuration	

1.7.3.4 Communication modules diagnosis

1.7.3.4.1 CM579-ETHCAT

Status codes

Hexadecimal Value	Definition	Description
0x00000000	TLR_S_OK	Status ok
0xC0650005	TLR_E_ETHERCAT_MASTER_ERROR_BUSSCAN_FAILED	Existing bus does not match configured bus.
0xC0650006	TLR_E_ETHERCAT_MASTER_NOT_ALL_SLAVES_AVAIL	Not all slaves are available.
0xC065000B	TLR_E_ETHERCAT_MASTER_INVALID_BUSCYCLETIME	The requested bus cycle time is invalid.
0xC065000C	TLR_E_ETHERCAT_MASTER_INVALID_BROKEN_SLAVE_BEHAVIOUR_PARA	Invalid parameter for broken slave behavior.
0xC065000F	TLR_E_ETHERCAT_MASTER_COE_INVALID_SLAVEID	Invalid SlaveId was used for CoE.
0xC0650012	TLR_E_ETHERCAT_MASTER_COE_INVALID_INDEX	Invalid Index on slave requested.
0xC0650013	TLR_E_ETHERCAT_MASTER_COE_INVALID_COMMUNICATION_STATE	Invalid bus communication state for CoE-Usage.
0xC0650014	TLR_E_ETHERCAT_MASTER_COE_FRAME_LOST	Frame with CoE data is lost.
0xC0650015	TLR_E_ETHERCAT_MASTER_COE_TIMEOUT	Timeout during CoE service.
0xC0650016	TLR_E_ETHERCAT_MASTER_COE_SLAVE_NOT_ADDRESSABLE	Slave is not addressable (not on bus or power down?).
0xC0650017	TLR_E_ETHERCAT_MASTER_COE_INVALID_LIST_TYPE	Invalid list type requested (during GetOdList).
0xC0650018	TLR_E_ETHERCAT_MASTER_COE_SLAVE_RESPONSE_TOO_BIG	Data in slave response is too big for confirmation packet.
0xC0650019	TLR_E_ETHERCAT_MASTER_COE_INVALID_ACCESSBITMASK	Invalid access mask selected (during GetEntryDesc).
0xC065001A	TLR_E_ETHERCAT_MASTER_COE_WKC_ERROR	Slave Working Counter Error during CoE service.
0xC065001C	TLR_E_ETHERCAT_MASTER_INVALID_COMMUNICATION_STATE	Command is not usable in the communication state.
0xC065001E	TLR_E_ETHERCAT_MASTER_BUS_SCAN_CURRENTLY_RUNNING	The scan is already running. It cannot be started twice at the same time.
0xC065001F	TLR_E_ETHERCAT_MASTER_BUS_SCAN_TIMEOUT	Timeout during bus scan. But at least a link is established.
0xC0650020	TLR_E_ETHERCAT_MASTER_BUS_SCAN_NOT_READY_YET	The bus scan was not started before or is not finish yet.
0xC0650021	TLR_E_ETHERCAT_MASTER_BUS_SCAN_INVALID_SLAVE	The requested slave is invalid.
0xC0650022	TLR_E_ETHERCAT_MASTER_COE_INVALIDACCESS	Slave does not allow reading or writing (CoE-Access).
0xC0650023	TLR_E_ETHERCAT_MASTER_COE_NO_MBX_SUPPORT	Slave does not support a mailbox.

Hexadecimal Value	Definition	Description
0xC0650024	TLR_E_ETHERCAT_MASTER_COE_NO_COE_SUPPORT	Slave does not support CoE.
0xC0650025	TLR_E_ETHERCAT_MASTER_TASK_CREATION_FAILED	Task could not be created during run time.
0xC0650026	TLR_E_ETHERCAT_MASTER_INVALID_SLAVE_SM_CONFIGURATION	The Sync Manager configuration of a slave is invalid.
0xC0650027	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TOGGLE	SDO abort code: Toggle bit not alternated.
0xC0650028	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TIMEOUT	SDO abort code: SDO protocol timed out.
0xC0650029	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_CCS_SCS	SDO abort code: Client/server command specifier not valid or unknown.
0xC065002A	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_BLK_SIZE	SDO abort code: Invalid block size (block mode only).
0xC065002B	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_SEQNO	SDO abort code: Invalid sequence number (block mode only).
0xC065002C	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_CRC	SDO abort code: CRC error (block mode only).
0xC065002D	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_MEMORY	SDO abort code: Out of memory.
0xC065002E	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_ACCESS	SDO abort code: Unsupported access to an object.
0xC065002F	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_WRITEONLY	SDO abort code: Attempt to read a write only object.
0xC0650030	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_READONLY	SDO abort code: Attempt to write a read only object.
0xC0650031	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_INDEX	SDO abort code: Object does not exist in the object dictionary.
0xC0650032	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_PDO_MAP	SDO abort code: Object cannot be mapped to the PDO.
0xC0650033	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_PDO_LEN	SDO abort code: The number and length of the objects to be mapped would exceed PDO length.
0xC0650034	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_P_INCOMP	SDO abort code: General parameter incompatibility reason.
0xC0650035	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_I_INCOMP	SDO abort code: General internal incompatibility in the device.
0xC0650036	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_HARDWARE	SDO abort code: Access failed due to an hardware error.
0xC0650037	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE	SDO abort code: Data type does not match, length of service parameter does not match.
0xC0650038	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE1	SDO abort code: Data type does not match, length of service parameter too high.
0xC0650039	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE2	SDO abort code: Data type does not match, length of service parameter too low.

Hexadecimal Value	Definition	Description
0xC065003A	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_OFFSET	SDO abort code: Sub-index does not exist.
0xC065003B	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE	SDO abort code: Range of values of parameter exceeded (only for write access).
0xC065003C	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE1	SDO abort code: Value of parameter written too high.
0xC065003D	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE2	SDO abort code: Value of parameter written too low.
0xC065003E	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_MINMAX	SDO abort code: Maximum value is less than minimum value.
0xC065003F	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_GENERAL	SDO abort code: general error.
0xC0650040	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER	SDO abort code: Data cannot be transferred or stored to the application.
0xC0650041	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER1	SDO abort code: Data cannot be transferred or stored to the application because of local control.
0xC0650042	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER2	SDO abort code: Data cannot be transferred or stored to the application because of the present device state.
0xC0650043	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DICTIONARY	SDO abort code: Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).
0xC0650044	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_UNKNOWN	SDO abort code: unknown code.
0xC0CC0001	ECM_ERROR_LLD_TIMEOUT	LLD: Timeout
0xC0CC0003	ECM_ERROR_LLD_UNSUPPORTED_COMMAND	LLD: Unsupported command
0xC0CC0004	ECM_ERROR_LLD_DUPLICATE_FIXED_STATION_ADDRESS	LLD: Duplicate fixed station address
0xC0CC0005	ECM_ERROR_LLD_SII_CHECKSUM_ERROR	LLD: SII Checksum Error
0xC0CC0006	ECM_ERROR_LLD_SII_EEPROM_LOADING_ERROR	LLD: SII EEPROM Loading Error
0xC0CC0007	ECM_ERROR_LLD_SII_MISSING_ERROR_ACK	LLD: SII Missing Error Ack
0xC0CC0008	ECM_ERROR_LLD_STATE_CHANGE_FAILED	LLD: State Change Failed
0xC0CC0009	ECM_ERROR_LLD_UNEXPECTED_AL_STATUS	LLD: Unexpected AL Status
0xC0CC000A	ECM_ERROR_LLD_UNEXPECTED_WKC	LLD: Unexpected WKC
0xC0CC000B	ECM_ERROR_LLD_MAILBOX_NOT_AVAILABLE	LLD: Mailbox not available

Hexadecimal Value	Definition	Description
0xC0CC000C	ECM_ERROR_LLD_MAILBOX_MESSAGE_TOO_LARGE	LLD: Mailbox message too large
0xC0CC000D	ECM_ERROR_LLD_CONFIGURATION_IN_PROGRESS	LLD: Configuration in progress
0xC0CC000E	ECM_ERROR_LLD_TOO_MANY_CYCLIC_FRAMES	LLD: Too many cyclic frames
0xC0CC000F	ECM_ERROR_LLD_CYCLIC_FRAME_EXCEEDS_MTU	LLD: Cyclic frame exceeds MTU
0xC0CC0010	ECM_ERROR_LLD_INVALID_CYCLIC_TELEGRAM_CONFIG	LLD: Invalid cyclic telegram config
0xC0CC0011	ECM_ERROR_LLD_BUILDING_COPY_ROUTINES_FAILED	LLD: Building copy routines failed
0xC0CC0012	ECM_ERROR_LLD_UNSUPPORTED_SLAVE_STATION_ADDRESS	LLD: Unsupported slave station address
0xC0CC0013	ECM_ERROR_LLD_STATION_ADDRESS_NOT_ALLOWED	LLD: Station Address not allowed
0xC0CC0014	ECM_ERROR_LLD_INVALID_STD_TX_MBX_PHYS_OFFSET	LLD: Invalid Std TxMbx PhysOffset
0xC0CC0015	ECM_ERROR_LLD_INVALID_STD_RX_MBX_PHYS_OFFSET	LLD: Invalid Std Rx Mbx PhysOffset
0xC0CC0016	ECM_ERROR_LLD_INVALID_BOOT_TX_MBX_PHYS_OFFSET	LLD: Invalid BOOT Rx Mbx PhysOffset
0xC0CC0017	ECM_ERROR_LLD_INVALID_BOOT_RX_MBX_PHYS_OFFSET	LLD: Invalid BOOT Tx Mbx PhysOffset
0xC0CC0018	ECM_ERROR_LLD_INVALID_STD_TX_MBX_SM_NO	LLD: Invalid Std Tx Mbx SmNo
0xC0CC0019	ECM_ERROR_LLD_INVALID_STD_RX_MBX_SM_NO	LLD: Invalid Std Rx Mbx SmNo
0xC0CC001A	ECM_ERROR_LLD_INVALID_BOOT_TX_MBX_SM_NO	LLD: Invalid BOOT Tx Mbx SmNo
0xC0CC001B	ECM_ERROR_LLD_INVALID_BOOT_RX_MBX_SM_NO	LLD: Invalid BOOT Rx Mbx SmNo
0xC0CC001C	ECM_ERROR_LLD_UNCONFIGURED_SLAVE_STATION_ADDRESS	LLD: Unconfigured slave station address
0xC0CC001D	ECM_ERROR_LLD_WRONG_SLAVE_STATE	LLD: Wrong slave state
0xC0CC001E	ECM_ERROR_LLD_CYCLE_TIME_TOO_SMALL	LLD: Cycle time too small
0xC0CC001F	ECM_ERROR_LLD_REPETITION_COUNT_NOT_SUPPORTED	LLD: Repetition count not supported
0xC0CC0020	ECM_ERROR_LLD_INVALID_CALLBACK_TYPE	LLD: Invalid callback type
0xC0CC0021	ECM_ERROR_LLD_INVALID_CYCLE_MULTIPLIER	LLD: Invalid cycle multiplier
0xC0CC0022	ECM_ERROR_LLD_UNKNOWN_ERROR	LLD: Unknown Error
0xC0CC0023	ECM_ERROR_LLD_INVALID_REG_LENGTH	LLD: Invalid reg length

Hexadecimal Value	Definition	Description
0xC0CC0024	ECM_ERROR_LLD_INVALID_PARAMETER	LLD: Invalid parameter
0xC0CC0025	ECM_ERROR_LLD_IRQ_NOT_AVAILABLE	LLD: IRQ not available
0xC0CC0026	ECM_ERROR_LLD_IOMEM_IRQ_NOT_AVAILABLE	LLD: IOMem Irq not available
0xC0CC0027	ECM_ERROR_LLD_HW_INIT_FAILED	LLD: Hardware init failed
0xC0CC0028	ECM_ERROR_LLD_MUTEX_CREATION_FAILED	LLD: Mutex creation failed
0xC0CC0029	ECM_ERROR_LLD_DC_RX_LATCH_COMMAND_REQUIRED_FOR_DC	LLD: DC Rx Latch command is not configured within cyclic frames
0xC0CC002A	ECM_ERROR_LLD_TX_PROCESS_IMAGE_EXCEEDED	LLD: Transmit process image is exceeded
0xC0CC002B	ECM_ERROR_LLD_RX_PROCESS_IMAGE_EXCEEDED	LLD: Receive process image is exceeded
0xC0CC002C	ECM_ERROR_LLD_MBX_STATE_IMAGE_EXCEEDED	LLD: Mailbox State image is exceeded
0xC0CC002D	ECM_ERROR_LLD_RESULT_DUPLICATE_BWR_RX_LATCH_CMD	LLD: Duplicate BWR Rx DC Latch command detected in cyclic frames
0xC0CC002E	ECM_ERROR_LLD_RESULT_DUPLICATE_EXT_SYSTIME_CONTROL_CMD	LLD: Duplicate External Sync SysTime Control command detected in cyclic frames
0xC0CC002F	ECM_ERROR_LLD_CC_PROCESS_IMAGE_EXCEEDED	LLD: Cross Communication Process image exceeded
0x40CD0017	ECM_INFO_EMC_BUS_IS_OFF	Bus is off
0xC0CD0001	ECM_ERROR_EMC_REQUEST_DESTINATION_PROBLEM	Request Destination Problem
0xC0CD0002	ECM_ERROR_EMC_INVALID_SLAVE_STATION_ADDRESS	Invalid slave station address
0xC0CD0003	ECM_ERROR_EMC_CONFIGURATION_BUFFER_IS_OPEN	Configuration buffer is open
0xC0CD0004	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION	Wrong state for reconfiguration
0xC0CD0005	ECM_ERROR_EMC_CONFIGURATION_BUFFER_IS_NOT_OPEN	Configuration buffer is not open
0xC0CD0006	ECM_ERROR_EMC_SLAVE_STATION_ADDRESS_ALREADY_IN_CONFIG	Slave station address already in config
0xC0CD0007	ECM_ERROR_EMC_INVALID_STD_MBX_PARAMETERS	Invalid Std Mbx parameters
0xC0CD0008	ECM_ERROR_EMC_INVALID_BOOT_MBX_PARAMETERS	Invalid BOOT Mbx parameters
0xC0CD0009	ECM_ERROR_EMC_STD_MBX_SM_ARE_OVERLAPPING	Std Mbx SMs are overlapping
0xC0CD000A	ECM_ERROR_EMC_BOOT_MBX_SM_ARE_OVERLAPPING	BOOT Mbx SMs are overlapping
0xC0CD000B	ECM_ERROR_EMC_SM_PARAMS_ALREADY_ADDED	SM Params already added

Hexadecimal Value	Definition	Description
0xC0CD000C	ECM_ERROR_EMC_INVALID_SM_NUMBER	Nvalid SM number
0xC0CD000D	ECM_ERROR_EMC_FMMU_PARAMS_ALREADY_ADDED	FMMU params already added
0xC0CD000E	ECM_ERROR_EMC_INVALID_FMMU_NUMBER	Invalid FMMU number
0xC0CD000F	ECM_ERROR_EMC_INVALID_MIN_STATE	Invalid min state
0xC0CD0010	ECM_ERROR_EMC_CYCLE_FRAME_AMOUNT_EXCEEDED	Cycle frame amount exceeded
0xC0CD0011	ECM_ERROR_EMC_INVALID_CYCLIC_FRAME_IN_CONFIGURATION	Invalid cycle frame in configuration
0xC0CD0012	ECM_ERROR_EMC_CYCLE_FRAME_INDEX_NOT_VALID	Cycle frame index not valid
0xC0CD0013	ECM_ERROR_EMC_INVALID_TELEGRAM_LENGTH	Invalid telegram length
0xC0CD0014	ECM_ERROR_EMC_CYCLE_FRAME_LENGTH_EXCEEDED	Cycle frame length exceeded
0xC0CD0015	ECM_ERROR_EMC_AMOUNT_OF_TELE-GRAMS_IN_CYCLIC_FRAME_EXCEEDED	Amount of telegrams in cyclic frame exceeded
0xC0CD0016	ECM_ERROR_EMC_STATE_CHANGE_IN_PROGRESS	State change in progress
0xC0CD0018	ECM_ERROR_EMC_TOO_MANY_SLAVES_GIVEN	Too many slaves given
0xC0CD0019	ECM_ERROR_EMC_DUPLI-CATE_STA-TION_ADDRESS_IN_LIST	Duplicate station address in list
0xC0CD001A	ECM_ERROR_EMC_COM-MAND_TYPE_NOT_ALLOWED_FOR_SLAVE_FSM	Command type not allowed for slave FSM
0xC0CD001B	ECM_ERROR_EMC_CONFIGURA-TION_DATA_INCORRECT	Configuration data incorrect
0xC0CD001C	ECM_ERROR_EMC_VEN-DORID_MISMATCH	VendorID mismatch
0xC0CD001D	ECM_ERROR_EMC_PRODUCT-CODE_MISMATCH	ProductCode mismatch
0xC0CD001E	ECM_ERROR_EMC_REVI-SIONNO_MISMATCH	Revision number mismatch
0xC0CD001F	ECM_ERROR_EMC_SERI-ALNO_MISMATCH	Serial number mismatch
0xC0CD0020	ECM_ERROR_EMC_LOST_CON-NECTION	Lost connection
0xC0CD0021	ECM_ERROR_EMC_UNKNOWN_STATE_CHANGE_HAPPENED	Unknown state change happened
0xC0CD0022	ECM_ERROR_EMC_UNEX-PECTED_STATE_CHANGE_HAP-PENED	Unexpected state change happened

Hexadecimal Value	Definition	Description
0xC0CD0023	ECM_ERROR_EMC_SLAVE_CHANGED_STATE	Slave changed state
0xC0CD0026	ECM_ERROR_EMC_DC_RX_TIMESTAMP_ERROR	DC Rx Timestamp error
0xC0CD0027	ECM_ERROR_EMC_DC_MASTER_PORT_TIMESTAMP_ERROR	DC master port timestamp error
0xC0CD0028	ECM_ERROR_EMC_INVALID_SLAVE_INDEX	Invalid slave index
0xC0CD0029	ECM_ERROR_EMC_WRONG_MASTER_STATE	
0xC0CD002A	ECM_ERROR_EMC_INVALID_TRANSFER_ID	Invalid Transfer Id
0xC0CD002B	ECM_ERROR_EMC_INVALID_SEGMENTATION	Invalid Segmentation
0xC0CD002C	ECM_ERROR_EMC_IP_PARAMS_ALREADY_ADDED	EoE IP Params already added
0xC0CD002D	ECM_ERROR_EMC_EOE_SUPPORT_NOT_AVAILABLE	EoE support not available
0xC0CD002E	ECM_ERROR_EMC_END_CONFIGURATION_IN_PROGRESS	End configuration in progress
0xC0CD002F	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION_BUS_IS_ON	Wrong state for reconfiguration (Bus Is On)
0xC0CD0030	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION_BUS_SCAN_ACTIVE	Wrong state for reconfiguration (Bus Scan Active)
0xC0CD0031	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION_IN_PROGRESS_TO_BUSOFF	Wrong state for reconfiguration (In Progress to Bus off)
0xC0CD0032	ECM_EROR_EMC_NO_DIAG_ENTRY_AVAILABLE	No Diag Entry available
0xC0CD0033	ECM_ERROR_EMC_SLAVE_SYNC_PARAMS_NOT_POSSIBLE_WITHOUT_WORKING_DC	A slave has been configured to have SYNC0 and/or SYNC1 but does not support DC at all.
0xC0CD0034	ECM_ERROR_EMC_MANDATORY_SLAVE_MISSING	At least one required slave for boot up is missing.
0xC0CD0035	ECM_ERROR_EMC_WRONG_SLAVE_AT_POSITION	A wrong slave at a specific position has been detected.
0xC0CD0036	ECM_ERROR_EMC_NO_DC_REF_CLOCK	No DC reference clock
0xC0CD0037	ECM_ERROR_EMC_DC_REF_CLOCK_DOES_NOT_PROVIDE_64BIT	DC Reference clock does not provide 64 Bit
0xC0CD0038	ECM_ERROR_EMC_INVALID_DC_REF_CLOCK	Invalid DC Reference clock
0xC0CD0039	ECM_ERROR_EMC_COE_SUPPORT_NOT_AVAILABLE	CoE support not available
0xC0CD003A	ECM_ERROR_EMC_SOE_SUPPORT_NOT_AVAILABLE	SoE support not available

Hexadecimal Value	Definition	Description
0xC0CD003B	ECM_ERROR_EMC_FOE_SUPPORT_NOT_AVAILABLE	FoE support not available
0xC0CD003C	ECM_ERROR_EMC_AOE_SUPPORT_NOT_AVAILABLE	AoE support not available
0x40CD003E	ECM_INFO_EMC_RECONNECTED	Reconnected
0x80CD003F	ECM_WARN_EMC_DC_STOPPED	DC stopped
0xC0CD0040	ECM_ERROR_EMC_STOPPED_DUE_SYNC_ERROR	Stopped due Sync Error
0xC0CD0041	ECM_ERROR_EMC_MANDATORY_SLAVE_NOT_IN_OP	At least one mandatory slave is not in OP
0xC0CD0042	ECM_ERROR_EMC_BUS_CYCLE_TIME_NOT_POSSIBLE	Bus Cycle Time not possible
0xC0CD0043	ECM_ERROR_EMC_TOPOLOGY_ERROR_DETECTED	Topology error detected
0xC0CD0044	ECM_ERROR_EMC_TOPOLOGY_MISMATCH_DETECTED	Topology mismatch detected
0xC0CD0045	ECM_ERROR_EMC_NO_VALID_TOPOLOGY_CONFIGURATION_DATA	No valid topology configuration data
0xC0CD0046	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0	Unexpected slave at port 0 of slave.
0xC0CD0047	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT1	Unexpected slave at port 1 of slave.
0xC0CD0048	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT2	Unexpected slave at port 2 of slave.
0xC0CD0049	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT3	Unexpected slave at port 3 of slave.
0xC0CD004A	ECM_ERROR_EMC_UNEXPECTED_SLAVE_RECONNECTED	-
0xC0CD004B	ECM_ERROR_EMC_UNEXPECTED_MISSING_SLAVE_AT_PORT0	Missing slave at port 0 of slave.
0xC0CD004C	ECM_ERROR_EMC_UNEXPECTED_MISSING_SLAVE_AT_PORT1	Missing slave at port 1 of slave.
0xC0CD004D	ECM_ERROR_EMC_UNEXPECTED_MISSING_SLAVE_AT_PORT2	Missing slave at port 2 of slave.
0xC0CD004E	ECM_ERROR_EMC_UNEXPECTED_MISSING_SLAVE_AT_PORT3	Missing slave at port 3 of slave.
0xC0CD004F	ECM_ERROR_EMC_SLAVE_NOT_CHECKED	Slave is not checked.
0xC0CD0050	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0_1	Unexpected slave at port 0 and 1 of slave.
0xC0CD0051	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0_2	Unexpected slave at port 0 and 2 of slave.
0xC0CD0052	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0_3	Unexpected slave at port 0 and 3 of slave.

Hexadecimal Value	Definition	Description
0xC0CD0053	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT1_2	Unexpected slave at port 1 and 2 of slave.
0xC0CD0054	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT1_3	Unexpected slave at port 1 and 3 of slave.
0xC0CD0055	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT2_3	Unexpected slave at port 2 and 3 of slave.
0xC0CD0056	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0_1_2	Unexpected slave at port 0, 1 and 2 of slave.
0xC0CD0057	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0_1_3	Unexpected slave at port 0, 1 and 3 of slave.
0xC0CD0058	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT0_2_3	Unexpected slave at port 0, 2 and 3 of slave.
0xC0CD0059	ECM_ERROR_EMC_UNEXPECTED_SLAVE_AT_PORT1_2_3	Unexpected slave at port 1, 2 and 3 of slave.
0xC0CD005A	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_1	Missing slave at port 0 and 1 of slave.
0xC0CD005B	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_2	Missing slave at port 0 and 2 of slave.
0xC0CD005C	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_3	Missing slave at port 0 and 3 of slave.
0xC0CD005D	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT1_2	Missing slave at port 1 and 2 of slave.
0xC0CD005E	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT1_3	Missing slave at port 1 and 3 of slave.
0xC0CD005F	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT2_3	Missing slave at port 2 and 3 of slave.
0xC0CD0060	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_1_2	Missing slave at port 0, 1 and 2 of slave.
0xC0CD0061	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_1_3	Missing slave at port 0, 1 and 3 of slave.
0xC0CD0062	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_2_3	Missing slave at port 0, 2 and 3 of slave.
0xC0CD0063	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT1_2_3	Missing slave at port 1, 2 and 3 of slave.
0xC0CD0065	ECM_ERROR_EMC_HC_PARTICIPANT_NOT_ALLOWED_IN_MANDATORY_SLAVE_LIST	A Hot Connect group participant is not allowed to be configured a mandatory slave
0xC0CD0066	ECM_ERROR_EMC_HC_PARTICIPANT_NOT_ALLOWED_IN_MULTIPLE_HC_GROUPS	A Hot Connect group participant is not allowed to be configured in multiple Hot Connect groups
0xC0CD0067	ECM_ERROR_EMC_GC_GROUP_HEAD_IS_NOT_LISTED_FOR_HC_DETECTION	Hot Connect group head is not listed for Hot Connect detection
0xC0CD0068	ECM_ERROR_EMC_DC_SETUP_CALCULATION_ERROR	DC Setup calculation has encountered an error
0xC0CD0069	ECM_ERROR_EMC_NON_DC_SLAVE_MORE_THAN_2_PORTS_IN_DC_SETUP	A slave, which does not support DC, has more than 2 ports in a DC setup

Hexadecimal Value	Definition	Description
0xC0CD006A	ECM_ERROR_EMC_HC_GROUP_CONTAINS_NOT_CONFIG-URED_SLAVE	A Hot Connect group has been defined to include a slave address that has no configuration
0xC0CD006B	ECM_ERROR_EMC_ALCON-TROL_TIMEOUT	AL Control Timeout happened i.e. a slave ESM state change was not completed in time
0xC0CD006C	ECM_ERROR_EMC_DC_MEAS-UREMENT_ERROR	DC measurement encountered an error
0xC0CD006D	ECM_ERROR_EMC_RX_DESTINA-TION_EXCEEDS_RX_IMAGE_SIZE	Receive destination exceeds receive image size
0xC0CD006E	ECM_ERROR_EMC_TX_SOURCE_EXCEEDS_TX_IMAGE_SIZE	Transmit source exceeds transmit image size
0xC0CD006F	ECM_ERROR_EMC_WCSTA-TEBIT_EXCEEDS_RX_IMAGE_SIZE	WcState bit placement exceeds receive image size
0xC0CD0070	ECM_ERROR_EMC_WKC_MAP-PING_EXCEEDS_RX_IMAGE_SIZE	Wkc value placement exceeds receive image size
0xC0CD0071	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0	DC Latch Error detected at port 0 of slave
0xC0CD0072	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT1	DC Latch Error detected at port 1 of slave
0xC0CD0073	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT2	DC Latch Error detected at port 2 of slave
0xC0CD0074	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT3	DC Latch Error detected at port 3 of slave
0xC0CD0075	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1	DC Latch Error detected at ports 0 and 1 of slave
0xC0CD0076	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_2	DC Latch Error detected at ports 0 and 2 of slave
0xC0CD0077	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_3	DC Latch Error detected at ports 0 and 3 of slave
0xC0CD0078	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT1_2	DC Latch Error detected at ports 1 and 2 of slave
0xC0CD0079	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT1_3	DC Latch Error detected at ports 1 and 3 of slave
0xC0CD007A	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT2_3	DC Latch Error detected at ports 2 and 3 of slave
0xC0CD007B	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_2	DC Latch Error detected at ports 0, 1 and 2 of slave
0xC0CD007C	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_3	DC Latch Error detected at ports 0, 1 and 3 of slave
0xC0CD007D	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORTS0_2_3	DC Latch Error detected at ports 0, 2 and 3 of slave
0xC0CD007E	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORTS1_2_3	DC Latch Error detected at ports 1, 2 and 3 of slave
0xC0CD007F	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORTS0_1_2_3	DC Latch Error detected at ports 0, 1, 2 and 3 of slave
0xC0CD0080	ECM_ERROR_EMC_ASSIGN_PDO_IS_MISSING_PDO_MAPPING	AssignPDO data is missing related PDO mapping data

Hexadecimal Value	Definition	Description
0xC0CD0081	ECM_ERROR_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_TO_SAME_SM	Parts of Ext Sync object are not mapped to the same SyncManager
0xC0CD0082	ECM_ERROR_EMC_DUPLICATE_EXT_SYNC_OBJ	Duplicate Ext Sync object mapping
0xC0CD0083	ECM_ERROR_EMC_UNSUPPORTED_EXT_SYNC_OBJ_RECORD	Unsupported Ext Sync object record detected
0xC0CD0084	ECM_ERROR_EMC_UNSUPPORTED_MAPPING_OF_EXT_SYNC_OBJ_RECORD	Unsupported mapping of Ext Sync object record detected
0xC0CD0085	ECM_ERROR_EMC_MISSING_MAPPING_OF_EXT_SYNC_OBJ_RECORD	Missing mapping of Ext Sync object record detected
0xC0CD0086	ECM_ERROR_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_TO_SAME_FMMU	Parts of Ext Sync object are not mapped to the same FMMU
0xC0CD0087	ECM_ERROR_EMC_EXT_SYNC_OBJ_INTERNAL_ERROR	Internal error detected regarding Ext Sync object
0xC0CD0088	ECM_ERROR_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_IN_ONE_CYCLIC_CMD	Parts of Ext Sync object are not mapped within the same cyclic command
0xC0CD0089	ECM_ERROR_EMC_UNSUPPORTED_FMMU_MAPPING_OF_EXT_SYNC_OBJ_RECORD	Unsupported FMMU mapping of Ext Sync object detected
0xC0CD008A	ECM_ERROR_EMC_EXT_SYNC_REQUIRES_ADJUST_EXT_SYNC_CMD	Unicast Ext Sync control (APWR/FPWR 0x910) is required
0xC0CD008B	ECM_ERROR_EMC_EXT_SYNC_CMD_DOES_NOT_MATCH_XRMW_CMD	Unicast Ext Sync control does not match xRMW command
0xC0CD008C	ECM_ERROR_EMC_EXT_SYNC_REQUIRES_XRMW_CMD	Ext Sync requires DC configuration (xRMW command to 0x910)
0xC0CD008D	ECM_ERROR_EMC_EXPLICIT_DEV_IDENT_FAILED_ALSTATUS	Explicit Device identification via ALSTATUS failed
0xC0CD008E	ECM_ERROR_EMC_EXPLICIT_DEV_IDENT_FAILED_REG	Explicit Device identification via register failed
0xC0CD008F	ECM_ERROR_EMC_COPY_INFOS_FOUND_AT_UNMAPPED_RECEIVE_DATA	CopyInfos found at unmapped receive data
0xC0CD0090	ECM_ERROR_EMC_COPY_INFO_RECEIVE_DATA_AREA_NOT_MATCHING	CopyInfo receive data area is not matching
0xC0CD0091	ECM_ERROR_EMC_SDO_UPLOAD_TOO_LONG	SDO Upload data too long
0xC0CD0092	ECM_ERROR_EMC_SDO_UPLOAD_TOO_SHORT	SDO Upload data too short

Hexadecimal Value	Definition	Description
0xC0CD0093	ECM_ERROR_EMC_SDO_UPLOAD_COMPARE_DOES_NOT_MATCH_EXPECTATION	SDO Upload compare does not match expectation
0xC0CD0094	ECM_ERROR_EMC_SOE_READ_TOO_LONG	SoE Read IDN data too long
0xC0CD0095	ECM_ERROR_EMC_SOE_READ_TOO_SHORT	SoE Read IDN data too short
0xC0CD0096	ECM_ERROR_EMC_SOE_READ_COMPARE_DOES_NOT_MATCH_EXPECTATION	SoE Read compare does not match expectation
0xC0CD0097	ECM_ERROR_EMC_REG_INITCMD_COMPARE_DOES_NOT_MATCH_EXPECTATION	Register read compare does not match expectation
0xC0CD0098	ECM_ERROR_EMC_REDUNDANCY_PORT_ONLY_POSSIBLE_ONCE	Redundancy port can only be placed once into configuration
0xC0CD0099	ECM_ERROR_EMC_STARTUP_SCAN_SII_FAILED	Startup scan of SII failed
0xC0CD009A	ECM_ERROR_EMC_STARTUP_VERIFY_SII_FAILED	Startup verification of SII failed
0xC0CD009B	ECM_ERROR_EMC_MAIN_PORT_NOT_CONNECTED	Main port not connected during topology scan
0xC0CD009C	ECM_ERROR_EMC_BUS_SCAN_TOO_MANY_SLAVES	Bus scan detects too many slaves
0xC0CD009D	ECM_ERROR_EMC_BUS_SCAN_SPLIT_RING_NOT_SUPPORTED	Bus Scan detects unsupported split ring topology
0xC0CD009E	ECM_ERROR_EMC_BUS_SHUTDOWN	Bus is shutting down
0xC0CD009F	ECM_ERROR_EMC_MASTER_ADDRESS_NOT_ALLOWED_AS_STATION_ADDRESS	Master address (0) is not allowed as station address
0xC0CD00A0	ECM_ERROR_EMC_FIRST_STATION_HAS_INVALID_PORT_0	First station has invalid port 0
0xC0CD00A1	ECM_ERROR_EMC_STATION_HAS_INVALID_PORT	Station has invalid port
0xC0CD00A2	ECM_ERROR_EMC_STATION_HAS_NOT_LISTED_STATION_ADDRESS_IN_PORT	Station has not listed station address in port
0xC0CD00A3	ECM_ERROR_EMC_PORT_CONNECTION_BETWEEN_STATIONS_DOES_NOT_MATCH	Port connection between stations does not match
0xC0CD00A4	ECM_ERROR_EMC_STATION_HAS_ALREADY_USED_STATION_ADDRESS_IN_PORT	Station has already used station address in port
0xC0CD00A5	ECM_ERROR_EMC_INVALID_SM_PHYS_START_ADDRESS	Invalid Sm physical start address

Hexadecimal Value	Definition	Description
0xC0CD00A6	ECM_ERROR_EMC_DC_TOPOLOGY_ON_REDUNDANCY_PORT_NOT_SUPPORTED	DC topology on redundancy port connection not supported. DC slaves having AutoIncrement positions behind redundancy port
0xC0CD00A7	ECM_ERROR_EMC_SM_ASSIGN_PDO_ALREADY_ADDED	Sm AssignPdo already added
0xC0CD00A8	ECM_ERROR_EMC_BASE_SYNC_OFFSET_PERCENTAGE_OUT_OF_RANGE	Base Sync Offset percentage out of range
0xC0CF0001	ECM_ERROR_COE_INITIALIZATION_ERROR	CoE: Initialization Error
0xC0CF0002	ECM_ERROR_COE_INVALID_TRANSFER_HANDLE	CoE: Invalid transfer handle used
0xC0CF0003	ECM_ERROR_COE_NO_MAILBOX_AVAILABLE	CoE. No mailbox available
0xC0CF0004	ECM_ERROR_COE_INVALID_TRANSFER_STATE	CoE: Invalid transfer state
0xC0CF0005	ECM_ERROR_COE_TRANSFER_SEGMENT_TOO_LONG	CoE: Transfer segment is too long
0xC0CF0006	ECM_ERROR_COE_SHUTTING_DOWN	CoE is shutting down.
0xC0CF0007	ECM_ERROR_COE_MAX_TOTAL_BYTES_SMALLER_THAN_ACTUAL_TOTAL_BYTES	CoE: Maximum total bytes is smaller than actual total bytes.
0xC0CF0008	ECM_ERROR_COE_MAILBOX_TRANSMIT_FAILED	CoE: Mailbox transmit failed
0xC0CF0009	ECM_ERROR_COE_TRANSFER_ABORTED	CoE: Transfer has been aborted.
0xC0CF000A	ECM_ERROR_COE_SDOINFO_INITIALIZATION_ERROR	0xC0CF000B
0xC0CF000C	ECM_ERROR_COE_PROTOCOL_ERROR	CoE Protocol Error
0xC0CF000D	ECM_ERROR_COE_NO_AOE_AVAILABLE	CoE: No AoE available
0xC0CF000F	ECM_ERROR_COE_INVALID_SLAVE_STATION_ADDRESS	CoE: Invalid slave station address
0xC0CF8000	ECM_ERROR_COE_ABORTCODE_TOGGLE_BIT_NOT_ALTERNATED	SDO Abort Code: Toggle Bit not alternated
0xC0CF8001	ECM_ERROR_COE_ABORTCODE_COMMAND_SPECIFIER_NOT_VALID	SDO Abort Code: Command specifier not valid
0xC0CF8002	ECM_ERROR_COE_ABORTCODE_PROTOCOL_TIMEOUT	SDO Abort Code: Protocol Timeout
0xC0CF8003	ECM_ERROR_COE_ABORTCODE_OUT_OF_MEMORY	SDO Abort Code: Out Of Memory
0xC0CF8004	ECM_ERROR_COE_ABORTCODE_UNSUPPORTED_ACCESS	SDO Abort Code: Unsupported access
0xC0CF8005	ECM_ERROR_COE_ABORTCODE_OBJECT_IS_WRITE_ONLY	SDO Abort Code: Object is write only

Hexadecimal Value	Definition	Description
0xC0CF8006	ECM_ERROR_COE_ABORT-CODE_OBJECT_IS_READ_ONLY	SDO Abort Code: Object is read only
0xC0CF8007	ECM_ERROR_COE_ABORT-CODE_SUB-INDEX_CANNOT_BE_WRITTEN_SI 0_NZ	SDO Abort Code: Subindex cannot be written if subindex 0 is not zero
0xC0CF8008	ECM_ERROR_COE_ABORT-CODE_COM-PLETE_ACCESS_NOT_SUP-PORTED	SDO Abort Code: Complete access not supported
0xC0CF8009	ECM_ERROR_COE_ABORT-CODE_OBJECT_LENGTH_EXCEE DS_MAILBOX_SIZE	SDO Abort Code: Object length exceeds mailbox size
0xC0CF800A	ECM_ERROR_COE_ABORT-CODE_OBJECT_MAPPED_TO_RX PDO_NO_WRITE	SDO Abort Code: Object mapped to RxPDO, SDO Download blocked
0xC0CF800B	ECM_ERROR_COE_ABORT-CODE_OBJECT_DOES_NOT_EXIS T	SDO Abort Code: Object does not exist
0xC0CF800C	ECM_ERROR_COE_ABORT-CODE_OBJECT_CANNOT_BE_PD O_MAPPED	SDO Abort Code: Object cannot be mapped to PDO
0xC0CF800D	ECM_ERROR_COE_ABORT-CODE_PDO_LENGTH_WOULD_E XCEED	SDO Abort Code: PDO Length would exceed maximum size
0xC0CF800E	ECM_ERROR_COE_ABORT-CODE_GEN_PARAM_INCOMPATI-BILITY	SDO Abort Code: General param-eter incompatibility
0xC0CF800F	ECM_ERROR_COE_ABORT-CODE_ACCESS_FAILED_DUE_TO _HW_ERROR	SDO Abort Code: Access failed due to hardware error
0xC0CF8010	ECM_ERROR_COE_ABORT-CODE_DATA-TYPE_DOES_NOT_MATCH	SDO Abort Code: Data type does not match
0xC0CF8011	ECM_ERROR_COE_ABORT-CODE_DATA-TYPE_LENGTH_TOO_LONG	SDO Abort Code: Data type length too long
0xC0CF8012	ECM_ERROR_COE_ABORT-CODE_DATA-TYPE_LENGTH_TOO_SHORT	SDO Abort Code: Data type length too short
0xC0CF8013	ECM_ERROR_COE_ABORT-CODE_SUB-INDEX_DOES_NOT_EXIST	SDO Abort Code: Subindex does not exist
0xC0CF8014	ECM_ERROR_COE_ABORT-CODE_RANGE_OF_PARAM-ETER_EXCEEDED	SDO Abort Code: Range of param-eter exceeded
0xC0CF8015	ECM_ERROR_COE_ABORT-CODE_VALUE_OF_PARAM_WRITT EN_TOO_HIGH	SDO Abort Code: Value of param-eter written too high
0xC0CF8016	ECM_ERROR_COE_ABORT-CODE_VALUE_OF_PARAM_WRITT EN_TOO_LOW	SDO Abort Code: Value of param-eter written too low

Hexadecimal Value	Definition	Description
0xC0CF8017	ECM_ERROR_COE_ABORT-CODE_MIN_VALUE_IS_LESS_THAN_MAX_VALUE	SDO Abort Code: Minimum value is less than maximum value
0xC0CF8018	ECM_ERROR_COE_ABORT-CODE_GENERAL_ERROR	SDO Abort Code: General Error
0xC0CF8019	ECM_ERROR_COE_ABORT-CODE_NO_TRANSFER_TO_APP	SDO Abort Code: Data cannot be transferred or stored to the application
0xC0CF801A	ECM_ERROR_COE_ABORT-CODE_LOCAL_CONTROL	SDO Abort Code: Data cannot be transferred or stored to the application because of local control
0xC0CF801B	ECM_ERROR_COE_ABORT-CODE_NO_TRANSFER_DUE_TO_CURRENT_STATE	SDO Abort Code: Data cannot be transferred or stored to the applcation because of the present device state
0xC0CF801C	ECM_ERROR_COE_ABORT-CODE_NO_OBJECT_DICTIONARY_PRESENT	SDO Abort Code: Object dictionary dynamic generation fails or no object dictionary is present
0xC0CF801D	ECM_ERROR_COE_ABORT-CODE_UNKNOWN_ABORT_CODE	SDO Abort Code: Unknown abort code
0xC0CF801E	ECM_ERROR_COE_ABORT-CODE_GEN_INTERNAL_COMPAT	SDO Abort Code: General internal incompatibility in the device
0xC0D00001	ECM_ERROR_EOE_INVALID_MAC_ADDRESS	Invalid MAC address
0xC0D00002	ECM_ERROR_EOE_INVALID_CALLBACK_TYPE	Invalid callback type
0xC0D00003	ECM_ERROR_EOE_DESTINATION_UNREACHABLE	Destination unreachable
0xC0D00004	ECM_ERROR_EOE_INVALID_EOE_RESPONSE	Invalid EoE Response
0xC0D00005	ECM_ERROR_EOE_UNKNOWN_ERROR	SetIPParam/SetFilterParam: Unknown error
0xC0D00006	ECM_ERROR_EOE_UNSPECIFIED_ERROR	SetIPParam/SetFilterParam: Unspecified Error
0xC0D00007	ECM_ERROR_EOE_UNSUPPORTED_FRAME_TYPE	SetIPParam/SetFilterParam: Unsupported frame type
0xC0D00008	ECM_ERROR_EOE_NO_IP_SUPPORT	SetIPParam/SetFilterParam: No IP support
0xC0D00009	ECM_ERROR_EOE_DHCP_NOT_SUPPORTED	SetIPParam/SetFilterParam: DHCP not supported
0xC0D0000A	ECM_ERROR_EOE_NO_FILTER_SUPPORT	SetIPParam/SetFilterParam: No filter supported
0xC0D0000B	ECM_ERROR_EOE_TIMEOUT	EoE Timeout
0xC0D0000C	ECM_ERROR_EOE_SHUTTING_DOWN	EoE is shutting down
0xC0D0000D	ECM_ERROR_EOE_MASTER_ADDRESS_NOT_ALLOWED	EoE: Master address is not allowed to use here
0xC0D0000E	ECM_ERROR_EOE_CONFIGURATION_IS_NOT_OPEN	EoE: Configuration is not open

Hexadecimal Value	Definition	Description
0xC0D0000F	ECM_ERROR_EOE_CONFIGURATION_IS_ALREADY_OPEN	EoE: Configuration is already open
0xC0D00010	ECM_ERROR_EOE_DUPLICATE_IP_ADDRESS	EoE: Duplicate IP address
0xC0D00011	ECM_ERROR_EOE_DUPLICATE_MAC_ADDRESS_ON_MULTIPLE_PORTS	EoE: Duplicate MAC address on multiple ports
0xC0D00012	ECM_ERROR_EOE_FRAME_TOO_LARGE	EoE: Frame too large
0xC0D00013	ECM_ERROR_EOE_IF_INITIALIZATION_ERROR	EoE: Interface initialization error
0xC0D00014	ECM_ERROR_EOE_IF_NO_FRAME_AVAILABLE	EoE: No Frame available
0xC0D00015	ECM_ERROR_EOE_LINK_DOWN	EoE: Link down
0xC0D10002	ECM_ERROR_FOE_ERROR_UNKNOWN_ERROR	-
0xC0D10003	ECM_ERROR_FOE_INVALID_TRANSFER_HANDLE	FoE: Invalid transfer handle
0xC0D10004	ECM_ERROR_FOE_INVALID_TRANSFER_STATE	FoE: Invalid transfer state
0xC0D10005	ECM_ERROR_FOE_INVALID_SLAVE_STATION_ADDRESS	FoE: Invalid slave station address
0xC0D10006	ECM_ERROR_FOE_WRONG_SLAVE_STATE	FoE: Wrong slave state
0xC0D10007	ECM_ERROR_FOE_NO_MAILBOX_AVAILABLE	FoE: No mailbox available
0xC0D10008	ECM_ERROR_FOE_TRANSFER_ABORTED	FoE: Transfer has been aborted
0xC0D10009	ECM_ERROR_FOE_PROTOCOL_TIMEOUT	FoE: Protocol Timeout
0xC0D1000A	ECM_ERROR_FOE_TRANSFER_SEGMENT_TOO_LONG	FoE: Transfer segment is too long
0xC0D1000B	ECM_ERROR_FOE_MAILBOX_TRANSMIT_FAILED	FoE: Mailbox transmit failed
0xC0D1000C	ECM_ERROR_FOE_FILENAME_TOO_LONG	FoE: Filename is too long
0xC0D1000D	ECM_ERROR_FOE_BUFFER_EXCEEDED	FoE: Buffer is exceeded
0xC0D1000E	ECM_ERROR_FOE_FIRST_SEGMENT_SHOULD_NOT_BE_EMPTY	FoE: First segment should not be empty
0xC0D1000F	ECM_ERROR_FOE_SEGMENT_SHOULD_BE_EMPTY	FoE: Segment should be empty
0xC0D18000	ECM_ERROR_FOE_ERROR_NOT_DEFINED	FoE: Error Response: not defined
0xC0D18001	ECM_ERROR_FOE_ERROR_NOT_FOUND	FoE: Error Response: Not Found
0xC0D18002	ECM_ERROR_FOE_ACCESS_DENIED	FoE: Error Response: Access Denied

Hexadecimal Value	Definition	Description
0xC0D18003	ECM_ERROR_FOE_ERROR_DISK_FULL	FoE: Error Response: Disk full
0xC0D18004	ECM_ERROR_FOE_ERROR_ILLEGAL	FoE: Error Response: Illegal
0xC0D18005	ECM_ERROR_FOE_ERROR_PACKET_NUMBER_WRONG	FoE: Error Response: Packet number is wrong
0xC0D18006	ECM_ERROR_FOE_ERROR_ALREADY_EXISTS	FoE: Error Response: Already exists
0xC0D18007	ECM_ERROR_FOE_ERROR_NO_USER	FoE: Error Response: No User
0xC0D18008	ECM_ERROR_FOE_ERROR_BOOTSTRAP_ONLY	FoE: Acces to specified file is only allowed in BOOT state
0xC0D18009	ECM_ERROR_FOE_ERROR_NOT_BOOTSTRAP	FoE: Access to specified file is only allowed when in PREOP, SAFEOP or OP
0xC0D1800A	ECM_ERROR_FOE_ERROR_NO_RIGHTS	FoE: No Rights
0xC0D1800B	ECM_ERROR_FOE_ERROR_PROGRAM_ERROR	FoE: Program Error
0xC0D20001	ECM_ERROR_SOE_UNKNOWN_SOE_ERROR	SoE: Unknown SoE Error
0xC0D20002	ECM_ERROR_SOE_INITIALIZATION_ERROR	SoE: Initialization error
0xC0D20003	ECM_ERROR_SOE_INVALID_TRANSFER_HANDLE	SoE: Invalid transfer handle
0xC0D20004	ECM_ERROR_SOE_NO_MAILBOX_AVAILABLE	SoE: No Mailbox available
0xC0D20005	ECM_ERROR_SOE_INVALID_TRANSFER_STATE	SoE: Invalid transfer state
0xC0D20006	ECM_ERROR_SOE_TRANSFER_SEGMENT_TOO_LONG	SoE: Transfer segment is too long
0xC0D20007	ECM_ERROR_SOE_SHUTTING_DOWN	SoE is shutting down
0xC0D20008	ECM_ERROR_SOE_MAX_TOTAL_BYTES_SMALLER_THAN_ACTUAL_TOTAL_BYTES	SoE: Maximum total bytes is smaller than actual total bytes
0xC0D20009	ECM_ERROR_SOE_MAILBOX_TRANSMIT_FAILED	SoE: Mailbox transmit failed
0xC0D2000A	ECM_ERROR_SOE_INVALID_SOE_HEADER	SoE: Invalid SoE header
0xC0D2000B	ECM_ERROR_SOE_PROTOCOL_TIMEOUT	SoE: Protocol Timeout
0xC0D2000C	ECM_ERROR_SOE_PROTOCOL_ERROR	SoE: Protocol Error
0xC0D2000D	ECM_ERROR_SOE_TRANSFER_ABORTED	SoE: Transfer has been aborted
0xC0D2000E	ECM_ERROR_SOE_WRONG_SLAVE_STATE	SoE: Wrong slave state
0xC0D2000F	ECM_ERROR_SOE_NO_AOE_AVAILABLE	SoE: No AoE available

Hexadecimal Value	Definition	Description
0xC0D20010	ECM_ERROR_SOE_INVALID_SLAVE_STATION_ADDRESS	SoE: Invalid slave station address
0xC0D21001	ECM_ERROR_SOE_SSC_NO_IDN	SoE: No IDN
0xC0D21009	ECM_ERROR_SOE_SSC_INVALID_ACCESS_TO_ELEMENT_1	SoE: Invalid access to element 1
0xC0D22001	ECM_ERROR_SOE_SCC_NO_NAME	SoE: IDN has no name
0xC0D22002	ECM_ERROR_SOE_SSC_NAME_TRANSMISSION_IS_TOO_SHORT	SoE: Name transmission is too short
0xC0D22003	ECM_ERROR_SOE_SSC_NAME_TRANSMISSION_IS_TOO_LONG	SoE: Name transmission is too long
0xC0D22004	ECM_ERROR_SOE_SSC_NAME_CANNOT_BE_CHANGED	SoE: Name cannot be changed
0xC0D22005	ECM_ERROR_SOE_SSC_NAME_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Name is write protected at this time
0xC0D23002	ECM_ERROR_SOE_SSC_ATTRIBUTE_TRANSMISSION_IS_TOO_SHORT	SoE: Attribute transmission is too short
0xC0D23003	ECM_ERROR_SOE_SSC_ATTRIBUTE_TRANSMISSION_IS_TOO_LONG	SoE: Attribute transmission is too long
0xC0D23004	ECM_ERROR_SOE_SSC_ATTRIBUTE_CANNOT_BE_CHANGED	SoE: Attribute cannot be changed
0xC0D23005	ECM_ERROR_SOE_SSC_ATTRIBUTE_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Attribute is write protected at this time
0xC0D24001	ECM_ERROR_SOE_SSC_NO_UNIT	SoE: IDN has no unit
0xC0D24002	ECM_ERROR_SOE_SSC_UNIT_TRANSMISSION_IS_TOO_SHORT	SoE: Unit transmission is too short
0xC0D24003	ECM_ERROR_SOE_SSC_UNIT_TRANSMISSION_IS_TOO_LONG	SoE: Unit transmission is too long
0xC0D24004	ECM_ERROR_SOE_SSC_UNIT_CANNOT_BE_CHANGED	SoE: Unit cannot be changed
0xC0D24005	ECM_ERROR_SOE_SSC_UNIT_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Unit is write protected at this time
0xC0D25001	ECM_ERROR_SOE_SSC_NO_MAXIMUM_VALUE	SoE: IDN has no maximum value
0xC0D25002	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT	SoE: Minimum value transmission is too short
0xC0D25003	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_TRANSMISSION_IS_TOO_LONG	SoE: Minimum value transmission is too long
0xC0D25004	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_CANNOT_BE_CHANGED	SoE: Minimum value cannot be changed

Hexadecimal Value	Definition	Description
0xC0D25005	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Mimum value is write protected at this time
0xC0D26001	ECM_ERROR_SOE_SSC_NO_MAXIMUM_VALUE	SoE: IDN has no maximum value
0xC0D26002	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT	SoE: Maximum value transmission is too short
0xC0D26003	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_LONG	SoE: Maximum value transmission is too long
0xC0D26004	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_CANNOT_BE_CHANGED	SoE: Maximum value cannot be changed
0xC0D26005	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Maximum value is write protected at this time
0xC0D27002	ECM_ERROR_SOE_SSC_OPDATA_TRANSMISSION_IS_TOO_SHORT	SoE: OpData transmission is too short
0xC0D27003	ECM_ERROR_SOE_SSC_OPDATA_TRANSMISSION_IS_TOO_LONG	SoE: OpData transmission is too long
0xC0D27004	ECM_ERROR_SOE_SSC_OPDATA_CANNOT_BE_CHANGED	SoE: OpData cannot be changed
0xC0D27005	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: OpData is write protected at this time
0xC0D27006	ECM_ERROR_SOE_SSC_OPDATA_IS_LOWER_THAN_MINIMUM_VALUE	SoE: OpData is lower than minimum value
0xC0D27007	ECM_ERROR_SOE_SSC_OPDATA_IS_HIGHER_THAN_MAXIMUM_VALUE	SoE: OpData is higher than maximum value
0xC0D27008	ECM_ERROR_SOE_SSC_OPDATA_IS_INVALID	SoE: OpData is invalid
0xC0D27009	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_BY_PASSWORD	SoE: OpData is write protected by password
0xC0D2700A	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_DUE_CYCLICALLY_CONFIGURED	SoE: OpData is write protected due to being cyclically configured
0xC0D2700B	ECM_ERROR_SOE_SSC_OPDATA_INVALID_DIRECT_ADDRESSING	SoE: Invalid direct addressing
0xC0D2700C	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_DUE_OTHER_SETTINGS	SoE: OpData is write protected due to other settings.
0xC0D2700D	ECM_ERROR_SOE_SSC_OPDATA_INVALID_FLOATING_POINT_NUMBER	SoE: Invalid floating point number

Hexadecimal Value	Definition	Description
0xC0D2700E	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_PARAMETERIZATION_LEVEL	SoE: OpData is write protected at parameterization level
0xC0D2700F	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_OPERATION_LEVEL	SoE: OpData is write protected at operation level
0xC0D27010	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_ALREADY_ACTIVE	SoE: Procedure command already active
0xC0D27011	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_NOT_INTERRUPTIBLE	SoE: Procedure command not interruptible
0xC0D27012	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_NOT_EXECUTABLE_AT_THIS_TIME	SoE: Procedure command is not executable at this time
0xC0D27013	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_NOT_EXECUTABLE_INVALID_PARAM	SoE: Procedure command is not executable due to invalid parameter
0xC0D4005C	ECM_ERROR_ENI_NO_SLAVES_IN_ENI	ENI does not contain any slaves
0xC0D50001	ECM_ERROR_ALSTATCODE_UNSPECIFIED_ERROR	ALStatusCode: Unspecified error
0xC0D50002	ECM_ERROR_ALSTATCODE_NO_MEMORY	ALStatusCode: No memory
0xC0D50003	ECM_ERROR_ALSTATCODE_INVALID_DEVICE_SETUP	ALStatusCode: Invalid Device Setup
0xC0D50011	ECM_ERROR_ALSTATCODE_INVALID_REQUESTED_STATE_CHANGE	ALStatusCode: Invalid requested state change
0xC0D50012	ECM_ERROR_ALSTATCODE_UNKNOWN_REQUESTED_STATE	ALStatusCode: Unknown requested state
0xC0D50013	ECM_ERROR_ALSTATCODE_BOOTSTRAP_NOT_SUPPORTED	ALStatusCode: Bootstrap not supported
0xC0D50014	ECM_ERROR_ALSTATCODE_NO_VALID_FIRMWARE	ALStatusCode: No valid firmware
0xC0D50015	ECM_ERROR_ALSTATCODE_INVALID_BOOT_MAILBOX_CONFIGURATION	ALStatusCode: Invalid BOOT mailbox configuration
0xC0D50016	ECM_ERROR_ALSTATCODE_INVALID_PREOP_MAILBOX_CONFIGURATION	ALStatusCode: Invalid PREOP mailbox configuration
0xC0D50017	ECM_ERROR_ALSTATCODE_INVALID_SYNC_MANAGER_CONFIGURATION	ALStatusCode: Invalid sync manager configuration
0xC0D50018	ECM_ERROR_ALSTATCODE_NO_VALID_INPUTS_AVAILABLE	ALStatusCode: No valid inputs available

Hexadecimal Value	Definition	Description
0xC0D50019	ECM_ERROR_ALSTAT-CODE_NO_VALID_OUTPUTS	ALStatusCode: No valid outputs
0xC0D5001A	ECM_ERROR_ALSTAT-CODE_SYNCHRONIZA-TION_ERROR	ALStatusCode: Synchronization error
0xC0D5001B	ECM_ERROR_ALSTAT-CODE_SYNC_MAN-AGER_WATCHDOG	ALStatusCode: Sync Manager watchdog
0xC0D5001C	ECM_ERROR_ALSTAT-CODE_INVALID_SYNC_MAN-AGER_TYPES	ALStatusCode: Invalid Sync Manager Types
0xC0D5001D	ECM_ERROR_ALSTAT-CODE_INVALID_OUTPUT_CON-FIGURATION	ALStatusCode: Invalid output configuration
0xC0D5001E	ECM_ERROR_ALSTAT-CODE_INVALID_INPUT_CONFIGU-RATION	ALStatusCode: Invalid input configuration
0xC0D5001F	ECM_ERROR_ALSTAT-CODE_INVALID_WATCHDOG_CO NFIGURATION	ALStatusCode: Invalid Watchdog configuration
0xC0D50020	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_COLD_STA RT	ALStatusCode: Slave needs cold start
0xC0D50021	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_INIT	ALStatusCode: Slave needs INIT
0xC0D50022	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_PREOP	ALStatusCode: slave needs PREOP
0xC0D50023	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_SAFEOp	ALStatusCode: slave needs SAFEOP
0xC0D50024	ECM_ERROR_ALSTAT-CODE_INVALID_INPUT_MAPPING	ALStatusCode: Invalid Input Mapping
0xC0D50025	ECM_ERROR_ALSTAT-CODE_INVALID_OUTPUT_MAP-PING	ALStatusCode: Invalid Output Mapping
0xC0D50026	ECM_ERROR_ALSTAT-CODE_INCONSISTENT_SET-TINGS	ALStatusCode: Inconsistent settings
0xC0D50027	ECM_ERROR_ALSTAT-CODE_FREERUN_NOT_SUP-PORTED	ALStatusCode: FreeRun not supported
0xC0D50028	ECM_ERROR_ALSTAT-CODE_SYNCMODE_NOT_SUP-PORTED	ALStatusCode: SyncMode not supported
0xC0D50029	ECM_ERROR_ALSTAT-CODE_FREERUN_NEEDS_3BUFF ER_MODE	ALStatusCode: FreeRun needs 3Buffer mode
0xC0D5002A	ECM_ERROR_ALSTAT-CODE_BACK-GROUND_WATCHDOG	ALStatusCode: Background Watchdog
0xC0D5002B	ECM_ERROR_ALSTAT-CODE_NO_VALID_INPUTS_AND_ OUTPUTS	ALStatusCode: No valid Inputs and Outputs

Hexadecimal Value	Definition	Description
0xC0D5002C	ECM_ERROR_ALSTAT-CODE_FATAL_SYNC_ERROR	ALStatusCode: Fatal Sync error
0xC0D5002D	ECM_ERROR_ALSTAT-CODE_NO_SYNC_ERROR	ALStatusCode: No Sync error
0xC0D50030	ECM_ERROR_ALSTAT-CODE_INVALID_DC_SYNC_CON-FIGURATION	ALStatusCode: Invalid DC SYNC configuration
0xC0D50031	ECM_ERROR_ALSTAT-CODE_INVALID_DC_LATCH_CON-FIGURATION	ALStatusCode: Invalid DC Latch configuration
0xC0D50032	ECM_ERROR_ALSTAT-CODE_PLL_ERROR	ALStatusCode: PLL error
0xC0D50033	ECM_ERROR_ALSTAT-CODE_DC_SYNC_IO_ERROR	ALStatusCode: DC Sync IO error
0xC0D50034	ECM_ERROR_ALSTAT-CODE_DC_SYNC_TIMEOUT_ERR OR	ALStatusCode: DC Sync Timeout Error
0xC0D50035	ECM_ERROR_ALSTAT-CODE_DC_INVALID_SYNC_CYCL E_TIME	ALStatusCode: DC Invalid Sync Cycle Time
0xC0D50036	ECM_ERROR_ALSTAT-CODE_DC_SYNC0_CYCLE_TIME	ALStatusCode: DC Sync0 Cycle Time
0xC0D50037	ECM_ERROR_ALSTAT-CODE_DC_SYNC1_CYCLE_TIME	ALStatusCode: DC Sync1 Cycle Time
0xC0D50041	ECM_ERROR_ALSTAT-CODE_MBX_AOE	ALStatusCode: MBX_AOE
0xC0D50042	ECM_ERROR_ALSTAT-CODE_MBX_EOE	ALStatusCode: MBX_EOE
0xC0D50043	ECM_ERROR_ALSTAT-CODE_MBX_COE	ALStatusCode: MBX_COE
0xC0D50044	ECM_ERROR_ALSTAT-CODE_MBX_FOE	ALStatusCode: MBX_FOE
0xC0D50045	ECM_ERROR_ALSTAT-CODE_MBX_SOE	ALStatusCode: MBX_SOE
0xC0D5004F	ECM_ERROR_ALSTAT-CODE_MBX_VOE	ALStatusCode: MBX_VOE
0xC0D50050	ECM_ERROR_ALSTAT-CODE_EEPROM_NO_ACCESS	ALStatusCode: EEPROM no access
0xC0D50051	ECM_ERROR_ALSTAT-CODE_EEPROM_ERROR	ALStatusCode: EEPROM error
0xC0D50060	ECM_ERROR_ALSTAT-CODE_SLAVE_RESTARTED_LOC ALLY	ALStatusCode: Slave restarted locally
0xC0D50061	ECM_ERROR_ALSTAT-CODE_DEVICE_IDENTIFICA-TION_VALUE_UPDATED	ALStatusCode: Device identificatin value updated
0xC0D500F0	ECM_ERROR_ALSTAT-CODE_APPLICATION_CON-TROLLER_AVAILABLE	ALStatusCode: Application controller available

Hexadecimal Value	Definition	Description
0xC0D58000	ECM_ERROR_ALSTAT-CODE_VENDOR_SPE-CIFIC_CODE_START	Begin of vendor-specific ALStatus-Code mapping
0xC0D5FFFF	ECM_ERROR_ALSTAT-CODE_VENDOR_SPE-CIFIC_CODE_END	End of vendor-specific ALStatus-Code mapping
0xC0D60001	ECM_ERROR_IF_COE_SUP-PORT_NOT_AVAILABLE	CoE support is not configured
0xC0D60002	ECM_ERROR_IF_SOE_SUP-PORT_NOT_AVAILABLE	SoE support is not configured
0xC0D60003	ECM_ERROR_IF_FOE_SUP-PORT_NOT_AVAILABLE	FoE support is not configured
0xC0D60004	ECM_ERROR_IF_AOE_SUP-PORT_NOT_AVAILABLE	AoE support is not configured
0xC0D60005	ECM_ERROR_IF_INVALID_TRANS PORT_TYPE	Invalid transfer type
0xC0D60006	ECM_ERROR_IF_SOE_INVALID_D RIVE_NO	SoE: Invalid drive number
0xC0D60007	ECM_ERROR_IF_SOE_INVALID_E LEMENT_FLAGS	SoE: invalid element flags
0xC0D60008	ECM_ERROR_IF_INVALID_SOE_T RANSFER_ID	SoE: Invalid transfer ID
0xC0D60009	ECM_ERROR_IF_TRANSFER_AB ORTED	Transfer aborted
0xC0D6000A	ECM_ERROR_IF_OUT_OF_PACKE TS	Out of packets
0xC0D6000B	ECM_ERROR_IF_OUT_OF_TRAN SFER_CONTEXTS	Out of transfer contexts
0xC0D6000C	ECM_ERROR_IF_INVALID_SUB-INDEX_FOR_COMPLETE_ACCESs	CoE: Invalid subindex for Complete Access
0xC0D6000D	ECM_ERROR_IF_INVALID_COE_T RANSFER_ID	CoE: Invalid transfer ID
0xC0D6000E	ECM_ERROR_IF_INVALID_COE_S DOINFO_LISTTYPE	CoE: Invalid SDOINFO ListType
0xC0D6000F	ECM_ERROR_IF_FILE_READ_ER ROR	File Read Error
0xC0D60010	ECM_ERROR_IF_COULD_NOT_O PEN_FILE	Could not open file
0xC0D60011	ECM_ERROR_IF_INVALID_CONFI G_NXD	Invalid config.nxd detected
0xC0D60012	ECM_ERROR_IF_CONFIG_NXD_ WITHOUT_SLAVES	Config.nxd does not contain any slaves
0xC0D60013	ECM_ERROR_IF_INVALID_FILE_N AME	Invalid file name
0xC0D60014	ECM_ERROR_IF_INVALID_FOE_T RANSFER_ID	Invalid FoE transfer id
0xC0D60015	ECM_ERROR_IF_INVALID_GET_T OPOLOGY_TRANSFER_ID	Invalid GetTopology transfer id

1.7.3.4.2 CM592-DP PROFIBUS DP master diagnosis

In Automation Builder, diagnosis messages of communication module CM592-DP are displayed at device tree node *“CM592-DP”* and all nodes below, slave devices and I/O modules.

Click tab *“Diagnosis”*.

Within PLC application, diagnosis messages can be read by diagnosis related methods of function block type *“Diag”*, provided in library *“Diag”*. Furthermore, at CM592-DP specific I/O driver function block and slave and I/O module specific function blocks. *Chapter 1.7.1.4 “Diagnosis in IEC application” on page 4020*

In PLC display, diagnosis messages of CM592-DP are not shown.

Following diagnosis messages are signaled by CM592-DP.

CM592 communication module specific diagnosis messages:

Severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0	0	655360	Watchdog error communication module	
3	0	0	655361	Firmware version of CM592-DP not supported	Update firmware
3	0	0	655362	Configuration error	Check configuration and correct errors
3	0	0	655363	CM592-DP not found	Plug correct communication module
3	0	0	655364	CM592-DP has wrong type	Plug correct communication module
4	0	0	655365	No PROFIBUS slave device configured	Check configuration
4	0	0	655366	No PROFIBUS slave IO channel configured	Check configuration
3	0	0	655367	Configuration version mismatch	Use matching CPU firmware version
3	0	0	655368	Diagnosis lost, could not save additional diagnosis data	Check configuration - too many active diagnosis messages received
3	0	0	655369	CM592-DP is not communicating	Check bus connection and configuration
3	0	0	655370	CM592-DP signals communication error	Check bus connection and configuration
3	0	0	655371	Starting CM592-DP's protocol stack failed	Check bus connection and configuration

Severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0	0	655372	Stopping CM592-DP's protocol stack failed	
3	0	0	655373	PLC cannot be set to RUN due to error at CM592-DP	Check error log and correct errors
3	0	0	655374	CI54x communication interface module is sending not supported diagnosis format	Check configuration and FW revision of communication interface module

PROFIBUS standard diagnosis messages:

Severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0	0	65536	Standard diagnosis message received	Check additional data for details
3	0	0	65537	Slave device offline	Check if slave device is connected physically and up and running
3	0	0	65538	Slave device reports error in configuration data	Check if slave device description data (GSD) is up-to-date
3	0	0	65539	Slave device reports error in parameter data	Check if slave device description data (GSD) is up-to-date
3	0	0	65540	Slave device cannot provide valid data	Check slave device status
3	0	0	65541	Slave device reports extended diagnosis overflow	Start with resolving root causes of available diagnosis messages
3	0	0	196608	Identifier diagnosis message received	Check additional data for details
3	0	0	262144	Device diagnosis DPV0 format received	Check additional data for details
3	0	0	327680	Device diagnosis DPV1 alarm received	Check additional data for details
3	0	0	393216	Device diagnosis DPV1 status received	Check additional data for details

PROFIBUS channel diagnosis messages:

Severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0 - 63	0	131072	Channel diagnosis, channel x, Reserved error code y	
3	0 - 63	0	131073	Channel diagnosis, channel x, Short circuit	
3	0 - 63	0	131074	Channel diagnosis, channel x, Undervoltage	
3	0 - 63	0	131075	Channel diagnosis, channel x, Overvoltage	
3	0 - 63	0	131076	Channel diagnosis, channel x, Overload	
3	0 - 63	0	131077	Channel diagnosis, channel x, Overtemperature	
3	0 - 63	0	131078	Channel diagnosis, channel x, Line break	
3	0 - 63	0	131079	Channel diagnosis, channel x, Upper limit value exceeded	
3	0 - 63	0	131080	Channel diagnosis, channel x, Lower limit value exceeded	
3	0 - 63	0	131081	Channel diagnosis, channel x, Error	
3	0 - 63	0	131082 - 131087	Channel diagnosis, channel x, Reserved error code y	
3	0 - 63	0	131088 - 131103	Channel diagnosis, channel x, Manufacturer specific error y	

ABB Communication Interface Module (CI54x) specific diagnosis messages:

Severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	255	0	8722	Internal error	
3	255	0	8732	Internal error	

4	255	0	9480	I/O module removed from hot swap terminal unit or defective module on hot swap terminal unit	Plug I/O module, replace I/O module
4	255	0	9500	Wrong I/O module plugged on hot swap terminal unit	Remove wrong I/O module and plug projected I/O module
4	255	0	9514	No communication with I/O module on hot swap terminal unit	Replace I/O module
4	255	0	9526	I/O module does not support hot swap	Power off system and replace I/O module
4	255	0	9736	Hot swap terminal unit required but not found	Plug hot swap terminal unit
4	255	0	9764	Defective hot swap terminal unit	
4	255	0	9770	No communication with hot swap terminal unit	Restart, if error persists replace terminal unit
3	255	0	16131	Timeout	Replace I/O module
3	255	0	16137	Overflow diagnosis buffer	Restart
4	255	0	16138	Voltage overflow at outputs (above UP3 level)	Check terminals / check process supply voltage
3/4	255	0	16139	Process voltage UP or UP3 too low	Check process supply voltage
3	255	0	16145	No communication with I/O module	Replace I/O module
3	255	0	16147	Checksum error	Replace I/O module
3	255	0	16154	Parameter error	Check configuration
4	255	0	16159	At least one module does not support failsafe function	Check modules and parameterization

3	255	0	16160	Wrong I/O module type on socket	Replace I/O module / check configuration
4	255	0	16162	No response during initialization of the I/O module	Replace I/O module
3	255	0	16164	Internal data exchange failure	Replace I/O module
3	255	0	16168	Different hard-/firmware versions in the module	Replace I/O module
3	255	0	16171	Internal error	Replace I/O module
3/4	255	0	16173	No process voltage UP or UP3	Check process voltage
4	255	0	16174	Voltage feedback on activated digital outputs DO0...DO7 on UP3	Check terminals
4	255	0	16175	Sensor voltage too low	
3	0 - 31	0	18	Test error	
4	0 - 31	0	257	Wrong measurement, false temperature at the compensation channel	
4	0 - 31	0	258	AI531: Wrong measurement; potential difference is too high; CD522: PWM duty cycle out of duty area	
4	0 - 31	0	260	Measurement overflow	Check channel wiring and sensor power supply
4	0 - 31	0	263	Measurement underflow at analog input	Check channel wiring and sensor power supply
4	0 - 31	0	266	Short circuit and cut wire or "out of range"	
4	0 - 31	0	267	Output/process voltage to small/low	
3	0 - 31	0	273	Test error	
4	0 - 31	0	303	Short circuit at an analog input	Check channel wiring

4	0 - 31	0	304	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	0 - 31	0	530	Internal fuse at 0V is defect. 0V not connected with GND	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module
3	0 - 31	0	540	Test error	
3	0 - 31	0	555	Internal error	
4	0 - 31	0	558	Externally voltage detected on digital output DO0...DO7	Check terminals
4	0 - 31	0	559	Short circuit at digital output	Check channel wiring
4	0 - 31	0	772	Analog value overflow at an analog output	Check output value
4	0 - 31	0	775	Analog value underflow at an analog output	Check output value
4	0 - 31	0	796	Different configuration	
3	0 - 31	0	1037	Test error	
4	0 - 31	0	1070	Externally voltage detected on digital output DC0...DC7	Check terminals
4	0 - 31	0	1071	Short circuit at digital output	Check terminals

1.7.3.4.3 CM582-DP PROFIBUS DP slave diagnosis

The diagnosis messages of the communication module CM582-DP are displayed in the tab *“Diagnosis ”* of node *“CM582-DP”* in the device tree of the Automation Builder. Within PLC application they can be read with the diagnosis methods of IO driver or function block *“Diag”*.

In the PLC display the diagnosis messages of CM582-DP are not shown.

The following diagnosis messages are signaled by CM582-DP:

Error severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0	0	1000x	No communication module or wrong type found	Plug the correct communication module
3	0	0	1001	Type of CM582-DP not supported	Exchange the communication module
3	0	0	1002	Firmware version of CM582-DP not supported	Update firmware of CM582-DP

Error severity	SubSysteminfo	Additional	Error code	Meaning	Remedy
3	0	0	1003	Identification of communication module failed	Exchange the communication module or plug the correct communication module
3	0	0	2000	Watchdog error	-
3	0	0	2001	CM582-DP is not communicating	Check bus connection and configuration
3	0	0	2002	CM582-DP signals communication error	Check bus connection and configuration
3	0	0	2003	Starting of CM582-DP's protocol stack failed	Check bus connection and configuration
3	0	0	2004	Stopping of CM582-DP's protocol stack failed	-
3	0	0	2005	PLC cannot be set to run due to an error of CM582-DP	Check error log and correct errors
3	0	0	3000	Configuration error	Check configuration and correct errors
3	0	0	3001	Configuration version mismatch	Use matching CPU firmware version
3	0	0	3002	Writing parameters to CM582-DP failed	Check configuration and correct errors
3	0	0	3003	Configuration of IM0 data failed	Check configuration and correct errors
3	0	0	3004	Reading of a parameter failed	Check configuration and correct errors
3	0	0	3005	Parameter value not supported or out of limits	Check configuration and correct errors

1.7.3.4.4 AC500-S: errors from safety CPU and safety I/O modules

Table 754: Error messages for safety CPU

Severity	Error code	Description	Remedy
2	8235	Internal error	Replace module
2	8448	Operation finished	Change Safety PLC switch address setting or remove memory card from non-safety PLC. Restart Safety PLC. If this error persists, replace Safety PLC.
2	8449	Wrong user data	Delete user data from Safety PLC. Restart Safety PLC and write user data again.
2	8450	Internal PROFIsafe initialization error	Restart Safety PLC. If this error persists, replace Safety PLC. Contact ABB technical support.
2	8460	Flash read error	Restart Safety PLC. If this error persists, replace Safety PLC. Contact ABB technical support.

Severity	Error code	Description	Remedy
2	8466	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8476	Boot project download error	Reload boot project. If this error persists, replace Safety PLC.
2	8488	Wrong firmware version	Update Safety PLC firmware. Restart Safety PLC. If this error persists, replace Safety PLC.
2	8491	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8496	Overvoltage or under-voltage detected	Restart Safety PLC. Check Safety PLC setting for power supply error. If this error persists, replace Safety PLC.
2	8500	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8704	User program triggered safe stop	Check user program
2	8705	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8706	Internal PROFIsafe error	Restart Safety PLC. If this error persists, replace Safety PLC. Contact ABB technical support.
2	8707	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8714	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8717	Flash write error	Restart Safety PLC. If this error persists, replace Safety PLC. Contact ABB technical support.
2	8721	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8722	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8723	Checksum error has occured in Safety PLC	Restart Safety PLC. If this error persists, replace Safety PLC.
2	8729	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8741	Cycle time error in Safety PLC	Check Safety PLC watchdog time.
2	8742	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8746	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8747	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8756	Internal error	Contact ABB technical support. Replace Safety PLC.
2	8758	Internal error	Contact ABB technical support. Replace Safety PLC.

Severity	Error code	Description	Remedy
2	8990	PROFIsafe configuration error	Check F-Parameter configuration of I/O module and reload boot project
3	12561	Safety source addresses cannot be checked	Check PROFIsafe F-Host library version (2.0.0 or above). If this error persists, contact ABB technical support.
3	12570	Error in configuration data, safety PLC has not accepted configuration data, e.g., mismatch between safety and non-safety PLC configuration.	Create new configuration data for both safety and non-safety PLC again, re-create and download boot projects to both safety and non-safety PLC again.
3	12571	Error in configuration data, Safety PLC cannot read configuration data	Create boot project
3	12598	PROFIsafe F_Dest_Add rules are violated	Check Safety PLC configuration or switch address setting against PROFIsafe F_Dest_Add configuration rules. Restart Safety PLC. If this error persists, contact ABB technical support.
3	32770	Watchdog error communication module	
3	32771	Wrong firmware version of communication module	Update firmware
3	32772	Initialisation of safety module on slot failed. More than one safety module plugged	Remove this module or Only that one safety module plugged -> defective, replace this module
3	32774	Invalid configuration data	Check configuration
3	32775	Safety module not found	Check configuration. At Safety PLC: Check Safety PLC switch address setting. Restart Safety PLC. If this error persists, replace Safety PLC.
3	32776	Safety module has wrong type	Check configuration
4	16640	Reserved switch address setting.	Warning
4	16644	Boot project not loaded, maximum power dip reached	Restart Safety PLC
4	16648	Power dip data missed or corrupted. Default power dip data was flashed by Safety PLC	Warning
4	16659	CRC error boot project	Create new boot project and restart Safety PLC
4	16909	Flash write error (production data)	Warning

Severity	Error code	Description	Remedy
4	16935	More than one instance of SF_WDOG_TIME_SET or SF_MAX_POWER_DIP_SET	Warning
4	16922	No or wrong configuration data from PM5x, run state not possible	Create correct boot project at PM5x
4	17421	Flash write error (boot project)	Warning
4	17677	Flash write error (boot code)	Warning
4	17933	Flash write error (firmware)	Warning
4	18189	Flash write error (password)	Warning
4	18445	Flash write error (user data)	Warning
4	18701	Flash write error (user data)	Warning
4	18957	Flash write error (internal)	Warning
4	19213	Flash write error (internal)	Warning
4	19469	Flash write error (internal)	Warning
4	32777	Program not started because of configuration error	Check configuration
4	32778	Program not started, no application running in safety module	Check configuration, download safety application to safety module

Table 755: Error messages for safety I/O modules (channel or module reintegration is possible)

Severity	Error code	Description	Remedy
3	3	Discrepancy time expired	Check discrepancy time value, channel wiring and sensor.
3	12	Test pulse error	Check wiring and sensor.
3	13	Channel test pulse cross-talk error	Check wiring and sensor. If this error persists, replace I/O module. Contact ABB technical support.
3	25	Channel stuck-at error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	28	Channel cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	260	Measurement overflow at the I/O module	Check channel wiring and sensor power supply.
3	263	Measurement underflow at the I/O module	Check channel wiring and sensor power supply.

Severity	Error code	Description	Remedy
3	311	Channel value difference too high	Adjust tolerance window for channels. Check channel wiring and sensor configuration.
3	525	Channel readback error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	530	Channel cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
3	16138	Process voltage too high	Check process voltage
3	16139	Process voltage too low	Check process voltage
3	16148	PROFIsafe communication error	Restart I/O module. If this error persists, contact ABB technical support.
3	16153	PROFIsafe watchdog timed out.	Restart I/O module. If this error persists, increase PROFIsafe watchdog time.
3	16171	Internal error in the device	Replace I/O module

Table 756: Error messages for safety I/O modules (channel or module reintegration ist not possible)

Severity	Error code	Description	Remedy
3	16146	Plausibility check failed (iParameter)	Check configuration
3	16147	Checksum error in the I/O module	Check safety configuration and CRCs for I- and F-Parameters.
3	16154	Parameter value	Check master or configuration
3	16156	F-Parameter configuration and address switch value do not match.	Check I/O module F-Parameter configuration and module address switch value.

1.7.3.4.5 CM579-PNIO – PROFINET I/O controller diagnosis

Diagnosis data for CM579-PNIO is not displayed in PLC display. In Automation Builder, we recommend to use methods with text output to get diagnosis messages in clear text format. E.g., `DiagGetFirstValAndTxt`.

Output string:

<timestamp>; <error severity>; <device name>; <error location>; error ID <id>: <error text>

For experts: manual interpretation

If you need to access the diagnosis data directly, you have to interpret them manually. Refer to the example to learn how to interpret them correctly ↯ *Chapter 1.7.3.4.5.1 “Manual interpretation of CM579-PNIO diagnosis” on page 4111*.

Diagnosis messages are included in diagnosis text lists *“Diag_PNIO_Controller”* and *“Diag_PNIO_Vendor ID_Device ID”*.

Error severity	SubSysteminfo		Additional		Error code		Meaning	Remedy
	Word 2 (bit 16..31)	Word 1 (bit 0..15)	Word 2 (bit 16..31) ADD_SUB_TYPE	Word 1 (bit 0..15) ADD_TYPE	Word 2 (bit16...31)	Word 1 (bit0...15)		
	Sub1_	Sub2_	Add_Word 1_Word 2		Err_x or Err_Word 1_Word 2 (Word1/2 in hex format)			
3	tbd	tbd	0	1 (general)	Err_Gen_x		General error, <Err_Gen_x_text>	
3	0	0	0	2 (runtime)	Err_Rt_x		General error, <Err_Rt_x_text>	
3	0	0	0	2 (runtime)	1		Runtime error; communication module watchdog error	
3	0	0	0	2 (runtime)	2		Runtime error; PROFINET controller is not communicating	
3	0	0	0	2 (runtime)	3		Runtime error; PROFINET controller signals communication error	
4	0	0	0	2 (runtime)	4		Connection error; No connection to PROFINET I/O device	
11	tbd	tbd	0	3 (configuration)	Err_Cfg_x		Configuration error, <Err_Cfg_x_text>	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	1 (USI: 16#8000) channel diagnosis	4 (diagnosis alarm)	0	Error type	Subslot <subslot idx>, channel <channel idx>, channel diagnosis; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	1 (USI: 16#8000) channel diagnosis	4 (diagnosis alarm)	0	Error type	Subslot <subslot idx>, channel diagnosis; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	2 (USI: 16#8002) extended channel diagnosis	4 (diagnosis alarm)	Extended error Type	Error type	Subslot <subslot idx>, channel <channel idx>, extended channel diagnosis; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	2 (USI: 16#8002) extended channel diagnosis	4 (diagnosis alarm)	Extended error Type	Error type	Subslot <subslot idx>, extended channel diagnosis; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	3 (USI: 16#8003) qualified channel diagnosis	4 (diagnosis alarm)	Extended error Type	Error type	Subslot <subslot idx>, channel <channel idx>, qualified channel diagnosis; <error text>	

Error severity	SubSysteminfo		Additional		Error code		Meaning	Remedy
	Word 2 (bit 16..31)	**Word 1 (bit 0..15)**	**Word 2 (bit 16..31) ADD_SUB_TYPE**	**Word 1 (bit 0..15) ADD_TYPE**	**Word 2 (bit16...31)**	**Word 1 (bit0...15)**		
	Sub1_	**Sub2_**	**Add_Word 1_Word 2**		**Err_x or Err_Word 1_Word 2 (Word1/2 in hex format)**			
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	3 (USI: 16#8003) qualified channel	4 (diagnosis alarm)	Extended error Type	Error type	Subslot <subslot idx>, qualified channel diagnosis; <error text>	
2, 3, 4 11	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	0	5 (S500 process alarm)	32 bit error code		Subslot <subslot idx>, channel <channel idx>, S500 diagnosis; <error text>	
2, 3, 4 11	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	0	5 (S500 process alarm)	32 bit error code		Subslot <subslot idx>, S500 diagnosis; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	Alarm type = (14 .. 30) & (32 ..)	6 (alarm)	Alarm type = (14 .. 30) & (32 ..)		Subslot <subslot idx>, channel <channel idx>, PNIO alarm; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = (14 .. 30) & (32 ..)	6 (alarm)	Alarm type = (14 .. 30) & (32 ..)		Subslot <subslot idx>, PNIO alarm; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	Alarm type = 1	6 (alarm)	Alarm type = 1		Subslot <subslot idx>, channel <channel idx>, diagnosis alarm; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 1	6 (alarm)	Alarm type = 1		Subslot <subslot idx>, diagnosis alarm; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	Alarm type = 2	6 (alarm)	Alarm type = 2		Subslot <subslot idx>, channel <channel idx>, process alarm; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 2	6 (alarm)	Alarm type = 2		Subslot <subslot idx>, process alarm; <error text>	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 3	6 (alarm)	Alarm type = 3		Subslot <subslot idx>, pull alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 4	6 (alarm)	Alarm type = 4		Subslot <subslot idx>, plug alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 5	6 (alarm)	Alarm type = 5		Subslot <subslot idx>, status alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 6	6 (alarm)	Alarm type = 6		Subslot <subslot idx>, update alarm	

Error severity	SubSysteminfo		Additional		Error code		Meaning	Remedy
	Word 2 (bit 16..31)	**Word 1 (bit 0..15)**	**Word 2 (bit 16..31) ADD_SUB_TYPE**	**Word 1 (bit 0..15) ADD_TYPE**	**Word 2 (bit16...31)**	**Word 1 (bit0...15)**		
	Sub1_	**Sub2_**	**Add_Word 1_Word 2**		**Err_x or Err_Word 1_Word 2 (Word1/2 in hex format)**			
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 7	6 (alarm)	Alarm type = 7		Subslot <subslot idx>, rendundancy status changed alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 8	6 (alarm)	Alarm type = 8		Subslot <subslot idx>, supervisor controlled alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 9	6 (alarm)	Alarm type = 9		Subslot <subslot idx>, supervisor released alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 10	6 (alarm)	Alarm type = 10		Subslot <subslot idx>, wrong submodule plugged alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 11	6 (alarm)	Alarm type = 11		Subslot <subslot idx>, wrong submodule returned alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	Alarm type = 12	6 (alarm)	Alarm type = 12		Subslot <subslot idx>, channel <channel idx>, diagnosis disappeared alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 12	6 (alarm)	Alarm type = 12		Subslot <subslot idx>, diagnosis disappeared alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 13	6 (alarm)	Alarm type = 13		Subslot <subslot idx>, port data changed alarm	
3	Subslot index (0 – 16#9FFF)	Channel index (16#8000)	Alarm type = 31	6 (alarm)	Alarm type = 31		Used module pulled alarm	

Manual interpretation of CM579-PNIO diagnosis

For better understanding, we show the manual interpretation of CM579-PNIO diagnosis with an example.

System: AC500 CM579-PNIO + CI501-PNIO + optional S500 I/O inserted as PROFINET standard device

Error: discrepancy time expired (class 3, error ID 3) at channel 4 of first attached S500 I/O device on CI501-PNIO

For comparison: If a method with text output is used, e.g. DiagGetFirstValAndTxt the text output will be the string: `Timestamp; E3; device name; subslot 1, channel 4, extended channel diagnosis; error ID 3: discrepancy time expired (class 3, error ID 3)`

In Automation Builder the following error entry data is displayed:

Diag_Val[1]	AC500_DiagTypes.D...		
dtTimestamp	DATE_AND_TIME	DT#2019-10-15-14:48:34	RTC time of event
eClass	TECLASS	eDiagClass_3_Error	Severity of error event
szDevice	STRING(80)	'M0_PN03_M2'	Name of device
eHwInterfaceId	TEHWID	eDiagHwId_Coupler2	Identifier of hardware interface
dwSubSystemInfo	DWORD	65540	Any number describin...ail/location within d...
dwAdditional	DWORD	131076	Additional number des...ng detail/location wi...
dwErrorCode	DWORD	196867	Actual error code
wSizeExtDiag	WORD	44	Number of bytes of extended diagnosis data
pExtDiagData	POINTER TO BYTE	16#0079EA40	internal reference ne... for acknowledgement
hSource	POINTER TO BYTE	16#0077ABE8	internal reference needed for text convertion
pConn	POINTER TO IoStan...	16#B573DDD8	internal reference needed for text convertion

Analyze the data in the following order: Element *"dwAdditional"* for the type of diagnosis, element *"Error Code"*, element *"SubSysteminfo"*.

Type of diagnosis

1. Analyze element *"dwAdditional"*, column "Additional" in error lists.

   Convert the given value from decimal to hexadecimal format.

   "dwAdditional" = 131076 = 16#20004
2. Interpretation:

   Word 2 = 2

   Word 1 = 4
3. Generate error text Add_Word 1_Word 2 = Add_4_2
4. Look up which error type it is.

   Add_4_2 = extended channel diagnosis

Data analysis

1. Analyze element *"Error Code"*:

   "dwErrorCode" = 196867 = 16#30103 →

   Word 2 (extended error type) = 16#0003

   Word 1 (error type) = 16#0103

   ➔ Error text: Err_Word 1 (hex)_Word 2 (hex)

   ➔ Err_0103_0003 →

   Error ID 3 – discrepancy time expired (class 3, error ID 3)
2. Analyze *"SubSysteminfo"*

   "dwSubSysteminfo" = 65540 = 16#10004 →

   Word 2 (subslot index) = 1

   Word 1 (channel index) = 4

Error severity	SubSysteminfo		Additional		Error code		Meaning	Remedy
	Word 2 (bit 16..31)	Word 1 (bit 0..15)	Word 2 (bit 16..31) ADD_SUB_TYPE	Word 1 (bit 0..15) ADD_TYPE	Word 2 (bit16...31)	Word 1 (bit0...15)		
3	Subslot index (0 – 16#9FFF)	Channel index (0 – 16#7FFF)	2 (USI: 16#8002) extended channel diagnosis	4 (diagnosis alarm)	Extended error type	Error type	Subslot <subslot idx>, channel <channel idx>, extended channel diagnosis; <error text>	
3	1	4	2	4	16#0003	16#0103	Subslot 1, channel 4, extended channel diagnosis; <error text>	
	dwSubSysteminfo		Add_Word 1_Word 2		dwErrorCode		Entries in Diag values	
3	Sub2_ = Subslot	Sub1_ = Channel	Add_4_2 = extended channel diagnosis		Err_Word 1_Word 2 Err_0103_0003 = error ID 3: Discrepancy time expired (class 3, error ID 3)		Text search criteria (ID) in text list Diag_PNIO_26_22 Text from Automation Builder error list (default) in text list Diag_PNIO_26_22	

1.8 Engineering interfaces and tools

1.8.1 Export and import interfaces

1.8.1.1 Exporting and importing ECAD data (PBF)

Automation Builder provides an ECAD interface for exchanging the PLC configuration data with EPLAN Electric P8 and Zuken E3. This feature removes double data entry between electrical engineering in the ECAD tool and the control logic programming in Automation Builder by synchronizing the PLC hardware including topology and I/O signals between these tools.

Automation Builder - ECAD interface supports various flexible workflows:

- Enables PLC hardware planning and configuration in the ECAD tool and allows importing the exported data from the ECAD tool through the *PBF* file (process integration bus interchange format) into the Automation Builder project with diff and merge functionality, providing full control on selective import/merge.
- Enables PLC hardware configuration in Automation Builder and allows exporting the configuration to the ECAD tool through a *PBF* file.
- Supports bi-directional roundtrip engineering with loss less data exchange between Automation Builder and the ECAD tool.

Automation Builder uses the rack information to identify the relations between:

- PLC and devices plugged to I/O bus or extension bus.
- Fieldbus slave and attached IO devices.

It is recommended to assign the PLC, IO devices, communication modules and fieldbus slaves properly to the rack in the ECAD project. If the rack information is missing, devices will be imported to the device pool and must be arranged manually in the Automation Builder project or mapped to already existing devices.

1.8.1.1.1 Requirements on EPLAN electric P8

- EPLAN Electric P8 with PLC and Bus Extension. It is recommended to use version 2.3 or later.
- Use of appropriate part data and macros for ABB devices. This can be achieved by getting the part data and macros from the EPLAN data portal.

1.8.1.1.2 Importing PLC data from the ECAD tool

You can create a new Automation Builder PLC project from the existing PLC hardware configuration in your ECAD tool, by importing the exported *PBF* file to Automation Builder.

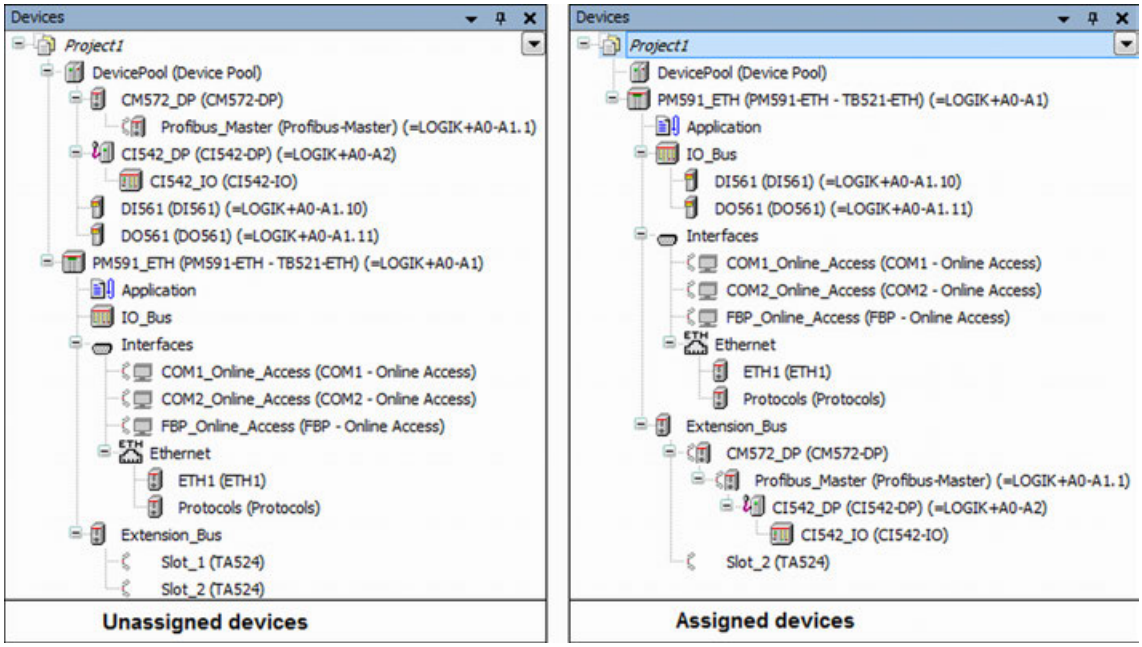
Import *PBF* file to Automation Builder

1. From the main menu, select *"Project ➔ Import ➔ ECAD (PBF)"*.
2. From the file system, select the *PBF* file.

   Automation Builder starts importing the devices and its associated signals from the *PBF* file. After a successful import, the result is displayed in the **Project Compare –Differences** view. You can now decide and selectively merge the differences.

   ↬ *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*
3. Select the DevicePool node and click *"Accept Block"* to accept the complete PLC structure in the ECAD tool.
4. Select the PLC node and click *"Accept Block"* to accept all child device nodes.

   > *The DevicePool node holds all devices coming from the ECAD tool without any hierarchy information. The missing hierarchy information can be defined after closing the editor.*
5. Close the **Project Compare – Differences** view to accept the changes.
6. Arrange unassigned devices in the DevicePool to the PLC hardware structure by drag-and-drop.



⇨ The I/O signals assigned to I/O devices in the *PBF* file are imported and allocated to IO devices. IO signals can be viewed in I/O mapping editor of the I/O devices.

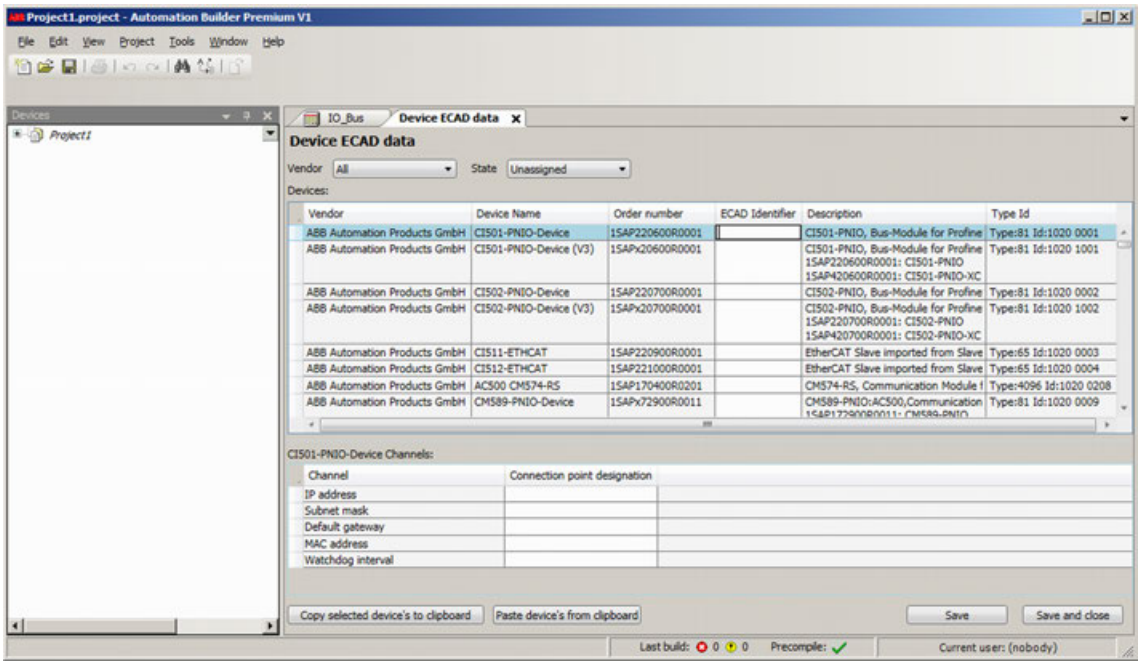
1.8.1.1.3 Importing third party devices

Prerequisite: To import third party devices from ECAD to Automation Builder, install third party fieldbus devices (for example, GSD, GSDML and EDS files) using *“Tools ➔ Device repository”* in Automation Builder.

1. From the main menu, select *“Project ➔ Import ➔ ECAD (PBF)”*.
2. From the file system, select the ECAD *pbf* file which consists of third party devices.

   ⇨ When the device identifier of the third party device installed in Automation Builder does not match with the device identifier of the device imported from ECAD, an error window is shown with the devices which are failed to import with error identifier 14.

   To import third party devices, it is required to assign ECAD identifier (PLC type designation/order number) in Automation Builder in *“Tools ➔ Device ECAD data”*. Click the link in the *Import* window to see the error messages in a text file.
3. Click *“Continue”* in the Import window to import valid devices to the project that are imported successfully or click *“Cancel”* to cancel the import process.
4. In Automation Builder, click *“Tools ➔ Device ECAD data”*.
5. In the Device ECAD data editor, add the ECAD identifier for the devices shown in the import errors window with error identifier 14, to enable these devices for export and import.



   ⇨ Also, add the ECAD identifiers for all devices which need to support export/import in ECAD.
6. Reimport the *pbf* file to import the third party devices.

1.8.1.1.4 Exporting PLC data to ECAD tool

1. Open the existing PLC project.
2. In the device tree, right-click *“PLC ➔ Export ➔ ECAD (PBF)”*.
3. Select the desired location in the file system to save the *PBF* file.

   The ECAD user can import the exported *PBF* file from Automation Builder and can use the imported PLC data for electrical engineering purpose. If the user modifies imported PLC data in the ECAD project, the data can be imported back to the Automation Builder project which supports the round trip engineering efficiently with loss less synchronization of the data.

1.8.1.1.5 Exporting third party devices

1. Right-click on a PLC device, click *“Export”* and select *“ECAD (PBF)”*.
2. Save the file to the desired location in the file system.

   If the third party devices does not contain assigned ECAD identifiers, a message is displayed showing which devices cannot be exported.

   ⇨ To add ECAD identifiers to the devices, see *Importing third party devices* *Chapter 1.8.1.1.3 “Importing third party devices” on page 4114*.

   After adding ECAD identifiers to the third party devices, execute *“Export”* to export the devices including third party devices.

1.8.1.1.6 Importing ECAD PLC data to existing AB project

Automation Builder ECAD interface supports concurrent engineering by importing the ECAD data to the existing Automation Builder PLC project.

1. From the main menu, select *“Project ➔ Import ➔ ECAD (PBF)”*.
2. Select the PBF file which has been created during the export from the ECAD tool.
3. Select the PLC from the list and click *“OK”*.

   ⇨ A dialog window is displayed if the Automation Builder project provides PLCs of the identical type as defined in the PBF file.

   By selecting *“None”* in the dialog window a new PLC is defined in the ECAD tool.
4. In the **Project Compare – Differences** view, click  to merge device signals.

   ⇨ The differences between the current PLC hardware configuration in Automation Builder and the ECAD PLC data are displayed.
5. Select the differences as desired and click *“Accept Single”* to accept the selected difference block.
6. Close the **Project Compare – Differences** view to accept the changes.

1.8.1.1.7 Arrange or map devices imported to the device pool

Devices that are imported to the device pool because of missing hierarchy information (mainly rack information) must be arranged manually in the Automation Builder project or mapped to already existing devices.

Arrange devices imported to the device pool

Arrange the unassigned devices in the DevicePool to the PLC hardware structure by drag-and-drop.

Map devices imported to the device pool

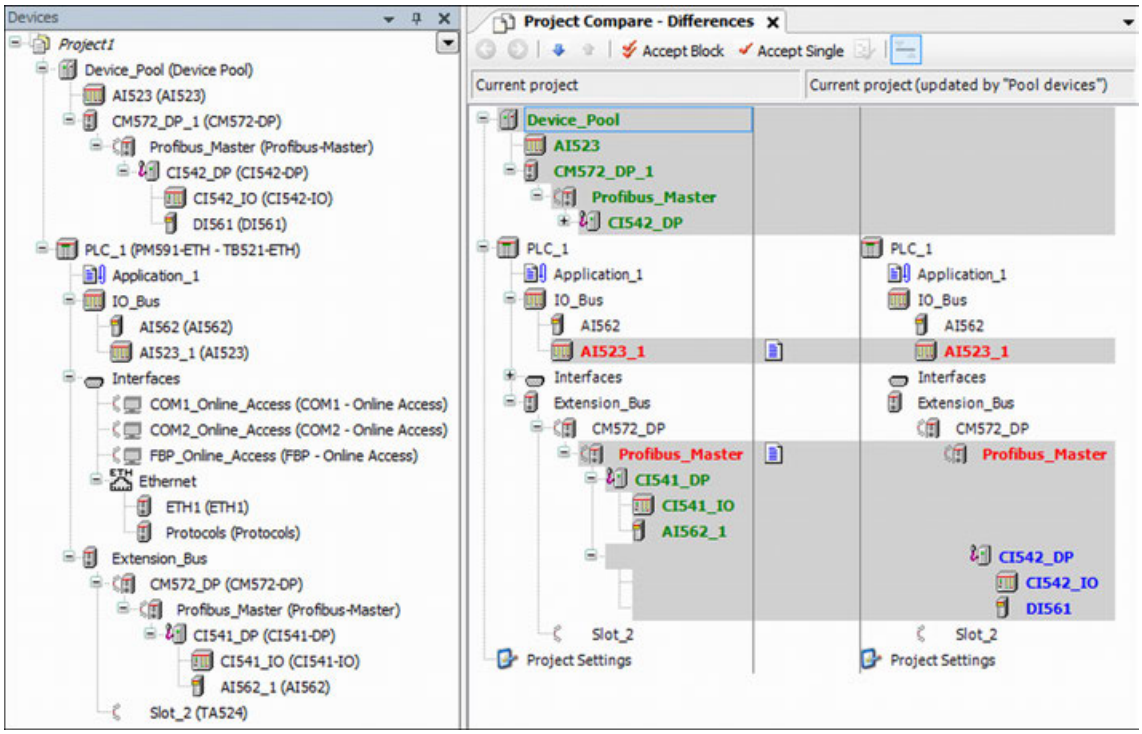
If the devices are already added to the Automation Builder project prior to the import, you have to map the instances of the same type manually (one instance in the Automation Builder project tree and one instance in the DevicePool).

After mapping the devices, you can selectively merge the device parameter or signal in the difference view.

To map pool devices, proceed as follows:

1. In the device tree, select the Device Pool node, click *“Project”* and select *“Map pool devices”*.
2. Map the device pool instances of identical types in the project from the drop-down list and click *“OK”*.

   ⇨ Pool devices which are mapped are removed from the device pool and mapped to the corresponding Automation Builder device. Differences between the signals of the mapped I/O devices are displayed. e.g. AI523_1 device:



1.8.1.1.8 Limitations

The following limitations are considered when working with the Automation Builder ECAD interface:

- The scope of a PBF file is limited to one single PLC including all connected devices.
- There is no representation of XC variants of devices in Automation Builder. Therefore, always use the standard variant for export. This might lead to part data mismatch when importing into the ECAD tool.
- In reimport or round trip import cases, if any changes are made in ECAD by adding a new communication module with connecting to one of the PLC slot or replacing existing communication module, then those device changes to the communication modules are not displayed as connected to PLC slots during the import in Automation Builder diff and import, instead those CM modules are added under the device pool. After merging and importing is completed, to work with device pool devices ↳ *Chapter 1.8.1.1.7 “Arrange or map devices imported to the device pool” on page 4115*.
- IO mapping data cannot be imported for IO devices plugged to an EtherCAT slave when they are imported individually to the device pool because of missing hierarchy information. After arranging the devices properly in the device tree, the import can be done again to import also the IO mapping data.

1.8.1.2 Exporting and importing I/O mapping (CSV)

The I/O module mappings of an Automation Builder project can be exported to CSV for bulk editing in MS Excel or other documentation purposes. I/O mappings can be exported at single I/O module level or at PLC level.

Further, the I/O module mappings can be imported with the option of displaying differences and merging each single changed or import all signals at once by overwriting existing I/O module signals.

1.8.1.2.1 Exporting IO mapping data to CSV

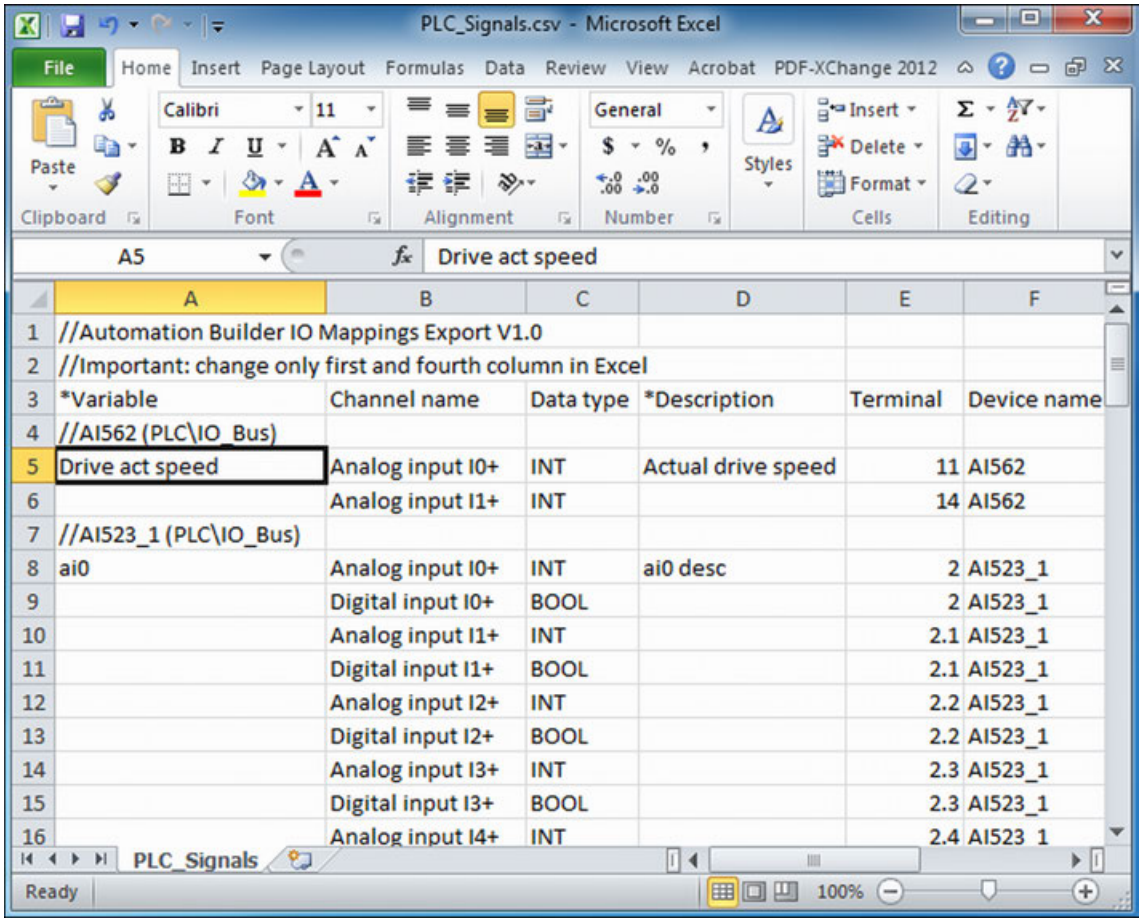
To export I/O mappings to a CSV signal list, proceed as follows:

1. In the device tree, right-click *"PLC ➔ Export ➔ IO mapping (CSV)"*.
2. Save the **IO mappings CSV** to the desired location in the file system.

   If the CSV signal list has been exported successfully, a success message is displayed. The status of the export is shown in the dialog.
3. In the export dialog, click the link to open the exported **IO mapping CSV** file in MS Excel.

> *The template can only be opened if MS Excel is installed and configured to open .csv files.*

4. In the IO mapping (CSV) file, change **Variable** and **Description** fields to edit I/O mappings.



	A	B	C	D	E	F
1	//Automation Builder IO Mappings Export V1.0					
2	//Important: change only first and fourth column in Excel					
3	*Variable	Channel name	Data type	*Description	Terminal	Device name
4	//AI562 (PLC\IO_Bus)					
5	Drive act speed	Analog input I0+	INT	Actual drive speed	11	AI562
6		Analog input I1+	INT		14	AI562
7	//AI523_1 (PLC\IO_Bus)					
8	ai0	Analog input I0+	INT	ai0 desc	2	AI523_1
9		Digital input I0+	BOOL		2	AI523_1
10		Analog input I1+	INT		2.1	AI523_1
11		Digital input I1+	BOOL		2.1	AI523_1
12		Analog input I2+	INT		2.2	AI523_1
13		Digital input I2+	BOOL		2.2	AI523_1
14		Analog input I3+	INT		2.3	AI523_1
15		Digital input I3+	BOOL		2.3	AI523_1
16		Analog input I4+	INT		2.4	AI523_1

⇨

> *Do not modify other field's data in IO mapping (CSV) file.*

1.8.1.2.2 Importing I/O mapping data from CSV

To import an edited I/O mapping (CSV) file, proceed as follows:

1. From the main menu, select *“Project ➔ Import ➔ IO mapping (CSV) ➔ Open”*.
2. A CSV signal list import dialog is displayed.
   ⇨ With *“YES”*, all I/O mappings will be imported without difference view. With *“NO”*, the difference view is displayed with the I/O mapping differences.
3. In the **Project Compare – Differences** view, click  to merge I/O mappings.
4. Select the signal row for which the difference is to be accepted. Select the **Variable** field and click *“Accept Single”* to merge the I/O mappings.
5. Close the **Project compare – Differences** view to accept the changes and merge the I/O mappings with the Automation Builder project.

1.8.1.3 Exporting and importing device list (CSV)

The Automation Builder project devices can be exported to CSV for bulk device renaming or adding device tag labels to devices in MS Excel or other documentation purposes. A devices export is only possible at PLC level.

Automation Builder provides importing devices in bulk based on device type, instance and hierarchy information provided in the CSV file.

1.8.1.3.1 Exporting device list to CSV

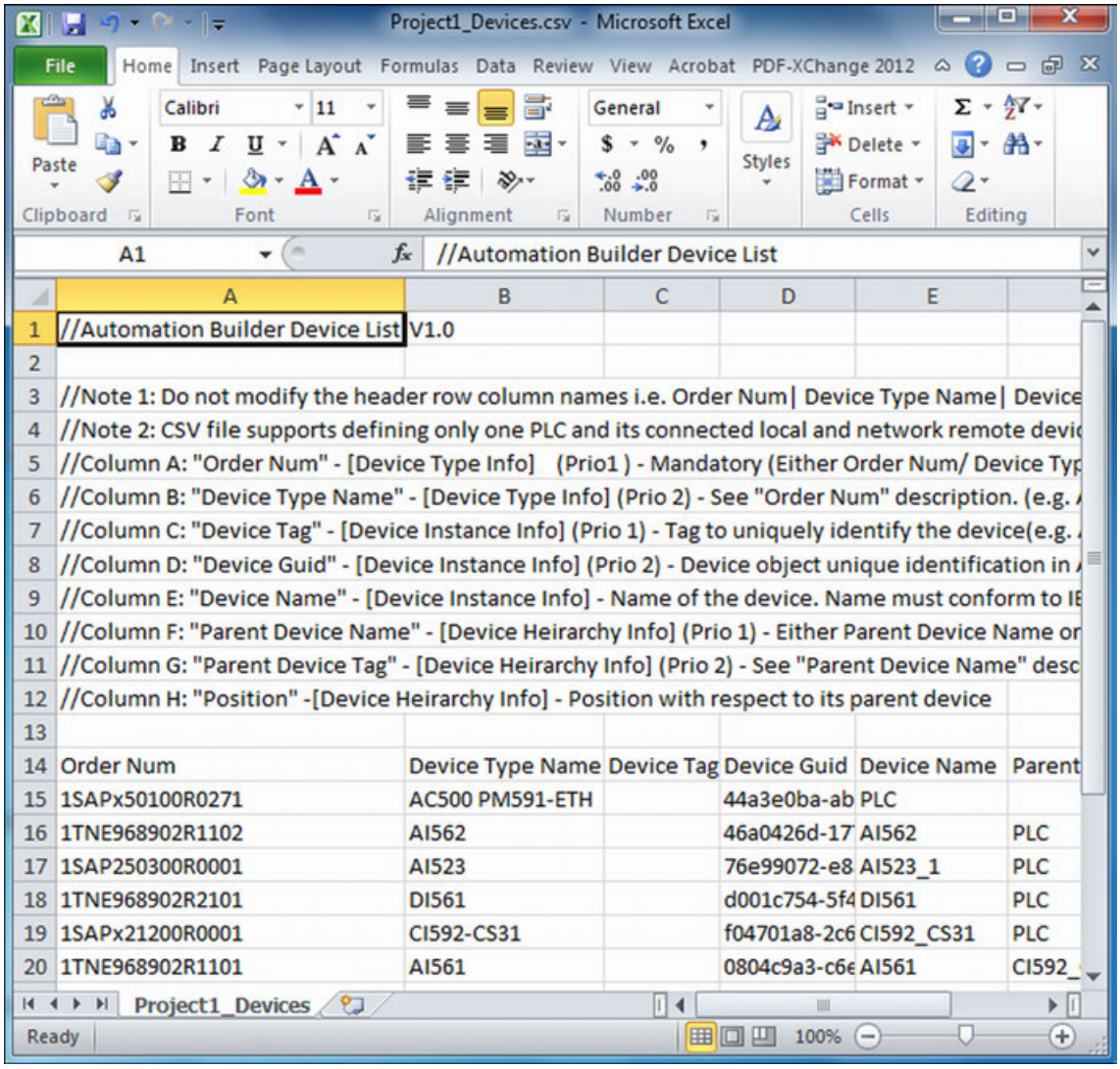
To export a CSV device list, proceed as follows:

1. In the device tree, right-click *“PLC ➔ Export ➔ Device list (CSV)”*.
2. Select the desired location in the file system to save the **Device list (CSV)**.

   If the CSV device list is exported successfully, a success message is displayed.

3. In the Export dialog, click the link to open the exported CSV device list.

   The exported CSV device list consists of all devices connected to the PLC that is exported. Each row represents a device with its device type and hierarchy information.



1.8.1.3.2 Creating CSV device list

To create the devices in CSV, use the device list template provided in Automation Builder.

▷ In the main menu, click *"Tools ➔ Create CSV Device list"*.

⇨ The device list template is opened in the MS Excel.

> *The template can only be opened if MS Excel is installed and configured to open .csv files.*

In this file, add each device in a separate row with device information like Device Type (Order Num or Device Type Name) and instance details (name, tag) and hierarchy information (parent Device name, parent Device Tag, position). The mandatory information required to import CSV is only Device Type. All other fields are optional. After editing the device list CSV file, save it in the file system and close.

1.8.1.3.3 Importing a device list from CSV

To import devices from CSV in bulk, proceed as follows:

1. From the main menu, click *“Project ➔ Import ➔ Device list (CSV)”*.
2. Select the device list CSV file from the file system and click *“Open”* in the Import dialog.

   All devices that are defined in the CSV are imported. The **Project Compare – Differences** view displays the current project and the project that has been updated by the import file.
3. Select the desired devices and click *“Accept Block”* to accept all the devices and its child device nodes or *“Accept Single”* to accept only a single device.
4. After closing the **Project Compare – Differences** view, the devices are imported to the Automation Builder project.

   ⇨ The devices (except PLC) are placed under the device pool if the valid device hierarchy information is not provided in the CSV device list file. By drag-and-drop devices can be assigned to the desired PLC hardware structure ↳ *Chapter 1.8.1.1.7 “Arrange or map devices imported to the device pool” on page 4115*.

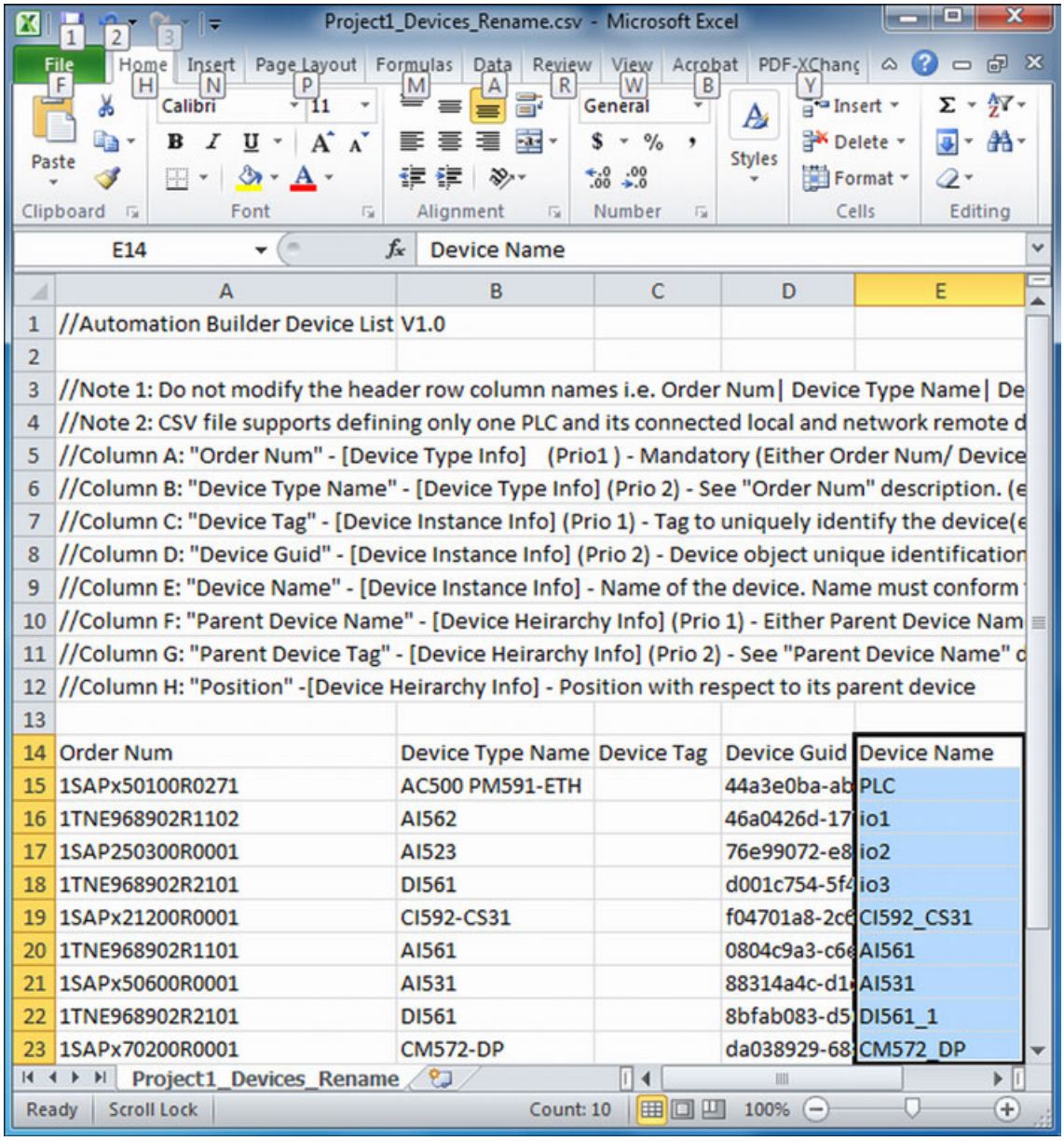
   If a device tag is provided for a device in CSV, it appears next to each device node in the device tree.

1.8.1.3.4 Renaming devices

To rename the devices, proceed as follows:

1. In the device tree, right-click *“PLC ➔ Export ➔ Device list (CSV)”*.
2. Select the desired location from the file system to save the CSV device list.

3. Rename the device names in the column **Device Name**:



4. Click *“Project ➔ Import ➔ Device list (CSV)”*.

5. Select the updated CSV file from the file system.

   Open the **Project Compare – Differences** view. If only the device names have been changed in the CSV file, the difference view does not show the changes.

> *Device Name changes are not displayed as changes in the difference view.*

6. Close the **Project Compare – Differences** view. The Renamed Devices dialog is displayed with the current name and the new name provided in the CSV file.

7. In the Rename Devices window, select the desired devices and click *“OK”*.

   The device names are updated in the Automation Builder project.

1.8.2 CODESYS Security Agent

1.8.2.1 Integration in CODESYS Development System

At this time, you can configure and create certificates of the controller with the CODESYS Security Agent. You can then configure encrypted communication with the controller, as well as encrypt the boot application, download, and online change.

See also

- *Chapter 1.8.2.2 “Encrypted Communication with Devices via Controller Certificates” on page 4122*
- *Chapter 1.8.2.3 “Encryption of the Boot Application, Download, and Online Change” on page 4123*

1.8.2.2 Encrypted Communication with Devices via Controller Certificates

Details on how to encrypt and to sign the application on the controller is described in an application note:*AC500 V3 - Encrypt and Sign your Application*

Requirement: A digital signature for certificate exchange is configured.
Chapter 1.4.1.8.17 “Encrypting an application” on page 294

> ***No certificate use in live system***
> *Self-signed certificates should **never** be used on production or public websites. The certificates that are created in the following steps are self signed.*

We assume that there is still no certificate on the controller that is intended for encrypted communication. In the following steps, you generate this kind of certificate and encrypt communication:

1. Configure the active path to the controller.
2. Open the *“Security Screen”* view by double-clicking the symbol in the status bar or by clicking *“View ➔ Security Screen”*. Select the *“Devices”* tab.
3. Click the button to refresh the list of available devices and their certificate stores.
4. Select the corresponding device on the left side.
   ⇨ On the right side, there is still no license listed for the *“Encrypted communication”* use case.
5. On the right side, select *“Encrypted Communication”* and click the button to create a new certificate on the device.

   Change the default key length to 4096. Otherwise an error occurs that is only visible in the log of the PLC.

   ⇨ The certificate is generated and listed in the table with its properties. The symbol before *“Encrypted communication”* now appears as such: . The field in the "Valid until" column is highlighted in green because the remaining time is still at least two-thirds of the entire validity period.

6. In this step, you activate encrypted communication with the controller.

   Open the *“Security Screen”* view of CODESYS (*“Users”* tab). In the *“Security Level”* group, select the *“Enforce encrypted communication”* option.

   ⇨ As of this point, communication with all controllers is possible only as long as the certificate is valid on the controller and you have a key for it.

   The connecting line between the development system, the gateway, and the controller is displayed in yellow on the *“Communication Settings”* tab of the device editor of the controller.

   As an alternative to the *“Enforce encrypted communication”* option that was just described and which applies to all controllers, you can also encrypt communication with a specific controller only. To do this, open the *“Communication”* tab in the device editor of the controller. Click *“Encrypted Communication”* in the *“Device”* list box.

7. Now log back in again to the controller.

   ⇨ A dialog opens with the notification that the certificate of the controller is not signed by a trusted source. In addition, the dialog displays information about the certificate and prompts for you to install it as a trustworthy certificate in the local store in the "Controller Certificates" folder.

8. Confirm the dialog.

   ⇨ The certificate is installed in the local store and you are logged in to the controller.

   In the future, communication with the controller will be encrypted automatically with this control certificate.

   Note: When logging in to the controller, the expiration date of the certificate currently in use is checked. You get a warning if the remaining time is just one-third of the entire time or less. Then you can renew the certificate in time in the security screen.

See also

- *Chapter 1.8.2.4.1 “View 'Security Screen' - 'Devices'” on page 4125*
- *Chapter 1.8.2.1 “Integration in CODESYS Development System” on page 4122*

1.8.2.3 Encryption of the Boot Application, Download, and Online Change

Details on how to encrypt and to sign the application on the controller is described in an application note:*AC500 V3 - Encrypt and Sign your Application*

Aim: You want to encrypt boot applications, downloads, and online changes with a certificate to make sure that the application on the controller cannot be exchanged at will. To do this, you need to download a corresponding certificate of the type "Encrypted Application" from the controller and install it to the "Windows Certificate Store" of your computer. This certificate is required for all development environments that need to make changes to the application on the controller. For example, if this application has to be downloaded from another computer, then the certificate also has to exist on this computer.

See also

- *Chapter 1.8.2.1 “Integration in CODESYS Development System” on page 4122*
- CODESYS Help: "Security", "Encryption", "Certificate"

Encrypting the boot application, download, and online change with the encryption wizard

Requirement: The active path to the controller is configured.

1. Open the *“Properties”* dialog of the application.
2. Click the *“Encryption”* tab. Set *“ Encryption Technology”* to *“Encryption with certificates”*.
   ⇨ The *“Encryption Wizard”* button is available in the *“Certificates”* field.
3. Click the *“Encryption Wizard”* button.
   ⇨ The *“Encryption Wizard”* dialog opens. The status is `Not connected` and under *“Details”* is `Ready`.
4. Click the *“Start”* button.
   ⇨ The wizard searches for suitable certificates on the controller. If necessary, the controller creates a new certificate which is registered in the Certificate Store of your computer.

   NOTE: A certificate obtained this way is automatically accepted as trusted.

   If a certificate for application encryption already exists on the controller, then it is used.

   If a new certificate has to be created on the controller for your CODESYS, then the *“Certificate Settings”* dialog opens for configuring the key length for the private key and the validity period.
5. In the *“Certificate Settings”* dialog, click *“OK”* to confirm the default or edited values for key length and validity period.
   ⇨ CODESYS saves the values in the CODESYS options as the default for the next certificate configuration of this kind.

   In the *“Details”* of the wizard, you see a description of the performed actions and the thumbprint of the recently created certificate.
6. When the status reaches *“Wizard finished”*, close the wizard.
   ⇨ The new certificate is listed in the *“Certificates”* field of the properties dialog. In the *“Certificate Store”*, it is listed under *“Controller Certificates”*. In the *“Security Screen”* view, on the *“Devices”* tab, the certificate is displayed in the right window with the *“Encrypted Application”* information.
7. Confirm the *“Properties”* dialog of the application.
8. Open the *“Security Screen”* view.
   ⇨ On the *“Project”* tab, in the *“Encryption of boot application, download and online change”* group, the certificate is displayed with the *“Encrypted Application”* information.

   Boot application, download, and online change are therefore encrypted and only possible as long as the configured certificate and signature are valid.

See also

- *Chapter 1.8.2.4.2 “Dialog 'Encryption Wizard'” on page 4128*
- CODESYS Help: Dialog "Properties" "Encryption"
- CODESYS Help: "Security", "Encryption", "Certificate"

Encrypting the boot application, download, and online change without the encryption wizard

Requirement: The active path to the controller is configured. There is still no certificate on the controller that is suitable and valid for encryption.

1. Open the *“Security Screen”* view by double-clicking the symbol in the status bar or by clicking *“View ➔ Security Screen”*. Open the *“Devices”* tab.
2. Click the *“Refresh the list of available devices and their certificate stores”* button.
3. Select the device listed on the left side.

4. Select *"Encrypted Application"* on the right side and click the *"Create a new certificate on the device"* button.

   Change the default key length to 4096. Otherwise an error occurs that is only visible in the log of the PLC.

   ⇨ The certificate is created and listed in the table with the symbol.

5. Double-click the certificate entry.

   ⇨ The Windows *"Certificate"* default dialog opens.

6. Click the *"Install certificate"* button on the *"General"* tab.

   ⇨ The *"Certificate Import Wizard"* opens.

7. In the *"Certificate Store"* dialog, select the *"Place all certificates in the following store"* option and select the *"Controller Certificates"* folder for *"Certificate Store"*.

   ⇨ The controller certificate is imported to the *"Controller Certificates"* directory and it is immediately available for the encryption of downloads, online changes, and boot applications.

8. Open the *"Project"* tab and double-click the application entry in the *"Encryption of boot application, download and online change"* group.

   ⇨ The *"Properties"* dialog of the application opens.

9. Click the *"Encryption"* tab and set *"Encryption Technology"* to *"Encryption with certificates"*. Then click . Note: If the *"Enforce encryption of downloads, online changes and boot applications"* option is selected in the *"Security Screen"*, then *"Encryption with certificates"* is already preset.

10. In the *"Certificate Selection"* dialog, select the corresponding certificate from the *"Controller Certificates"* folder and click .

11. Click *"OK"* to confirm the dialog.

    ⇨ The certificate is displayed in the properties dialog.

12. As above when using the wizard, steps 7 and 8.

Enforcing the encryption of boot applications, downloads, and online changes

▷ Open the *"Users"* tab in the *"Security Screen"*. In the *"Security level"* group, select the *"Enforce encryption of downloads, online changes and boot applications"* option.

⇨ Only with a valid certificate is it possible to change the application on the controller.

See also

- CODESYS Help: "Security-Screen"

1.8.2.4 Reference, User Interface

1.8.2.4.1 View 'Security Screen' - 'Devices'

Symbol:

Function: The tab allows for the configuration and the transfer of controller certificates for encrypted communication with the controller.

Call: Menu bar: *“View”*

The *“Devices”* tab shows all PLC devices configured in the project and their certificate store. If the communication path to the controller is configured, then you see the certificates that are stored in memory. Here you can create and configure new certificates on the controller. If a certificate currently in use is about to expire, then you get a warning when you log in to the device. From there you can also switch directly to the *“Security Screen”* to renew the certificate.

Left side: *“Information”*	Devices and certificate store Shows the individual devices as expandable nodes, each with the controller-specific certificate store below it.
Toolbar (left side)	: Refresh the display : Download: Transfer the selected certificate to the PLC

Right side: *“Information”*	If the active path to the controller is set and a device node is selected, then every use case for controller certificates is displayed on the right side. ● *“OPC UA Server”*: Encrypted communication over an OPC UA server ● *“Encrypted Communication”*: Encrypted communication between the development system and the controller ● *“Encrypted Communication”*: Encryption of the boot application ● *“Web server”*: Encrypted communication with the web server As long as a certificate is not available for one of these use cases, it is displayed with the  symbol as *“(not available)”*. When a certificate store is selected on the left side, all certificates in it are displayed on the right side with the following information: *“Information”*: Use case (currently the controller component in question is displayed: for example *“CmpSecureChannel”*.) *“Created for”*: Name of the computer for which the certificate was created (for example, *“MyLocalPC”*) *“Created by”*: Name of the computer on which the certificate was created (for example, *“MyLocalPC”*) *“Valid as of”*: Date (for example, *“07/20/2017 15:09:29”*) *“Valid until”*: Date (example: *“07/20/2022 00:00:00”*. Depending on the remaining time of the certificate, the highlight color of the field changes: green -> yellow (two-thirds expired) -> orange (nine-tenths expired) -> red (expired). Note: When logging in to the controller, you get a warning when two-thirds or more of the validity period have expired. Then you can renew the certificate here in the *“Security Screen”*. *“Thumbprint”*: Hash value from specific properties of the certificate for purposes of identification (for example, *“279e1a46b86bd636c8e6f19fd51c222469ec49a8”*) This thumbprint can be used together with the Mqtt library. Refer to the Mqtt library documentation in the Library Manager. Double-clicking a certificate entry opens the default Windows *“Certificate”* dialog. As a result, you can import a controller certificate into the Windows Certificate Store in the *“Controller Certificates”* folder, so that it is available for the encryption of boot applications, downloads, and online changes. If multiple certificates are available for one use case, then the system follows the steps below to determine the certificate that is used: ● Certificate that was created directly by the user (currently not supported) ● Filtering of existing certificates by: – 1. Subject (user of the certificate) – 2. Key usage – 3. Extended key usage – 4. Valid time stamp ● Dividing of detected, valid certificates as "signed" and "self-signed" ● Filtering of signed certificates, and the self-signed certificates by the following criteria: – 1. Longest validity period – 2. Strongest key

	Drag&Drop: Moving of the certificate to another certificate store of the same device Double-clicking a certificate entry opens the default Windows dialog for displaying all certificate information.
Toolbar (right side)	: Creates a new certificate for a specific use case The *“Certificate Settings”* dialog opens for configuring the *“Validity period”* of the certificate and the *“Key length”* for the private key. Clicking *“OK”* saves the specified values in the CODESYS options. The values are reset at the next operation. As long as the certificate is being created, "*“(computing)”*" is shown after the use case. You cannot cancel the creation operation, but you can close and continue working with the *“Security Screen”*. : Delete the selected certificate. : Upload and save the selected certificate to the local file system. : Details about the selected certificate: Opens the *“Certificate”* dialog with the *“General”* tab, *“Details”* tab, and *“Certification Path”* tab. : Renew the selected certificate. Opens the *“Certificate Settings”* dialog to create an additional new certificate for a certificate that will expire soon, with the same purpose and specified key length. The predefined values in the dialog are adapted, if necessary, depending on the selected certificate.

1.8.2.4.2 Dialog 'Encryption Wizard'

Function: The wizard makes sure that a certificate for the encryption of downloads, online changes, and boot applications is downloaded from the controller. If a valid certificate does not exist on the controller for this purpose, then the wizard makes sure that a certificate is created. Changes to the application on the controller (download, online change, boot application) are possible only when this certificate exists.

Call: *“Properties”* dialog of an application, *“Encryption”* tab, *“Encryption with certificates”* setting, *“Encryption Wizard”* button

Requirement: *“Encryption Technology”* is set to *“Encryption with certificates”*.

“Status”	Statuses while the wizard:is in action: ● *“Not connected”*: The connection to the controller has not been established yet or the device cannot be reached. ● *“Error connecting to the device”*: The network path to the controller has not been set correctly. ● *“Connecting...”*: A connection to the controller is being established. ● *“Processing request...”*: The wizard is checking for available certificates and if necessary makes sure that the controller creates a new certificate. The certificate downloaded from the controller is automatically classified as "trusted" and registered in the Certificate Store of the computer. ● *“Wizard finished”*
“Details”	Description of the individual actions of the wizard with corresponding notices in the case of failures
“Start”	If the connection path to the controller is set correctly in the device editor, then the wizard starts the necessary actions for encrypting downloads, online changes, and boot applications with a certificate. If an expired certificate exists, then a corresponding warning is displayed with a dialog prompt whether or not a new certificate should be created by the controller. When this is confirmed, the new certificate is created and loaded to the local Certificate Store. In this case, the existing boot application may not start anymore and must be created again with the new certificate. The *“Certificate Settings”* dialog opens when a new certificate is to be created on the device. Here you configure the *“Key length (bit)”* and the *“Validity period (days)”* for the certificate.

See also

● *Chapter 1.8.2.3 “Encryption of the Boot Application, Download, and Online Change” on page 4123*
● CODESYS Help: "Security", "Encryption", "Certificate"

1.8.3 CODESYS Static Analysis

Already when programming in CODESYS, CODESYS Static Analysis helps to write more readable code and to detect contradictory or unsupported settings. In particular, potential sources of error can be identified, such as test code or pointers that have not been checked for 0 before dereferencing. With specific checks, you can make sure that the code is portable. Example: The analysis should report the use of language resources for object orientation because the code is to run on platforms that do not support object orientation.

The analysis checks the source code of the CODESYS project and reports any deviations from certain coding rules, naming conventions, or permitted keywords and identifiers. CODESYS Static Analysis is based on the rule set defined in the PLCopen Coding Guidelines and extends it with additional test options.

You can display the detected deviations as errors or warnings in the message view before the project is downloaded to the target system. For errors that are reported by Static Analysis based on precompile information, there is support for an immediate error handling ("Quickfix").

You activate Static Analysis either explicitly by clicking *“Build ➔ Run Static Analysis”*, or you let it execute automatically at each code generation. You activate the automatic execution in the *“Static Analysis”* dialog of the projects settings. In this dialog, you also configure what is to be checked in detail. You can use pragma statements to exclude individual parts of the code from the check.

To evaluate the code quality, you can also display selected metrics that CODESYS Static Analysis detects in your code in a separate view. An example of this is the McCabe metric, which measures the cyclomatic complexity and indicates the number of execution paths that can be processed during code execution.

NOTICE!
The analysis is performed only for the code of the applications in the current project. Libraries are not taken into consideration.

The CODESYS development system contains a light version of Static Analysis that is extended by CODESYS Static Analysis.

See also

- *Chapter 1.8.3.2.2.1 “Dialog 'Static Analysis Settings' - 'Settings'” on page 4138*
Chapter 1.8.3.2.1 “Commands” on page 4133
- *Chapter 1.8.3.3.1 “Pragmas and Attributes” on page 4149*
- Usage and benefits of code optimizations are described in the *application example*.

1.8.3.1 Configuring and Running Static Analysis

Using a basic sample project below, you will find the most important steps and options for configuring and running a static analysis.

Requirements: CODESYS Static Analysis is installed.

Sample project If you want to reproduce the example project, create a standard project and insert the POUs below the application in the device tree. Then configure the communication settings for the connection to your local CODESYS Control Win V3.

```
FUNCTION_BLOCK fb1
VAR_INPUT
       iVar_fb1in1 : INT;
       ivar_fb1in2 : INT;
       rVar_fb1in3 : REAL;
END_VAR
VAR_OUTPUT
       iVar_fb1out:INT;
END_VAR
VAR
       P_fSampleProperty : INT;
    rVar : REAL;
       PRO : BOOL;
END_VAR
iVar_fb1out:=iVar_fb1in1 + 1;

FUNCTION_BLOCK fb2
VAR_INPUT
       iVar_fb2in:INT;
END_VAR
VAR_OUTPUT
       iVar_fb2out:INT;
END_VAR
VAR
END_VAR
```

```
PROGRAM PLC_PRG
VAR
    fb1_inst: fb1;
    fb2_inst: fb2;
END_VAR
fb1_inst(iVar_fb1in1 := 99);
fb2_inst(iVar_fb2in := 22);
fb2_inst(iVar_fb2in := 1);
```

Checking for compliance to rules

1. Click *"Build ➔ Static Analysis ➔ Settings"*. Switch to the *"Rules"* tab.

   ⇨ A list is displayed containing all possible rule checks. They are organized in a tree structure by topical category. The rule number is added in parentheses (for example, *"Unused variables (33)"* in category *"Unused objects"*).

2. Click the check box of the first line a few times (*"Rules"* node).

   ⇨ Clicking toggles the activation status. The check boxes in the entire tree have a red or orange check mark, or no check mark at all.

3. In this way, activate all entries with a red check mark. This means that CODESYS Static Analysis should report any detected rule violations as errors.

4. Click *"Build ➔ Static Analysis ➔ Run Static Analysis"*.

   ⇨ Errors are reported in the message view. The message texts are tagged with a § and begin with the error number "SA<rule number>".

5. Double-click the first message `SA0033: Unused variables 'iVar_fb2out'.`

   ⇨ The focus moves to the declaration part of function block `fb2` and the relevant variable is selected. The variable is declared, but not used. This is checked in Rule 33 (*"Unused variables"*). In the code, the relevant locations are underlined with a wavy line.

6. To test the automatic execution of the analysis, click *"Build ➔ Static Analysis ➔ Settings"*. On the *"Settings"* tab, select the *"Perform static analysis automatically"* option. Click *"OK"* to exit the dialog.

7. Click *"Online ➔ Login"*.

   ⇨ A dialog prompt indicates that compile errors exist. The errors reported by the code analysis are displayed again in the message view.

8. Click *"Build ➔ Static Analysis ➔ Settings"*. Switch to the *"Rules"* tab.Now clear all of the rules in the dialog. In the *"Unused Objects"* category, explicitly activate Rule SA0035 (*"Unused input variables (35)"*) with an orange-colored check mark to report a warning. See the tooltip for the rule text: *"This rule corresponds to the following PLCopen rules: CP24"*). Click *"OK"* to exit the dialog.

   In the project settings, click *"OK"*.

9. Click *"Build ➔ Generate Code"*.

   ⇨ The analysis is performed automatically. Two errors are reported in the message view: `§ SA0035: Unused input variable 'iVar_fb1in2` and `§ SA0035: Unused input variable 'iVar_fb1in3`.

10. Double-click the message and comment or remove the declaration. Perform the code analysis again.

    ⇨ No error messages are displayed.

See also

- *Chapter 1.8.3.2.2.1 “Dialog 'Static Analysis Settings' - 'Settings'” on page 4138*
- *Chapter 1.8.3.2.2.2 “Dialog 'Static Analysis Settings' - 'Rules'” on page 4139*

Checking for compliance to defined naming conventions

1. Click *“Build ➔ Static Analysis ➔ Settings”*. Click the *“Naming Conventions”* tab.
   ⇨ You see a table in a tree structure that is divided into expandable categories of variables and program blocks.
2. Expand the *“Prefixes for Variables” - “Prefixes for Types”* category, and in the *“Prefix”* column, specify `I` for *“INT (14)”*.

   Expand the *“Prefixes for POUs” - “Prefixes for POU Type”* category: In the *“Prefix”*column, specify the `prog` for *“PROGRAM (122)”* and `fb` for *“FUNCTIONBLOCK (103)”*.
3. Select the *“First character after prefix should be an upper case letter”* option. Clear all other options.
4. Click *“Build ➔ Static Analysis ➔ Run Static Analysis”*.
   ⇨ Error messages:
   - `NC0102: Invalid name 'PLC_PRG': Expect prefix 'prog'` because PLC_PRG does not have the required prefix
   - `First character after prefix should be uppercase: 'ivar_fb1in2'` because `ivar_fb1in2 : INT;` in `fb1`
   - `NC0014: Invalid variable name P_fSampleProperty: Expect prefix 'i'` because this integer variable does not have the required prefix

See also

- *Chapter 1.8.3.2.2.3 “Dialog 'Static Analysis Settings' - 'Naming Conventions'” on page 4140*

Checking for forbidden symbols

1. Click *“Build ➔ Static Analysis ➔ Settings”*. Click the *“Forbidden Symbols”* tab.
   ⇨ A line editor allows for specifying character strings that should not to be used in the code.
2. As an example, double-click the blank line and type in the invalid character string `PRO` directly. Double-click the next blank line and click … to open the input assistance. From *“Standard Types”*, select *“REAL”*. Click *“OK”* to exit the dialog.
3. Click *“Build ➔ Static Analysis ➔ Run Static Analysis”*.
   ⇨ The error messages `Forbidden symbol 'REAL'` and `Forbidden symbol 'PRO'` are displayed in the message view. Double-click the message text to jump to the relevant line of code.

See also

- *Chapter 1.8.3.2.2.5 “Dialog 'Static Analysis Settings' - 'Forbidden Symbols'” on page 4148*

Displaying of metrics

CODESYS Static Analysis performs selected tests on the code, and you can display the results in a view.

1. Click *"Build ➔ Static Analysis ➔ Settings"*. Click the *"Metrics"* tab.
   ⇨ The metrics that CODESYS Static Analysis applies to the code are listed in a table.
2. For this example, activate the *"Number of inputs variables"* metric and specify the permitted range of values: lower limit `1` and upper limit `2`. Activate some more metrics, for example *"Code size"* and *"Number of calls"*.
3. Click *"Build ➔ Static Analysis ➔ View Standard Metrics"*.
   ⇨ The view includes a table with a line for each *"Program unit"* of the sample program. For each activated metric, there is a column showing the measured values. Values that are outside of the range of values defined in the settings are highlighted in red. In the case of this specific example, this is at least the *"PLC_PRG/Inputs"* field because the number of input variables in this POU is greater than the defined upper limit of `2`.

See also

- ↳ *Chapter 1.8.3.2.2.4 "Dialog 'Static Analysis Settings' - 'Metrics'" on page 4147*

See also

- ↳ *Chapter 1.8.3.2.1 "Commands" on page 4133*

1.8.3.2 Reference, User Interface

1.8.3.2.1 Commands

Command 'Settings'

Function: The command opens the *"Static Analysis Settings"* dialog.

Call: Menu bar: *"Build ➔ Static Analysis"*

Requirement:

- The CODESYS Static Analysis package is installed.
- A project is open.

See also

- ↳ *Chapter 1.8.3.2.2.1 "Dialog 'Static Analysis Settings' - 'Settings'" on page 4138*

Command 'Run Static Analysis'

Symbol: §

Function: The command starts the static analysis for the active application and displays the results in the message view.

Call: Menu bar: *"Build ➔ Static Analysis"*

During the code analysis, CODESYS generates code just like the *"Build ➔ Generate Code"* command. The results of the analysis are displayed as errors § and warnings § in the message view (*"Build"* category). The numbers refer to the corresponding rules as they are defined in the project settings. The syntax for the displayed messages is *"SA<rule number:<rule text>"*.

See also

- *Chapter 1.8.3.3.2 "Rules" on page 4154*

Command 'View Standard-Metrics'

Symbol: 

Function: The command starts the static analysis for the active application and displays the metrics for all POUs in a table.

Call: Menu bar: *"Build ➔ View Standard Metrics"*

The metrics (code numbers) to be displayed are activated in the project settings. You can access the configuration by clicking *"Configure"* in the context menu of the displayed table. If a value is outside of the configured upper and lower limits, then the field in the table is highlighted in red.

See also

- *Chapter 1.8.3.2.2.4 "Dialog 'Static Analysis Settings' - 'Metrics'" on page 4147*

Standard Metrics

Metric	Description
"Code size"	Number of bytes
"Variable size"	Number of bytes
"Stack size"	Number of bytes
"Calls"	Number of calls
"Tasks"	Number of calls from tasks
"Global"	Number of different global variables
"I/Os"	Number of direct object accesses
"Local"	Number of local variables
"Inputs"	Number of input variables
"Outputs"	Number of output variables
"NOS"	Number of statements
"Comments"	Percentage of comments
"McGabe"	McGabe complexity
"Prather"	Prather complexity of nesting
"DIT"	Depth of inheritance tree
"NOC"	Number of children
"RFC"	Response for class
"Elshof"	Elshof complexity of reference
"CBO"	Coupling between objects
"LCOM"	Lack of cohesion in methods
"n1 (Halstead)"	Number of different used Halstead (n1) operators

Metric	Description
“N1 (Halstead)”	Number of Halstead (N1) operators
“n2 (Halstead)”	Number of different used Halstead (n2) operands
“N2 (Halstead)”	Number of operands (N2)
“HL (Halstead)”	Halstead length (HL)
“HV (Halstead)”	Halstead volume (HV)
“D (Halstead)”	Halstead difficulty (D)
“ SFC branches”	Number of SFC branches
“SFC steps”	Number of SFC steps

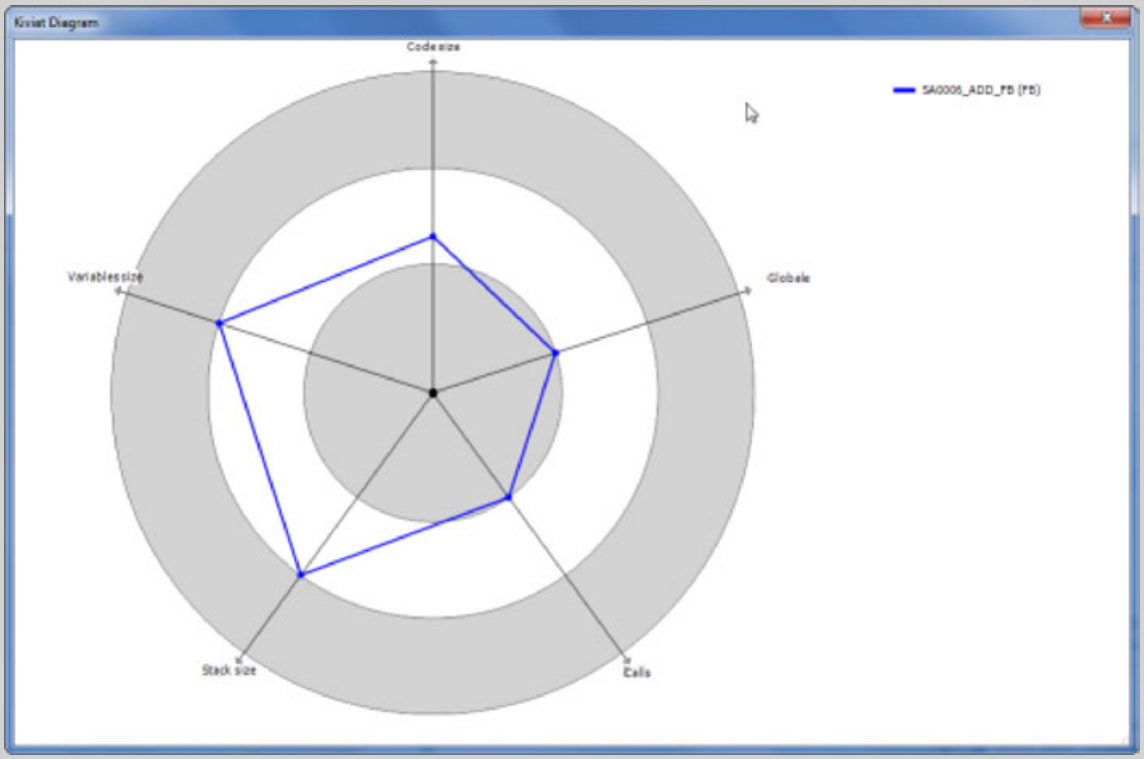
Commands in the context menu

The following commands are provided in the context menu of the table:

- *“Calculate”*: The values are refreshed.
- *“Copy Table”*: The table is copied to the clipboard. The separators are tabs.
- *“Print Table”*: The default dialog for setting up a print job opens.
- *“Export Table”*: The table is exported as a CSV file. The separators are semicolons.
- *“Kiviat Diagram”*: Requirement: At least three metrics have defined upper and lower limits.
  A radar chart is created for the selected POU. This visualizes the quality of POU code with respect to a given standard.
  Each metric is depicted as an axis with its origin at the center (value 0) which radiates outward into three concentric ring zones. The inner ring zone represents the range of values below the lower limit defined for the metric. The outer ring represents the range of values above the upper limit. The axes of the metrics are distributed uniformly around the circle.
  The current values of the individual metrics on the axes are connected by a line. In the ideal case, the complete line is located in the middle zone.
- *“Configure”*: The table for selecting the desired metrics opens. This corresponds to the table in the project settings.
- *“Open POU”*: The POU opens in the editor.

Example of a Kiviat diagram for five metrics

The name of the metric is displayed at the end of the respective axis and the name of the POU is displayed in the upper right corner of the diagram.



See also

- ↳ *Chapter 1.8.3.2.2.4 “Dialog 'Static Analysis Settings' - 'Metrics'” on page 4147*

Command 'Extract function'

Function: The command opens the *“Extract Function Configuration”* dialog.

The command extracts selected code from the ST editor and creates a new method or function containing this code. The affected code in the ST editor is replaced by a correct call. When code is extracted from a function block or the child of a function block, a new method is created from the code. When code is extracted from a program or a function, a new function is created from the code.

Call: Context menu: *“Refactoring”*

Requirements: When the selected code consists of one or more statements:

- The selected code does not contain any compile errors.
- The selected code is located in the implementation part of an ST POU.
- The selected code does not contain any exiting jumps
  Examples of exiting jumps include the following:
  – Using `RETURN` to exit the enclosing function
  – Using `CONTINUE` or `EXIT` to exit a loop enclosing the code

You can undo the changes that the “Extract function” command made in your project by positioning the cursor in the device tree and clicking “Edit ➔ Undo”.

Table 757: Dialog “Extract Function Configuration”

“Name”	Name of the recently created function or method The default name can be changed.
“Return value”	Determines the return value of a function if there are multiple output and/or input/output parameters
“Parameter”	Display of the available POUs Configuration whether the parameters are used as input, output, or input/output variables : Input variables : Output variables : Input/Output variables
	The changes made for *“Name”*, *“Return value”*, or *“Parameter”* are undone.
Upper code window	Recently created code of the call location
Lower code window	Recently created code of the function or method
“OK”	The displayed code changes are accepted in the ST POUs and the dialog is closed.
“Cancel”	The displayed code changes are rejected and the dialog is closed.

Command 'Detect clones'

Function: The command scans the program code of the open CODESYS project for copied code, and opens the *“Clone detection results”* view to display the detected cloned code blocks. In the process, only code blocks larger than a specific size are considered to be clones. Very small chunks of code are not displayed as clones.

Call:

- Menu bar: *“Build ➔ Static Analysis”*
- Context menu: *“Static Analysis”*

Requirement: The CODESYS project is open.

Two code positions are considered clones if they have the following properties:

- Same structural composition
- Variables have the same data type.
- Variable names may be different (exception: component access). However, an identifier that is contained multiple times in the code has to be in the same place in both code positions.
- Literals have the same data type.
- Literals may be different. A literal that occurs multiple times in the code has to occur at the same place in both code positions.

Table 758: View “Clone detection results”

“Summary”	Tab to display the search results ● *“Number of found cloned code sequences”* ● *“Number of statements compared”* ● *“Number of statements in cloned code”* ● *“Clone ratio”*: Specified as a percentage: *“Number of statements in cloned code”* / *“Number of statements compared”*

"Results"	The tab displays the code clones in a tree view and provides commands and filter options. The first occurrence of a duplicate from the set of duplicates is taken as the root node. The background color of the child nodes indicates whether the code is different or completely identical. Same colors mean the "same code". The contents of the tree view are sorted in descending order by the number of statements of the duplicated code.
Commands and filters on the *"Results"* tab	
"Subnodes/Clone"	Number of subnodes (statements) in the code block If the number of subnodes is less than 20, then the code clone is not considered.
"Filter on Object"	Input field for an *"Object"*, by which the clone list is filtered
"Show selected clones"	Requirement: Two child nodes of the same parent node are selected. Both programming objects are displayed in the upper part of the view for comparison. In the process, the code duplicates are highlighted and differences (for example, different variable names) are highlighted in a different color.
List of code clones	Columns ● *"Description"* ● *"Subnodes/Clone"* ● *"Object"* ● *"Position"* Double-clicking a child node opens the corresponding programming object, and the duplicated code block is selected there.

1.8.3.2.2 Dialogs

For the dialogs for the configuration of static code analysis, click *"Build ➔ Static Analysis ➔ Settings"*. Requirement: A CODESYS project is open.

Dialog 'Static Analysis Settings' - 'Settings'

Function: In the dialog, you select automatic static analysis, and save or load the project settings for static analysis as a CSA file.

Call:

- Menu bar: *"Project ➔ Project Settings"*, *"Static Analysis"* category, *"Open configuration dialog"* link
- Menu bar: *"Build ➔ Static analysis ➔ Settings"*

Requirement:

- The CODESYS Static Analysis package is installed.
- A project is open.

"Perform static analysis automatically"	☑: CODESYS performs the code check automatically at each code generation (for example, when the *"Build ➔ Generate Code"* command is executed or before a download. ☐: The code check is not performed automatically, but it can be performed explicitly by means of the *"Build ➔ Static Analysis ➔ Run Static Analysis"* command.
"Load"	Opens the *"Load Static Analysis Configuration"* dialog for selecting the project settings for the static analysis as a CSA file in the file system. When you click the *"Open"* button, the selected CSA file is loaded.
"Save"	Opens the *"Save Static Analysis Configuration"* dialog for saving all project settings in the *"Static Analysis"* category as a CSA file in the file system.

See also

- *Chapter 1.8.3.1 "Configuring and Running Static Analysis" on page 4130*

Dialog 'Static Analysis Settings' - 'Rules'

Function: In the dialog, you select the rules that are checked during the static analysis of the source code of a project.

Call:

- Menu bar: *"Project ➔ Project Settings"*, *"Static Analysis"* category, *"Open configuration dialog"* link
- Menu bar: *"Build ➔ Static analysis ➔ Settings"*

Requirement:

- The CODESYS Static Analysis package is installed.
- A project is open.

This tab shows a tree structure of all rules that can be checked during static analysis. By default, every rule is activated, with the exception of SA0016, SA0024, SA0073, SA0101, SA0105, SA0106, SA0133, SA0134, SA0150, SA0162, and all strict IEC rules.

Each rule has a unique number. When the rule is checked and a violation is detected, the rule number and an error description are shown in the message view in the *"Build"* category in the following format: **SA<rule number>**, where SA stands for "Static Analysis" (example: "SA003" for rule 3).

The list of available rules can be extended by specific plug-ins.

Some rules that are activated in the dialog can be deactivated temporarily in the application by applying a pragma.	
When you click the check box, the setting toggles between ☐, ☑, and ☑. When you activate or deactivate a parent node, all child rules are also activated or deactivated, respectively.	
"Filter":	• Input field for the strings to be searched for • : Rules are grouped by category. – *"Structured by Importance"*: Sorting by *"Importance High"*, *"Importance Medium"*, and *"Importance Low"* – *"Default"*: Default structuring of the rules in CODESYS Static Analysis • : Rules are displayed as a flat list. By clicking on the corresponding column header, the list can be sorted by rule number, activation/deactivation, rule-specific configuration, or importance.

Columns	
"Rules"	List of rules with rule number
Rule check	● ☐: The rule is not checked. ● ☑: If the result of the check is positive, then an error (§) for the static analysis is displayed in the message view. ● ☑: If the result of the check is positive, then a warning (§) for the static analysis is displayed in the message view.
"Precompile"	Rules which can be checked during precompile are identified by a check mark ✓ in this column. An immediate bugfix (Quickfix) is possible for these rules. You can execute an automatic, immediate error handling directly at the affected code positions.
"Rule specific configuration"	For some rules, you can double-click the field to open a rule-specific dialog to configure the rule.
"Importance":	Importance of the rule: ● 3 red stars: High ● 2 orange stars: Medium ● 1 gray star: Low

See also

- *Chapter 1.8.3.3.2 "Rules" on page 4154*
- *Chapter 1.8.3.3.1 "Pragmas and Attributes" on page 4149*
- *"Checking for compliance to rules" on page 4131*

Dialog 'Static Analysis Settings' - 'Naming Conventions'

Function: In the dialog, you define the prefixes for the data types and scopes of variables, as well as prefixes for POUs and user-defined data types (DUTs). Static analysis checks compliance with the naming conventions. When a convention is not observed, the static analysis reports an error message in the *"Messages"* view.

Call:

- Menu bar: *"Project* ➔ *Project Settings"*, *"Static Analysis"* category, *"Open configuration dialog"* link
- Menu bar: *"Build* ➔ *Static analysis* ➔ *Settings"*

Requirement:

- The CODESYS Static Analysis package is installed.
- A project is open.

The error messages are displayed in the following format: § **`NC <prefix convention number> :`** `<message text>`. NC stands for "Naming Convention". For example, the error message "§*"NC0102: Invalid name..."*" means a violation of naming convention 102 for POUs of type `PROGRAM`.

> *You can use the pragma `'naming'` to deactivate naming conventions for individual identifiers. The identifiers can begin with anything, not necessarily with the prefix.*

"Filter"	Input field for strings to be searched for

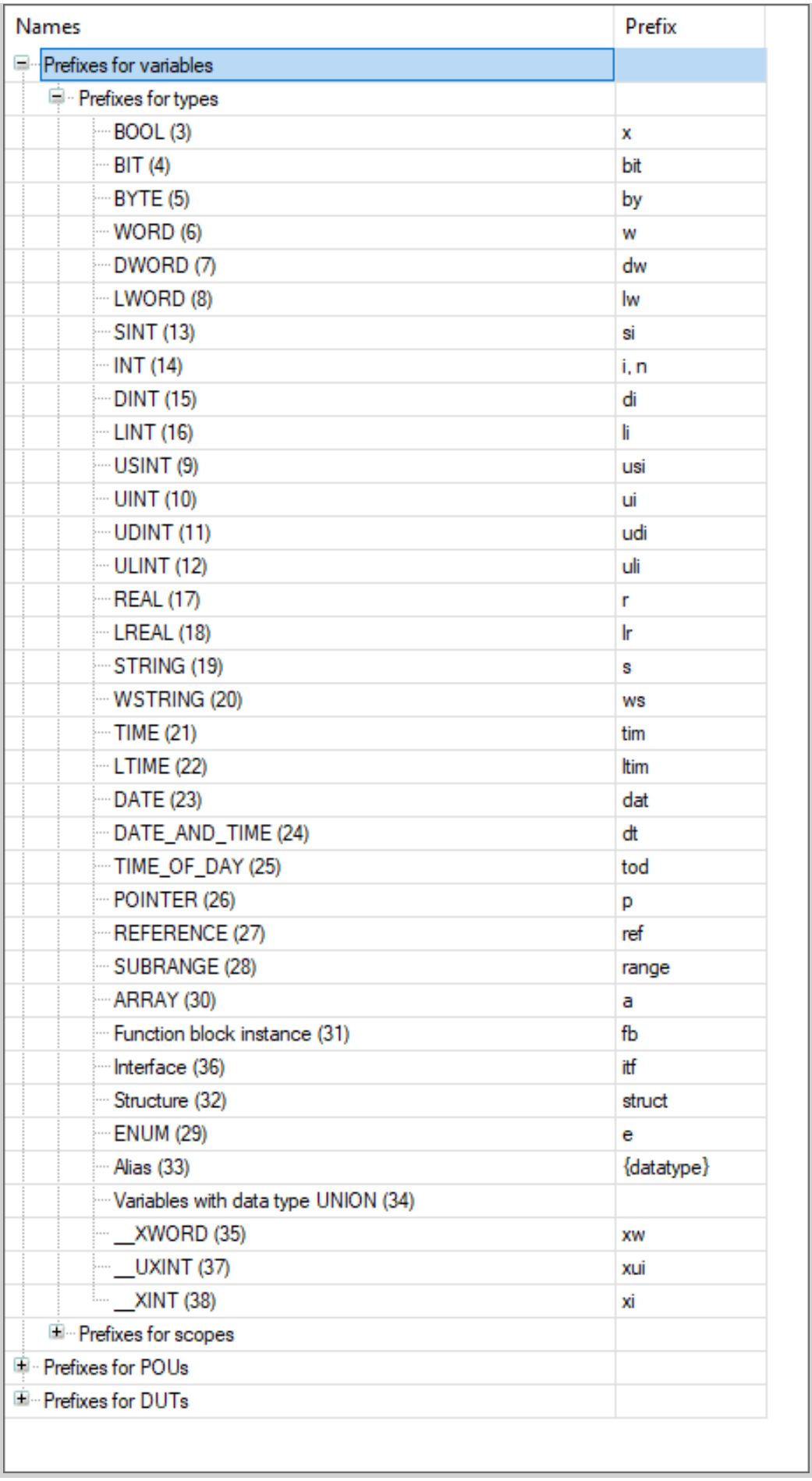
Table with the naming conventions	
“Names”	Nodes and elements for which a prefix can be defined. The number in parentheses after each element (for example, *“PROGRAM (102)”*) is the prefix convention number that is reported in the case of noncompliance with a naming convention.
“Prefix”	Input field of the prefix ● Multiple prefixes can be specified by means of comma separation. Example: *“Prefix for POUs”*, `PROGRAM (102):prog, PRG_` *“Prefix for POUs”*, `FUNCTION (103):fun, FUN_` ● Regular expressions (RegEx) are also possible for prefixes. To do this, an `@` has to be prepended. Example: The name has to begin with `x` and may contain one character from the scope `a-dA-D`: `@x[a-dA-D]`. ● For variables of type *“Alias”* and POUs of type *“Property”*, the prefix can be defined with the placeholder `{datatype}`.
“Prefixes for variables”	Organizational node for all variables for which a prefix can be defined dependent on data type or scope.
“Prefixes for POUs”	Organizational node for all POU types and method scopes for which a prefix can be defined
“Prefixes for DUTs”	Organizational node for the DUT data types (structure, enumeration, alias, or union) for which a prefix can be defined
“Prefixes for custom types”	Organizational node for special custom types (particularly those from libraries) You can extend the list with conventions: Click the blank space below it. In the *“Input Assistant”* dialog, specify the name of a custom type or select a custom type. To delete a convention, select it and press the *[Del]* key. Note: These conventions have priority over the prefixes which are defined with the attribute `{attribute 'nameprefix' := '<prefix>'}`.

Options	
“First character after prefix should be an upper case letter”	☑: Static analysis reports an error for a variable when the first character of the variable name after the defined prefix is not an uppercase letter.

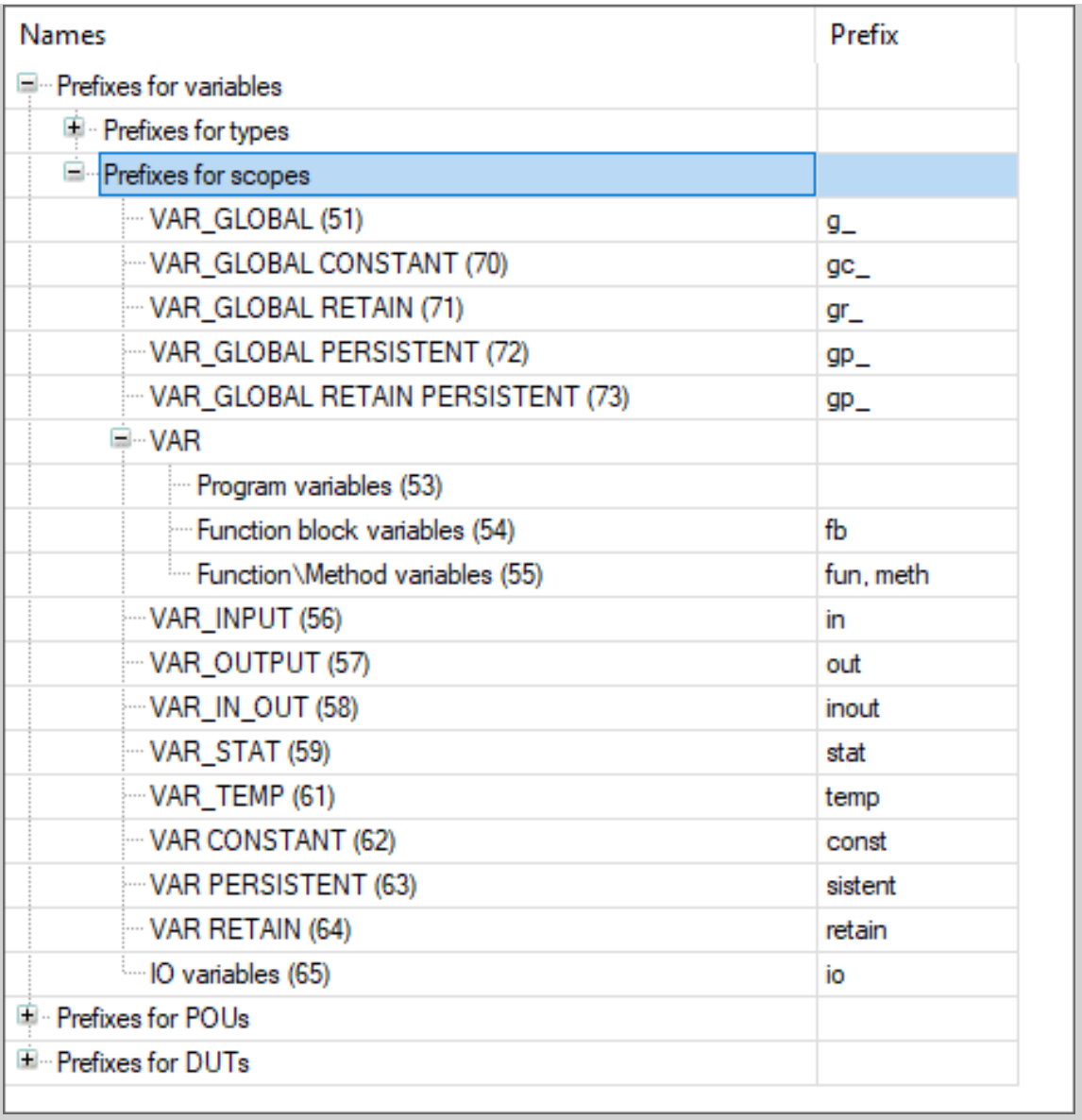
“Combine scope prefix with data type prefix”	☑: As its namespace, a variable must have the defined prefix followed by the defined prefix for its data type. Example: The following prefixes are defined: `g_` for *“VAR_GLOBAL”*, and `r` for the data type *“REAL”*. The code analysis reports errors for global REAL variables that do not have the prefix `g_r`. ☐: If conventions for the namespace are specified for a variable, then these conventions are taken into account. As a result, any data type conventions are ignored. Example: The following prefixes are defined: `g_` for *“VAR_GLOBAL”*, and `r` for the data type *“REAL”*. The code analysis reports exclusively errors for global `REAL` variables that do not have the prefix `g_`.

“Recursive prefixes for combinable data types”	☑: Variables of combined data types have to have compound prefixes that follow the defined naming conventions.
	Example:
	`ppiVariable : POINTER TO POINTER TO INT;`
	The prefix `p` was defined for variables of data type `POINTER`, and the prefix `I` was defined for the data type `INT`. Static analysis reports errors for all variables of type `POINTER TO POINTER TO INT` which do not have the prefix `ppi`.
	`refaiVar : REFERENCE TO ARRAY[1..3] OF INT;`
	The prefix `ref` was defined for the data type `REFERENCE TO`, the prefix `a` for an array, and the prefix `I` for the data type `INT`. Static analysis reports errors for all variables of type `REFERENCE TO ARRAY[1..3] OF INT` which do not have the prefix `refai`.

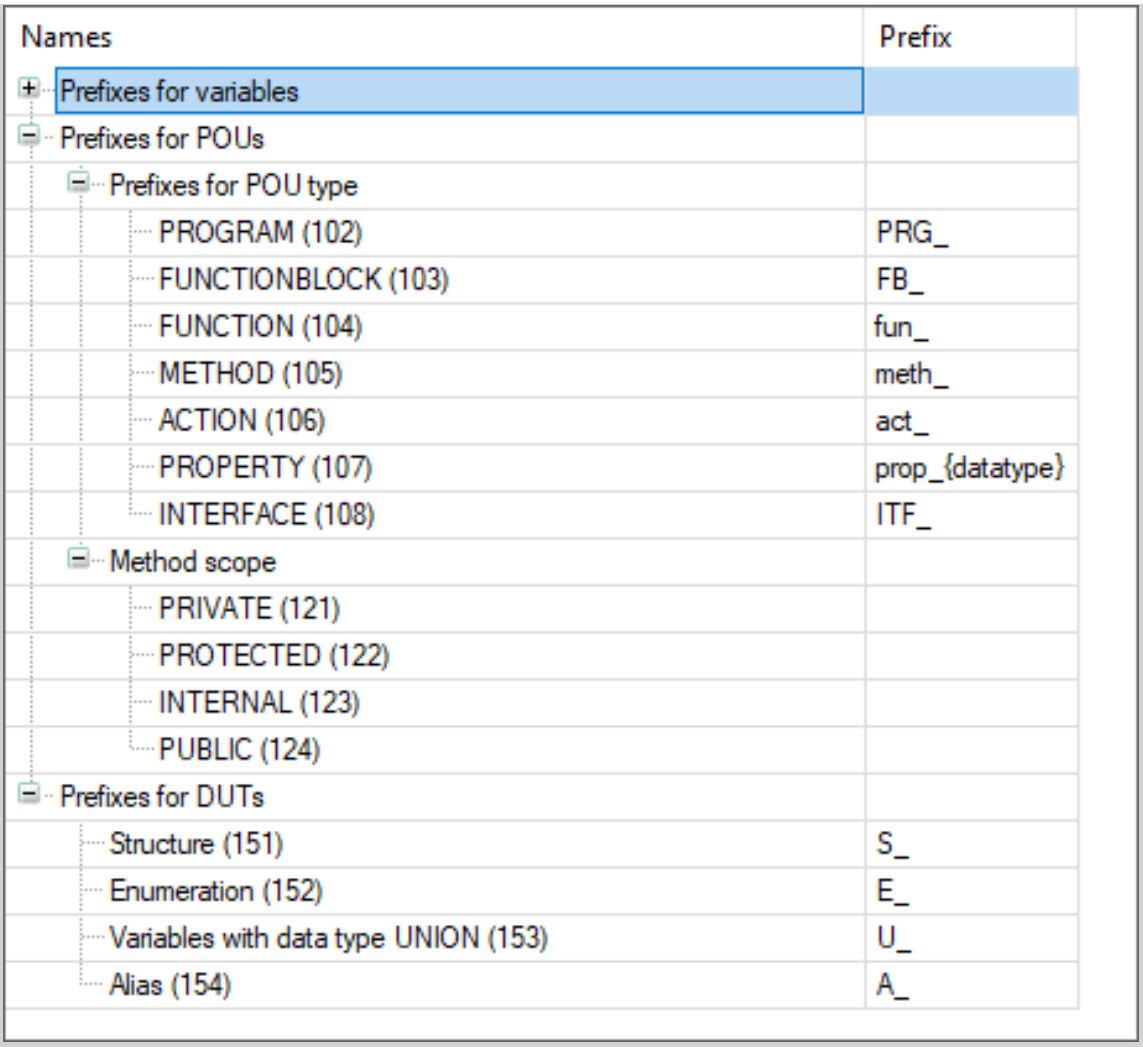
Example	The following naming convention corresponds for the most part to the recommendations described in the "Identifiers" chapter.



Names	Prefix
Prefixes for variables	
Prefixes for types	
BOOL (3)	x
BIT (4)	bit
BYTE (5)	by
WORD (6)	w
DWORD (7)	dw
LWORD (8)	lw
SINT (13)	si
INT (14)	i, n
DINT (15)	di
LINT (16)	li
USINT (9)	usi
UINT (10)	ui
UDINT (11)	udi
ULINT (12)	uli
REAL (17)	r
LREAL (18)	lr
STRING (19)	s
WSTRING (20)	ws
TIME (21)	tim
LTIME (22)	ltim
DATE (23)	dat
DATE_AND_TIME (24)	dt
TIME_OF_DAY (25)	tod
POINTER (26)	p
REFERENCE (27)	ref
SUBRANGE (28)	range
ARRAY (30)	a
Function block instance (31)	fb
Interface (36)	itf
Structure (32)	struct
ENUM (29)	e
Alias (33)	{datatype}
Variables with data type UNION (34)	
__XWORD (35)	xw
__UXINT (37)	xui
__XINT (38)	xi
Prefixes for scopes	
Prefixes for POUs	
Prefixes for DUTs	



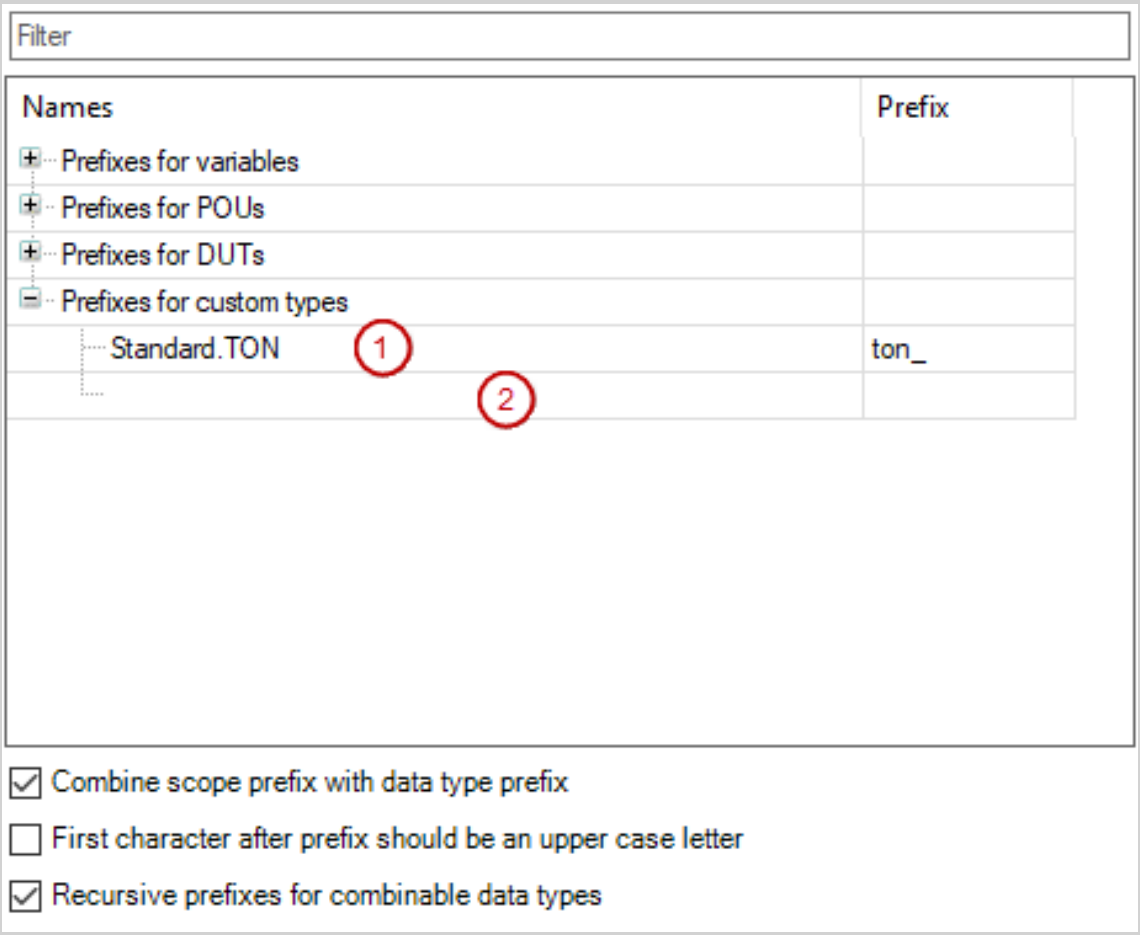
Names	Prefix
Prefixes for variables	
Prefixes for types	
Prefixes for scopes	
VAR_GLOBAL (51)	g_
VAR_GLOBAL CONSTANT (70)	gc_
VAR_GLOBAL RETAIN (71)	gr_
VAR_GLOBAL PERSISTENT (72)	gp_
VAR_GLOBAL RETAIN PERSISTENT (73)	gp_
VAR	
Program variables (53)	
Function block variables (54)	fb
Function\Method variables (55)	fun, meth
VAR_INPUT (56)	in
VAR_OUTPUT (57)	out
VAR_IN_OUT (58)	inout
VAR_STAT (59)	stat
VAR_TEMP (61)	temp
VAR CONSTANT (62)	const
VAR PERSISTENT (63)	sistent
VAR RETAIN (64)	retain
IO variables (65)	io
Prefixes for POUs	
Prefixes for DUTs	



Names	Prefix
Prefixes for variables	
Prefixes for POUs	
Prefixes for POU type	
PROGRAM (102)	PRG_
FUNCTIONBLOCK (103)	FB_
FUNCTION (104)	fun_
METHOD (105)	meth_
ACTION (106)	act_
PROPERTY (107)	prop_{datatype}
INTERFACE (108)	ITF_
Method scope	
PRIVATE (121)	
PROTECTED (122)	
INTERNAL (123)	
PUBLIC (124)	
Prefixes for DUTs	
Structure (151)	S_
Enumeration (152)	E_
Variables with data type UNION (153)	U_
Alias (154)	A_

Example

The naming convention (1) refers to the standard POU `TON`. As a result, declarations of the special library POU are checked for the prefix "ton_". Click the blank space (2) to insert more naming conventions.



See also

- *“Checking for compliance to defined naming conventions” on page 4132*
- *Chapter 1.8.3.3.1.3 “Attribute 'naming'” on page 4150*
- *Chapter 1.8.3.3.1.4 “Attribute 'nameprefix'” on page 4151*
- *Identifiers*
- *Data Type Alias*
- *PROPERTY*

Dialog 'Static Analysis Settings' - 'Metrics'

Function: In the dialog, you select the metrics to be displayed for each POU in the *“Standard Metrics”* view by means of the *“Build ➔ Static Analysis ➔ View Standard Metrics”* command.

Call:

- *“Open configuration dialog”* button in the menu *“Project ➔ Project Settings”*, *“Static Analysis”* category
- Menu bar: *“Build ➔ Static analysis ➔ Settings”*

Requirement:

- The CODESYS Static Analysis package is installed.
- A project is open.

The “Code size”, “Variable size”, “Stack size”, and “Calls” metrics are reported only for POUs from libraries which are integrated in the project.

Violations of the upper and lower limits of the activated metrics can be reported as build errors by means of static analysis rule `SA0150`.

“Metrics”	All selectable metrics are displayed in the column.
“Active”	☑: The metric is displayed for each POU in the *“Standard Metrics”* view following the *“Build ➔ Static Analysis ➔ View Standard Metrics”* command. ☐: The metric is not displayed in the *“Standard Metrics”* view following the *“Build ➔ Static Analysis ➔ View Standard Metrics”* command.
“ Lower limit”	Lower value from which the *“Metric”* is displayed
“Upper Limit”	Upper value to which the *“Metric”* is displayed

See also

- *“Displaying of metrics” on page 4133*
- *Chapter 1.8.3.3.2.52 “SA0150: Violations of lower or upper limits or the metrics” on page 4220*

Dialog 'Static Analysis Settings' - 'Forbidden Symbols'

Function: In the dialog, you define the keywords and symbols that must not be used in the project code.

Call:

- *“Open configuration dialog”* button in the menu *“Project ➔ Project Settings”*, *“Static Analysis”* category
- Menu bar: *“Build ➔ Static analysis ➔ Settings”*

Requirement:

- The CODESYS Static Analysis package is installed.
- A project is open.

Input line	Double-clicking the line opens the line editor for specifying a keyword or symbol. …: The Input Assistant opens for selecting the keyword or symbol.

See also

- *“Checking for forbidden symbols” on page 4132*

1.8.3.3 Reference, Programming

1.8.3.3.1 Pragmas and Attributes

CODESYS Static Analysis provides pragmas and attributes for activating or deactivating individual rules or naming conventions for static code analysis.

Requirement: The rules or conventions are activated or defined in the project settings.

Attributes are inserted in the declaration part of a POU to deactivate specific rules for an entire programming object.

Pragmas are used in the implementation part of a POU to deactivate specific rules for individual lines of code. One exception is Rule 164, which can also be switched off in the declaration part.

Rules that are deactivated in the project settings cannot be activated by means of pragmas or attributes.

Rule SA0004 cannot be deactivated by means of a pragma or an attribute.

See also

- *Chapter 1.8.3.3 “Reference, Programming” on page 4148*

Pragma 'analysis'

This pragma is used to deactivate the code rules for individual code lines of a POU. You deactivate code rules by specifying the rule numbers with a prepended minus sign ("-"). A prepended plus sign ("+") activates the rule. You can specify any number of rules in the pragma.

Insert location: Deactivation: In the implementation part, with `{analysis - ...}` before the first code line where the code analysis is deactivated. Activation: With `{analysis + ...}` after the last line of the deactivation. For Rule 164, the pragma can also be inserted in the declaration part before a comment.

Syntax:

```
Deactivation of rules:

{analysis -<rule number> ( , -<additional rule number> )* }
* : optional none, one or more additional rule numbers

Activation of rules:

{analysis +<rule number> ( , +<additional rule number> )* }
* : none, one or more additional rule numbers
```

Example

Rule 24 is deactivated for two lines and then reactivated. As a result, rule 24 is not checked in these lines so that `nTest:=DINT#99` is allowed for example.

```
  {analysis -24}
 nTest := 99;
 iVar := INT#2;
 {analysis +24}
```

Deactivating multiple rules:

```
 {analysis -10, -24, -18}
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Attribute 'analysis'

The attribute deactivates specific rules for an entire programming object. You deactivate the code rules by specifying the rule numbers with a prepended minus sign ("-"). You can specify any number of rules in the attribute.

Insert location: In the declaration part of a POU, in the first line.

Syntax:

```
{attribute 'analysis' := '-<rule number> ( , -<additional rule
number> )* '}
* : none, one or more additional rule numbers
```

Example

Rules 33 and 31 are deactivated for the entire structure:

```
{attribute 'analysis' := '-33, -31'}
TYPE My_Structure :
STRUCT
    iLocal : INT;
    uiLocal : UINT;
    udiLocal : UDINT;
END_STRUCT
END_TYPE
```

Rule 100 is deactivated for the array:

```
{attribute 'analysis' := '-100'}
PROGRAM PLC_PRG
VAR
    aBigData: ARRAY[1..10000] OF DWORD;
    aBigDATA_2: ARRAY[1..10000] OF DWORD;
END_VAR
;
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Attribute 'naming'

The attribute marks the code lines that are excluded from the analysis of naming convention. An `off` is assigned to the pragma attribute before the first code line where the code analysis is deactivated. An `on` is assigned after the last line. When an `omit` is assigned, only the next code line is ignored.

Insert location: Deactivation: In the declaration part of POUs and DUTs, above the affected lines. Activation: Below the affected lines.

Syntax:

```
{attribute 'naming' := '<switch state>'}
<switch state> : on | off | omit
on : naming is switched on
off : naming is switched off
omit : only next code line is switched off
```

Example

Defined naming conventions: 1) INT variable names must be prepended with "int" as the identifier prefix, for example "intVar1". 2) Program names must begin with "prog".

For the code presented below, the static analysis issues messages only for the following variables: `cccVar`, `aVariable`, and `bVariable`.

```
VAR
{attribute 'naming' := 'off'}
    iVarA : INT;
    iVarB : INT;
{attribute 'naming' := 'on'}
    iVarC : INT;
END_VAR

VAR
    ...
{attribute 'naming' := 'omit'}
    iVarC : INT;
...
END_VAR

{attribute 'naming' := 'omit'}
PROGRAM PLC_PRG
VAR
...
END_VAR

{attribute 'naming' := 'off'}
PROGRAM DoSomethingA
VAR
{attribute 'naming' := 'on'}
    iVarA : INT;
    iVarB : INT;
        …
VAR_END
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Attribute 'nameprefix'

The attribute defines a prefix for variables of a structured data type. The prefix must be prepended to the identifier of variables that are declared by this type.

Insert location: In the line before the declaration of a structured data type

Syntax:

```
{attribute 'nameprefix' := '<prefix>'}
```

Example

In the following example, Static Analysis issues a message for pB because the variable name does not begin with "point".

```
{attribute 'nameprefix' := 'point'}
TYPE DATAPOINT :
STRUCT
    iX: INT;
    iY: INT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    pointA : DATAPOINT;
    pB : DATAPOINT;
END_VAR
pointA.iX := 1;
pointA.iY := 10;
pB.iX := 2;
pB.iY := 20;
```

Error message after static analysis: *“Invalid variable name 'pB'. Expect prefix 'point'”*

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Attribute 'analysis:report-multiple-instance-calls'

The attribute marks a function block for checking for rule 105: Only function blocks with this attribute are checked whether the function block instances are called more than one time. If rule 105 is deactivated in the project settings, then the attribute does not have any effect.

Insert location: Top line in the declaration part of a function block.

Syntax:

```
{attribute 'analysis:report-multiple-instance-calls'}
```

Example

```
// {attribute 'analysis:report-multiple-instance-calls'} Deactivated
FUNCTION_BLOCK FB_DoA
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iA : INT;
END_VAR
iA := iA + 1;

{attribute 'analysis:report-multiple-instance-calls'}
FUNCTION_BLOCK FB_DoB
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iB : INT;
END_VAR
iB := iB +1;

PROGRAM PLC_PRG
VAR
    fbA : FB_DoA;
    fbB : FB_DoB;
ND_VAR

fbA();
fbB();    // SA0105
fbA();
fbB();    // SA0105

--> SA0105: Instance 'fbB' called more than once
```

See also

- *Chapter 1.8.3.3.2.46 “SA0105: Multiple instance calls” on page 4206*
- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

1.8.3.3.2 Rules

SA0001: Unreachable code

Detects lines of code that are not executed, for example due to a RETURN or CONTINUE statement

Justification: Unreachable code should always be avoided. The test often indicates that test code still exists which should be removed.

Importance: High

PLCopen rule: CP2

Example

```
PROGRAM PLC_PRG
VAR
    xReturn_Before_End: BOOL;
    xContinue_In_Loop_FUN: BOOL;
    iCounter: INT;
END_VAR

xContinue_In_Loop_FUN := FALSE;
FOR iCounter := INT#0 TO INT#5 BY INT#1 DO
  CONTINUE;
  xContinue_In_Loop_FUN := FALSE;
END_FOR

--> SA0001: Unreachable code detected in 'PLC_PRG'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0002: Empty objects

Detects POUs, GVLs, data type declarations, or interfaces that do not contain any code

Justification: Empty objects should be avoided. They are often a sign that an object has not been implemented completely. Exception: In some cases, no code is specified in the body of a function block when it should be used by interfaces only. In other cases, a method is created only because it is required by an interface without a sensible implementation being possible for the method. No matter the case, this kind of situation should be commented.

Importance: Medium

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0003: Empty statements

Detects lines of code that have a semicolon (`;`) but not a statement

Justification: An empty statement can be a sign for missing code.

Note: There are good reasons for using empty statements. For example, in a CASE statement it can make sense to explicitly program out all cases, even those where there is nothing to do. When this kind of empty CASE statement contains a comment, Static Analysis does not generate an error message.

Importance: Low

Examples

```
CASE value OF
1:DoSomething();
2:;
3:DoSomethingElse();
END_CASE

--> SA0003: Empty statements
```

```
CASE value OF
1:DoSomething();
2:;              //nothing to do
3:DoSomethingElse();
END_CASE

--> No SA error
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0004: Multiple write access on output

Detects outputs that are written to more than one location

Justification: The maintainability is degraded when an output is written in different locations in the code. Then it is uncertain which write access is the one that actually has an effect in the process. Good practice is to calculate the output variables in auxiliary variables and assign the calculated value at one location at the end of the cycle.

Importance: High

PLCopen rule: CP12

An error is not issued when an output variable (`VAR_IN_OUT`) is written in different branches of `IF` and `CASE` statements.

A pragma cannot deactivate this rule.

Example

```
VAR_GLOBAL
    g_xVar AT %QX0.0 : BOOL ;
    g_iTest AT %QW0 : INT ;
END_VAR

PROGRAM PLC_PRG
IF g_iCondition < INT#0 THEN
    g_xVar := TRUE;
    g_iTest := INT#12;
END_IF

CASE g_iCondition OF
    INT#1:
        g_xVar := FALSE;
    INT#2:
        g_iTest := INT#11;
    ELSE
        g_xVar := TRUE;
        g_iTest := INT#9;
END_CASE

--> SA0004: Multiple write access on output '%QX0.0'
--> SA0004: Multiple write access on output '%QW0'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0006: Write access from several tasks

Detects variables that are written by more than one task

Justification: A variable that is written in multiple tasks may change its value unexpectedly. This can lead to confusing situations. String variables (and on some 32-bit systems also 64-bit integer variables) can even reach an inconsistent state if the variable is written to two tasks simultaneously.

Exception: In specific cases, it may be necessary for several tasks to write a variable. For example, use semaphores to make sure that access does not lead to an inconsistent state.

Importance: High

PLCopen rule: CP10

Example

```
VAR_GLOBAL
    g_iTemp1: INT;
END_VAR

PROGRAM PLC_PRG    // Controlled by MainTask
g_iTemp1 := g_iTemp1 + INT#2;

PROGRAM PLC_PRG_1  //Controlled by SubTask
g_iTemp1 := g_iTemp1 - INT#3;


--> SA0006: Concurrent write access to 'g_iTemp1' in Tasks
MainTask, SubTask
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0007: Address operator on constants

Detects lines of code where the operator `ADR` is applied for a constant

Justification: Using a pointer to a constant variables overrides the `CONSTANT` property of the variable. The variable can be changed by means of the pointer without any notification from the compiler.

Exception: In rare cases, it might be useful to pass a pointer to a constant to a function. However, you have to make sure that this function does not change the transferred value. Whenever possible, use `VAR_IN_OUT CONSTANT`.

Importance: High

When the “replace constants” option is selected in the “Compiler options” of the project settings, the address operator is not permitted for scalar constants (integer, BOOL, REAL) and a complie error is issued. (Constant strings, structures, and arrays always have an address.)

Example

```
PROGRAM PLC_PRG
VAR CONSTANT
    c_iValue : INT := INT#15;
END_VAR
VAR
    poiValue : POINTER TO INT;
END_VAR
poiValue := ADR(c_iValue); // SA0007

--> SA0007: Address to constant variable 'c_iValue'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0008: Check subrange types

Detects out-of-range violations of subrange types. Assigned literals are already checked by the compiler. When constants are assigned, then the values must be within the defined range. When variables are assigned, then the data types must be identical.

Justification: If subrange types are used, then make sure that this subrange is not exited. The compiler checks for these kinds of subrange violations only for assignments of constants.

Importance: Low

The check is not performed for CFC objects because the code structure does not allow for it.

Example

```
VAR_GLOBAL
      iVarGlob:INT;
END_VAR

PROGRAM PLC_PRG
VAR
   iSubr1: INT (INT#1..INT#10);
   iSubr2: INT (INT#1..INT#1000);
   iCount: INT;
   by_SubType : BYTE (BYTE#0..BYTE#11);
   iVar : INT (-4095..4095);
END_VAR
   iSubr1 := nCount;              // SA0008
   iSubr1 := subr2;               // SA0008
   iSubr1 := gvl.iVarGlob;        // SA0008
   //byBYTE_SubType := BYTE#123;  //already detected by compiler,
error "Cannot convert type..."

--> SA0008: Subrange variable 'iSubr1' maybe out of allowed range
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0009: Unused return values

Detects function, method and property calls in which the return value is not used

Justification: When a function or method returns a return value, it should also be evaluated. The return value often indicates whether or not the function was executed successfully. If not, then you will not be able to identify later whether the return value was forgotten or if it is actually not needed.

Exception: If a return value is irrelevant to the call, then you can document this and omit the assignment. Error returns should never be ignored.

Importance: Medium

PLCopen rule: CP7 / CP17

Example

```
FUNCTION Return_BOOL : BOOL
VAR_INPUT
END_VAR
VAR
    xTest : BOOL;
END_VAR
xTest := FALSE;
Return_BOOL := xTest;

PROGRAM PLC_PRG

Return_BOOL ();  // SA0009

--> SA0009: Ignoring return value of 'Return_Bool'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0010: Arrays with only one component

Detects arrays with only one element

Justification: An array with one element can be replaced by a base-type variable. Access to this variable is considerably faster than access by index to the variable.

Exception: The length of an array is often determined by a constant and is a parameter for a program. Then the program can work with arrays of different lengths and does not have to be changed if the length is only 1. This kind of situation should be documented accordingly.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    aoiEmpty : ARRAY [22..22] OF INT;
    aorEmpty : ARRAY [1..1] OF REAL;
END_VAR

aoiEmpty;
aorEmpty;

--> SA0010: Vacuous array element in variable 'aoiEmpty'
--> SA0010: Vacuous array element in variable 'aorEmpty'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0011: Useless declarations

Detects structures or enumerations with only one component

Justification: This kind of declaration can be confusing for the reader. A structure with only one element can be replaced by an alias type. An enumeration with only one element can be replaced by a constant.

PLCopen rule: CP22 / CP24

Importance: Low

Example

```
TYPE SingleStruct :
STRUCT
    iPart : INT;
END_STRUCT
END_TYPE

TYPE myUnion :
UNION
    lrValue : LREAL;
END_UNION
END_TYPE

TYPE SingleEnum :
(
    OnlyOne := 1
);
END_TYPE

--> Useless declaration 'SingleStruct'
--> Useless declaration 'myUnion'
--> Useless declaration 'SingleEnum'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0012: Variable which could be declared as constants

Detects variables that are not accessed with write permission and therefore could be declared as constants

Justification: If a variable is written only at the declaration point and is otherwise used only for reading, then the static analysis assumes that the variable should also not to be changed. Firstly, a declaration as a constant results in checking that the variable is not changed when the program is changed. Secondly, the declaration as a constant may result in faster code.

> **NOTICE!**
> If multiple applications exist in one project, then only the objects below the currently active application are affected. If there is only one application, then the objects in the common POU pool are also affected.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    iVar : INT := INT#17;
    iTest : INT;
END_VAR
iTest := iTest + iVar;   // SA0012: iVar could be declared as
constant

--> SA0012: Variable 'iVar' could be declared as constant
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0013: Declarations with the same variable name

Detects variables with names that are already used by other variables (for example, global and local variables with the same name). Also detects variables with names of functions, actions, methods, or properties which are used in the same access scope. Variables are also detected that are declared in a GVL in the *“Devices”* view or in the POUs pool. For this, however, the GVL of the *“POUs”* view have to be used in the application program.

Justification: The same names can be confusing when reading the code, and they can cause errors if the wrong object is accessed unintentionally. We recommend that you use naming conventions to avoid these situations.

PLCopen rule: N5 / N9

Importance: Medium

Example

```
VAR_GLOBAL
    xVar1 : BOOL;
    iVar3 : INT;
END_VAR

PROGRAM PLC_PRG
VAR
    xVar1 : BOOL;  // SA0013
    iVar3 : INT;   // SA0013
END_VAR

xVar1 := NOT GVL.xVar1;
iVar3 := iVar3 + INT#2;
iVar3 := GVL.iVar3;

--> SA0013: Declaration of 'iVar1' hides symbol 'GVL.iVar1
--> SA0013: Declaration of 'xVar3' hides symbol 'GVL.xVar3
```

Example The function block `POU` has the action `ACT` and the method `METH`.

```
FUNCTION_BLOCK POU
VAR
    ACT : UINT;  // SA0013
    METH : BYTE; // SA0013
END_VAR

--> SA0013: Declaration of 'ACT' hides symbol 'POT.ACT'
--> SA0013: Declaration of 'METH' hides symbol 'POT.METH'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0014: Assignment of instances

Detects assignments to function block instances. In the case of instances with pointer or reference variables, these assignments are potentially risky.

Justification: This is a performance warning. When an instance is assigned to another instance, all elements and subelements are copied from the one instance to the other instance. Pointers to data are also copied, but not their referenced data, so that the target instance and the source instance contain the same data after the assignment. Depending on the size of the instances, this kind of assignment could last a long time. For example, if an instance should be passed to a function for processing, then it is much more efficient to pass a pointer to the instance. If you want to selectively copy values from one instance to another, then a copy method is useful: `inst_First.Copy_From(inst_Second)`.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    inst_First : My_FB;
    inst_Second : My_FB;
END_VAR
inst_First();
inst_Second := inst_First;  // SA0014

--> SA0014: Assignment of instances
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0015: Access to global data via FB_Init

Detects the access of a function block to global variables by means of the method `FB_Init`. The value of this variable depends on the order of initializations.

Justification: Depending on the declaration location of the POU instance, an uninitialized variable could be accessed if the rule is violated.

Importance: High

Example

```
VAR_GLOBAL
    g_xTest1 : BOOL;
    g_iTest3 : INT;
END_VAR

METHOD PUBLIC fb_init : BOOL
VAR_INPUT
  bInitRetains : BOOL; // If TRUE, the retain variables are
initialized (warm start / cold start)
  bInCopyCode : BOOL;  // If TRUE, the instance afterwards gets
moved into the copy code (online change)
END_VAR
g_xTest1 := NOT g_xTest1;       // SA0015
g_iTest3 := g_iTest3 + INT#1;   // SA0015

--> SA0015: FB_Init method of function block 'POU' accesses global
data
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0016: Gaps in structures

Detects gaps in structures or function blocks that are caused by the alignment requirements of the currently set target system. If possible, you should remove the gaps by resorting the structure elements or filling them with a dummy element. If this is not possible, then you can deactivate the rule for the affected structures by means of the `analysis` pragma.

Justification: Due to different alignment requirements on different platforms, there may be a different layout in the memory for these kinds of structures. Then the code can perform differently, depending on the platform.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    myStruct : Unpadded_Structure;
END_VAR
myStruct.iTest := 0;

TYPE Unpadded_Structure :
STRUCT
    xTest : BOOL;
    iTest : INT;    // SA0016
    byTest : BYTE;
    wTest : WORD;
END_STRUCT
END_TYPE

--> SA0016: Structure 'Unpadded_Structure' must be padded (pack-
mode=8)
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0017: Non-regular assignments

Detects assignments to pointers that are neither addresses (`ADR` operator, pointer variables) nor constants `0`

Justification: If a pointer contains a value that is not a valid address, then an access violation exception occurs when dereferencing the pointer.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    pInt : POINTER TO INT;
    dwAddress : DWORD;
END_VAR
dwAddress := dwAddress + DWORD#1;
pInt := dwAddress; // SA0017

--> SA0017: Non-regular assignment
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0018: Unusual bit access

Detects bit access to signed variables. However, the IEC 61131-3 standard permits only bit access and bit shift operations on bitfields. See also the strict rules SA0147 and SA0148.

Justification: Signed data types should not be used as bitfields and the other way around. The IEC 61131-3 standard does not provide for this kind of access, and therefore you should comply with this rule when you write portable code.

Importance: Medium

Exception for flag enumerations: When an enumeration is declared as a flag by means of the `{attribute 'flags'}` pragma attribute, the `SA0018` error is not issued for bit access with the `OR`, `AND` or `NOT` operators.

Example

```
PROGRAM PLC_PRG
VAR
    iTemp1 : INT;
    diTemp3 : DINT;
    uliTemp4 : ULINT;
    siTemp5 : SINT;
    usiTemp6 : USINT;
    byTemp2 : BYTE;
END_VAR
iTemp1.3 := TRUE;     // SA0018
diTemp3.4 := TRUE;    // SA0018
uliTemp4.18 := FALSE; // no error because this is an unsigned data
type
siTemp5.2 := FALSE;   // SA0018
usiTemp6.3 := TRUE;   // no error because this is an unsigned data
type
byTemp2.5 := FALSE;   // no error because the byte is a bitfield

--> SA0018: Unusual bit access
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*
- *Chapter 1.8.3.3.2.49.12 “SA0147: Unusual shift operation - strict” on page 4214*
- *Chapter 1.8.3.3.2.49.13 “SA0148: Unusual bit access - strict” on page 4215*

SA0020: Possibly assignment of truncated value to REAL variable

Detects operations on integer variables for which a truncated value could be assigned to a `REAL` data type variable

Justification: Static analysis issues an error when the result of an integer calculation is assigned to a `REAL` or `LREAL` variable. The programmer should be alerted to a possible incorrect interpretation of this kind of assignment: `lrealvar := dintvar1 * dintvar2`. Because the range of values of `LREAL` is greater than that of `DINT`, one could assume that the result of the calculation could always be represented in `LREAL`. But that is not the case. The processor calculates the result of the multiplication as an integer and then casts the result to `LREAL`. An overflow in the integer calculation would be lost. To work around the problem, the calculation has to be done as a `REAL` operation: `lreal_var := TO_LREAL(dintvar1) * TO_LREAL(dintvar2)`.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
 rx : LREAL;
 di : DINT;
END_VAR
rx := di * di                                  // SA0020
rx := TO_LREAL(di) * TO_LREAL(di) // No message

--> SA0020: Possibly assignment of truncated value to REAL variable
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0021: Transporting the address of a temporary variable

Detects address assignments of temporary variables (on the stack) to non-temporary variables

Justification: Local variables of a function or method are created on the stack and they exist only while the function or method is being processed. If a pointer points to this kind of variable after processing the method or function, then you can use this pointer to access undefined memory, or to access an incorrect variable in another function. This situation should be avoided at all costs.

Importance: High

Example

```
FUNCTION TempVarInFUNC : DWORD
VAR
    uiTemp : UINT;
END_VAR
TempVarInFUNC := ADR(uiTemp);    // SA0021

PROGRAM PLC_PRG
VAR
    dwTest : DWORD;
END_VAR
dwTest := TempVarInFUNC();

--> SA0021: Transporting address of temporary variable to outer
scope symbol
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0022: (Possibly) unassigned return value

Detects all functions and methods that include an execution thread without an assignment to the return value

Justification: An unassigned return value in a function or method is an indication of missing code. Even if the return value always has a default value, it is always useful to assign it again explicitly to avoid confusion.

Importance: Medium

Example

```
FUNCTION FUN : DINT
VAR_INPUT
    bTest : BOOL;
END_VAR

IF bTest THEN
    RETURN;
END_IF
FUN := 99;

--> SA0022: (Possibly) unassigned return value
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0023: Complex return values

Determines complex return values that cannot be returned with a simple registry copy of the processor. This includes structures, arrays, and return values of type `STRING` (regardless of the size of the used memory).

Justification: This is a performance warning. If large values are returned as the result of a function, method, or property, then the processor copies them multiple times when executing the code. This can lead to runtime problems and should be avoided whenever possible. Performance can be improved by passing a structured value as `VAR_IN_OUT` to a function or method and filling it in the function or method.

Importance: Medium

Example

```
TYPE LargeStructure :
STRUCT
    a : LINT;
    b : BOOL;
END_STRUCT
END_TYPE

FUNCTION Large_Return_Value_FUNC : LargeStructure   // SA0023

--> SA0023: Complex return values
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0024: Untyped literals / constants

Detects untyped literals and constants

Justification: CODESYS assigns types for literals depending on their use. In some cases, this can cause unexpected problems, which should be resolved better with a typed literal. For example: `dw := ROL(DWORD#1, i)`

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    iTemp1 : INT = 10;     // SA0024
    diTemp2 : DINT;
    liTemp3 : LINT;
    rTemp4 : REAL;
    lrTemp5 : LREAL;
END_VAR
iTemp1 := iTemp1 + INT#34;
diTemp2 := diTemp2 + 23;   // SA0024
liTemp3 := liTemp3 + 124;  // SA0024
rTemp4 := rTemp4 + 1.1;    // SA0024
lrTemp5 := lrTemp5 + 3.4;  // SA0024

--> SA0024: Untyped literal found
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0025: Unqualified enumeration constants

Detects enumeration constants for which a qualified name does not prepend the enumeration

Justification: Qualified access makes the code more readable and easier to maintain. Without forcing qualified variable names, an additional enumeration could be inserted when the program is extended. This enumeration contains a constant with the same name as an existing enumeration (see the example below: "red"). This would result in ambiguous access to this piece of code. We recommend to always use only enumerations with the {attribute 'qualified-only'}.

Importance: Medium

Example

```
TYPE COLOR
(red,green,blue);
END_TYPE

PROGRAM PLC_PRG
enumVar : COLOR;

enumVar := COLOR.red; // SA0025
enumVar := red;       // SA0025

--> SA0025: Enumeration constant 'red' not qualified
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0026: Possible truncated strings

Detects string assignments and string initializations that do not use sufficient string length

Justification: When strings of different lengths are assigned, a string could be truncated. This can have unexpected results.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    strVar1 : STRING[10];
    strVar2 : STRING[6];
    strVar3 : STRING[6] := 'abcdefghi';  // SA0026
END_VAR

strVar2 := strVar1;                      // SA0026

--> SA0026: Truncation of string 'abcdefghi'
--> SA0026: Possible truncation of string 'strVar1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0027: Multiple uses of identifiers

Detects multiple uses of a name/identifier for a variable or an object (POU) within the scope of a project

Justification: Same names can be confusing when reading the code. They can cause errors if the wrong object is accessed accidentally. Define and follow naming conventions to avoid any situation like this.

The following cases are detected:

- The name of an enumeration is identical to the name of another enumeration in the application or in an integrated library.
- The name of a variable is identical to the name of another object in the application or in an integrated library.
- The name of a variable is identical to the name of an enumeration constant in an enumeration in the application or in an integrated library.
- The name of an object is identical to the name of another object in the application or in an integrated library.

Importance: Medium

Example

The `Standard` library is integrated in the project and provides the `TON` function.

```
PROGRAM PLC_PRG
VAR
ton : INT;
END_VAR

--> Variable name 'ton' in 'PLC_PRG' already used for an object in
library 'standard, ...'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0028: Overlapping memory areas

Detects the lines of code where two or more variables reserve the same memory

Justification: When two variables reserve the same memory, the code may behave with unexpected results. This situation should be avoided at all costs. If you cannot avoid using a value in different interpretations (for example, one time as `DINT` and another time as `REAL`), then you should define a `UNION`. You can also use a pointer to access a value with a different type without the value being converted.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
iVvar1 AT %QB21: INT;
dwVar2 AT %QD5: DWORD;
END_VAR

--> The following variables access the same memory:
--> SA0028: iVar1 AT %QB21
--> SA0028: dwVar2 AT %QD5
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0029: Notation in code different to declaration

Detects the code locations where the notation of an identifier is different from the notation in its declaration

Justification: The IEC 61131-3 standard defines identifiers as not case-sensitive. This means that a variable declared as "varx" can also be used as "VaRx" in the code. However, this is confusing and misleading and should be avoided.

Importance: Medium

Example A POU `PLC_PRG` and a POU `fnc` (function) exist in the device tree.

```
PROGRAM PLC_PRG
VAR
    iVar: INT;
    _123test_var_: INT;
END_VAR

ivar := iVar + 1;                   // SA0029
_123TEST_var_ := _123test_var_;        // SA0029
Fnc();                              // SA0029

--> SA0029: Notation in code (ivar) must equal declaration (iVar)
--> SA0029: Notation in code (_123TEST_var_) must equal declaration
(_123test_var_)
--> SA0029: Notation in code (Fnc) must equal declaration (fnc)
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Unused Objects

SA0031: Unused signatures

Detects programs, function blocks, functions, data types, interfaces, methods, properties, and actions that are not called within the compiled program code

Justification: Unused objects unnecessarily increase the size of the project and can be confusing when reading the code.

Importance: Low

PLCopen rule: CP2

If multiple applications exist in a project, then only the objects below the currently active applications are affected. If there is only one application, then the objects in the POU pool are also affected.

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0032: Unused enumeration constants

Detects enumeration constants that are not used in the compiled program code

Justification: Unused enumeration constants unnecessarily increase the size of the enumeration definition and can be confusing when reading the program.

PLCopen rule: CP24

Importance: Low

If multiple applications exist in a project, then only the objects below the currently active applications are affected. If there is only one application, then the objects in the common POU pool are also affected.

Example

```
TYPE My_Enum :
(
    one := 1, two := 2
);
END_TYPEE

--> SA0032: Unused enumeration constant 'one'
--> SA0032: Unused enumeration constant 'two'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0033: Unused variables

Detects variables that are declared but not used within the compiled program code

Justification: Unused variables make a program less readable and maintainable. Unused variables unnecessarily fill memory and unnecessarily waste runtime during initialization.

Importance: Medium

PLCopen rule: CP22 / CP24

For GVL variables:If multiple applications exist in a project, then only the objects below the currently active applications are affected. If there is only one application, then the objects in the common POU pool are also affected.

Example

```
PROGRAM PLC_PRG
VAR
    iCounter1 : INT;
    iCounter2 : INT;        // SA0035
END_VAR

ICounter1 := 100;

--> SA0035: Unused Variable 'iCounter2'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0035: Unused input variables

Detects input variables that are not used by any function block instance

Justification: Unused variables make a program less readable and maintainable. Unused variables unnecessarily fill memory and unnecessarily waste runtime during initialization.

Importance: Medium

PLCopen rule: CP24

Example

```
FUNCTION_BLOCK AFB
VAR_INPUT
    iIn1: INT;
    iIn2: INT;
END_VAR
VAR_OUTPUT
    iOut1: INT;
END_VAR

PROGRAM PLC_PRG
VAR
    Fb1: AFB;
END_VAR

Fb1(iIn1 := 99)

--> SA0035: Unused input 'iIn2'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0036: Unused output variables

Detects output variables that are not used by any function block instance

Justification: Unused variables make a program less readable and maintainable. Unused variables unnecessarily fill memory and unnecessarily waste runtime during initialization.

Importance: Medium

PLCopen rule: CP24

Example

```
FUNCTION_BLOCK AFB
VAR_INPUT
    iIn1: INT;
    iIn2: INT;
END_VAR
VAR_OUTPUT
    iOut1: INT;
END_VAR

PROGRAM PLC_PRG
VAR
    Fb1: AFB;
END_VAR
Fb1(iIn1 := 99)

--> SA0036: Unused output 'iOut1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0034: Enumerations with incorrect assignment

Detects values that are assigned to an enumeration variable. Only defined enumeration constants of an enumeration variable are permitted to be assigned.

Justification: A variable of the enumeration type should have only the intended values, otherwise the code that uses this variable may not work correctly. We recommend to always use enumerations with the `{attribute 'strict'}`. Then the compiler already checks the correct use of the enumeration components.

Importance: High

Example

```
TYPE COLOR :
(
    Red := 0,
    Green,
    Yellow
);
END_TYPE

PROGRAM PLC_PRG
VAR
    eColor1: COLOR;
END_VAR

eColor1 := COLOR.Red;
eColor1 := 1;         // SA0034

--> SA0034: Use enumeration value instead of 'INT#1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0037: Write access to input variable

Detects input variables (VAR_INPUT) that are accessed with write permission within the POU

Justification: According to the IEC 61131-3 standard, an input variable must not be changed within a POU. This kind of access is also a cause for errors and makes the code poorly maintainable. This is an indication that a variable is used as both an input variable and an auxiliary variable. This kind of dual use should be avoided.

Importance: Medium

Example

```
VAR_GLOBAL
    g_xGlob AT %QX0.0 : BOOL;
END_VAR

PROGRAM PLC_PRG
VAR_INPUT
    xVarIn1:BOOL;
    xVarIn2:BOOL;
END_VAR
VAR
    iCondition : INT;
END_VAR

iCondition := iCondition + INT#1;
CASE iCondition OF
    INT#1:
    g_xGlob := xVarIn1;
    INT#2:
    g_xGlob := xVarIn2;
ELSE
    g_xGlob := FALSE;
        xVarIn1 := FALSE;     // SA0037
END_CASE

--> SA0037: Write access to input variable 'xVarIn1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0038: Read access to output variable

Detects output variables (VAR_OUTPUT) that are accessed with read permission within the POU

Justification: According to the IEC 61131-3 standard, it is prohibited to read an output within a POU. This is an indication that the output is not only used as an output but also as a temporary variable for intermediate results. This kind of dual use should be avoided.

Importance: Low

Example

```
VAR_GLOBAL
    g_xGlob AT %QX0.0 : BOOL ;
    g_iGlob AT %QW1 : INT ;
END_VAR

PROGRAM PLC_PRG
VAR_OUTPUT
    xVarOut1:BOOL;
    xVarOut2:INT;
    xVarOut3:INT;
END_VAR
VAR
    iCondition : INT;
END_VAR

iCondition := iCondition + INT#1;
CASE iCondition OF
    INT#1:
        xVarOut1 := g_xGlob;
        xVarOut2 := g_iGlob;
    INT#2:
        xVarOut3 := xVarOut2;  // SA0038
    ELSE
        xVarOut1 := FALSE;
        g_xGlob := xVarOut1;     // SA0038
        xVarOut2 := INT#0;
        xVarOut3 := INT#-1;
END_CASE

--> SA0038: Read access to output variable 'xVarOUT2'
--> SA0038: Read access to output variable 'xVarOUT1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0040: Possible division by zero

Detects code locations where there is possible division by zero

Justification: Division by zero should never occur, and a variable denominator should always be checked for 0 first.

Importance: High

Example

```
VAR_GLOBAL
    g_iVar AT %QW1 : INT ;
END_VAR

PROGRAM PLC_PRG
VAR
    iCounter : INT;
    iSumme:INT;
    iMid:INT;
    iVal1:INT := INT#2;
    iVal2:INT;
    iVal3:INT := INT#3;
    iVal4:INT := INT#4;
    iVal5:INT;
END_VAR

IF iVal2 <> 0 THEN
iVal1 := iVal1/iVal2;   // no error
END_IF;
iMid := iSumme / iCounter; //  SA0040
iCounter := iCounter + INT#1;
iSumme := g_iVar + iSumme;
IF iMid < INT#100 THEN
    iVal1 := iVal1 / iVal2;       //  SA0040
END_IF

--> SA0040: Possible division by zero
--> SA0040: Possible division by zero
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0041: Detect possible loop invariant code

Detects assignments in loops that calculate the same value for each loop cycle. These lines of code could possibly be inserted outside of the loop.

Justification: This is a performance warning. Code that is executed in a loop, but does the same thing in each loop cycle, can be executed outside of the loop.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    iCounter, iVar1, iVar2: INT;
END_VAR

FOR iCounter := 0 TO 10 DO
    iVar1 := 100;        // SA0041
    iVar2 := iVar2 + iVar1;
END_FOR

--> SAN0041: Possible loop invariant code 'iVar1 := 100'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0042: Usage of different access paths

Detects the usage of different access paths for the same variable

Justification: Different access to the same element decreases the readability and maintainability of a program. We recommend the consistent usage of `{attribute 'qualified-only'}` for libraries, global variable lists, and enumerations. This forces a fully qualified access.

Importance: Low

Example

```
VAR_GLOBAL
        iTemp:INT;
        instPOU:POU;
END_VAR

FUNCTION_BLOCK POU
VAR
    a:INT;
END_VAR
a := INT#1;

PROGRAM SA0042
VAR
    ptiTemp:POINTER TO INT;
    sTemp:STRING;
END_VAR

ptiTemp := ADR(iTemp);

ptiTemp^:= INT#1;
iTemp:= INT#2;                          //  SA0042 - direct access
on variable
GVL.iTemp := INT#3;                     //  SA0042 - access on
variable via GVL

sTemp := CONCAT( 'ab', 'cd');           // SA0042 - direct access on
function
sTemp := Standard.CONCAT( 'ab', 'cd'); // SA0042 - access on
function via Standard

instPOU();                              //  SA0042 - direct access
on POU instance
GVL.instPOU();                          //  SA0042 - access via GVL

--> SA0042: Different access paths for 'CONCAT'
--> SA0042: Different access paths for 'Standard.CONCAT'
--> SA0042: Different access paths for 'instPOU'
--> SA0042: Different access paths for 'GVL.instPOU'
--> SA0042: Different access paths for 'iTemp'
--> SA0042: Different access paths for 'GVL.iTemp'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0043: Use of a global variable in only one POU

Detects the use of a global variable in only a single POU

Justification: A global variable that is used in only one location should also only be declared at this location.

Importance: Medium

PLCopen rule: CP26

Example

```
VAR_GLOBAL
 g_xVar AT %QX0.0 : BOOL ;
    g_iTest AT %QW1 : INT ;
    g_wTest AT %QW2 : WORD;
END_VAR

PROGRAM prog1
VAR
    iCondition : INT;
    bTemp :BOOL;
END_VAR
iCondition := iCondition + INT#1;
IF iCondition < INT#0 THEN
    bTemp := g_xVar;   // SA0043 - g_xVar only read in this POU
ELSIF iCondition = INT#0 THEN
    bTemp := g_xVar;   // SA0043 - g_xVar only read in this POU
ELSE
       bTemp := g_xVar;   // SA0043 - g_xVar only read in this POU
       g_wTest := WORD#4; // g_WTest used also in prog2 -> OK
END_IF

PROGRAM prog2
VAR
    iCondition : INT;
END_VAR
iCondition := iCondition + INT#1;

CASE iCondition OF
    INT#1:
        g_iTest := WORD_TO_INT(g_wTest); // SA0043 - g_iTest only
written in this POU
    INT#2:
        g_iTest := INT#2;               // SA0043 - g_iTest only
written in this POU
    ELSE
        g_iTest := INT#3;               // SA0043 - g_iTest only
written in this POU
END_CASE

--> SA0043: Global variable 'g_xVar' only used in 'prog1'
--> SA0043: Global variable 'g_iTest' only used in 'prog2'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0044: Declarations with reference to interface

Detects declarations with `REFERENCE TO` interfaces and declarations of `VAR_IN_OUT` variables with interfaces (implicitly implemented by means of `REFERENCE TO`)

Justification: An interface type is always implicitly a reference to an instance of a function block that implements this interface. A reference to an interface is therefore a reference to a reference and can result in unwanted behavior.

Importance: High

Example `ITF` is an interface that is defined in the project.

```
PROGRAM PLC_PRG
VAR
    inst:POU;
    itf_inst1 : ITF;
    itf_ref : REFERENCE TO ITF; // SA0044
END_VAR FUNCTION_BLOCK POU
VAR_INPUT
    inst_itf2 : ITF;
END_VAR
VAR_OUTPUT
    inst_itf3 : ITF;
END_VAR
VAR_IN_OUT
    inst_itf4 : ITF;           // SA0044
END_VAR

--> SA0044: Reference to interface 'itf_ref'
--> SA0044: Reference to interface 'itf4_ref'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Conversions

SA0019: Implicit pointer conversions

Detects implicitly generated pointer conversions

Justification: In CODESYS, pointers are not strictly typed and they can be assigned to each other in any way. This is often used and therefore not reported by the compiler. However, it can also accidentally cause unexpected access. If you assign a `POINTER TO BYTE` to a `POINTER TO DWORD`, then you can unintentionally overwrite memory using the latter pointer. Therefore, always check this rule and block the message for cases in which you intentionally want to access a value with a different type.

Implicit data type conversions are reported with a different message.

Importance: High

PLCopen rule: CP25

Exception: `BOOL` <-> `BIT`

Example

```
PROGRAM PLC_PRG
VAR
    pINT : POINTER TO INT;
    byteVar : BYTE;
END_VAR

pINT := ADR(byteVar);

--> SA0019: Implicit conversion from pointer to 'POINTER TO BYTE'
to pointer to 'POINTER TO INT'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0130: Implicit expanding conversions

Detects implicit conversions from smaller data types to larger data types

Justification: The compiler permits any assignments of different types when the value range of the source type is completely contained within the value range of the target type. However, the compiler will build a conversion into the code as late as possible. For an assignment of type `lint := dint * dint`, the compiler performs the implicit conversion only after multiplication: `lint := TO_LINT(dint * dint)`. An overflow is therefore truncated. To prevent this, you can already convert the elements: `lint := TO_LINT(dint) * TO_LINT(dint)`. Therefore, it may be useful to report locations where the compiler implements implicit conversions in order to check whether these are exactly what is intended. Furthermore, explicit conversions can be used to improve portability to other systems when those systems have more restrictive type checks.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
 byTemp : BYTE;
 usiTemp : USINT;
    uiTemp: UINT;
    iTemp : INT;
    udiTemp: UDINT;
 diTemp : DINT;
 uliTemp : ULINT;
    liTemp : LINT;
    lwTemp : LWORD;
    lrTemp : LREAL;
END_VAR

liTemp := iTemp;       // SA0130
uliTemp := usiTemp;    // SA0130
lwTemp := udiTemp;     // SA0130
lrTemp := byTemp;      // SA0130
diTemp := uiTemp;      // SA0130

byTemp.5 := FALSE;     // OK (BIT_BOOL conversion)

--> SA0130: Implicit widening conversion from type 'INT' to type
'LINT'
--> SA0130: Implicit widening conversion from type 'USINT' to type
'ULINT'
--> SA0130: Implicit widening conversion from type 'UDINT' to type
'LWORD'
--> SA0130: Implicit widening conversion from type 'BYTE' to type
'LREAL'
--> SA0130: Implicit widening conversion from type 'UINT' to type
'DINT'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0131: Implicit narrowing conversions

Detects implicit conversions from larger data types to smaller data types

Justification: This message is obsolete now because it is already reported as a warning by the compiler.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    rTemp : REAL;
    lrTemp : LREAL;
END_VAR
rTemp := lrTemp;    // SA0131

--> SA0131: Implicit narrowing conversion from type 'LREAL' to type
'REAL'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0132: Implicit signed/unsigned conversions

Detects implicit conversions from signed data types to unsigned data types or the other way around.

This message is obsolete now because it is already reported as a warning by the compiler.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    byTest :BYTE;
    udiTest: UDINT;
    ulktest: ULINT;
    wTest    : WORD;
    lwTest : LWORD;
    siTest    : SINT;
    iTest    : INT;
    diTest : DINT;
    liTest    :LINT;
END_VAR
liTest := ulktest;    // SA0132
udiTest:= diTest;     // SA0132
siTest := byTest;     // SA0132
wTest := iTest;       // SA0132
lwTest := siTest;     // SA0132

--> SA0132: Implicit signed/unsigned conversion from type 'ULINT'
to type 'LINT'
--> SA0132: Implicit signed/unsigned conversion from type 'DINT' to
type 'UDINT'
--> SA0132: Implicit signed/unsigned conversion from type 'BYTE' to
type 'SINT'
--> SA0132: Implicit signed/unsigned conversion from type 'INT' to
type 'WORD'
--> SA0132: Implicit signed/unsigned conversion from type 'SINT' to
type 'LWORD'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0133: Explicit narrowing conversions

Detects explicit conversions from a larger data type to a smaller data type

Justification: A large number of type conversions may indicate that you have chosen the wrong data types for variables. For this reason, there are programming guidelines that require an explicit justification for data type conversions.

Importance: Low

Example

```
PROGRAM SA0133
VAR
    siVar:SINT;
    diVar:DINT;
    liVar:LINT;
    byVar:BYTE;
    uiVar:UINT;
    dwVar:DWORD;
    lwVar:LWORD;
    rVar:REAL;
    lrVar:LREAL;
END_VAR
siVar := LINT_TO_SINT(liVar);     // SA0133
byVar := DINT_TO_BYTE(diVar);     // SA0133
siVar := DWORD_TO_SINT(dwVar);    // SA0133
uiVar := LREAL_TO_UINT(lrVar);    // SA0133
rVar := LWORD_TO_REAL(lwVar);     // SA0133

--> SA0133: Explicit narrowing conversion from type 'LINT' to type
'SINT'
--> SA0133: Explicit narrowing conversion from type 'DINT' to type
'BYTE'
--> SA0133: Explicit narrowing conversion from type 'DWORD' to type
'SINT'
--> SA0133: Explicit narrowing conversion from type 'LREAL' to type
'UINT'
--> SA0133: Explicit narrowing conversion from type 'LWORD' to type
'REAL'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0134: Explicit signed/unsigned conversions

Detects explicit conversions from signed data types to unsigned data types and the other way around

Justification: Excessive use of type conversions may indicate that you have chosen the wrong data types for variables. For this reason, there are programming guidelines that require an explicit justification for data type conversions.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    byVar :BYTE;
    udiVar : UDINT;
    uliVar : ULINT;
    lwVar : LWORD;
    wVar : WORD;
    siVar    : SINT;
    iVar    : INT;
    diVar : DINT;
    liVar    : LINT;
END_VAR
liVar := ULINT_TO_LINT(uliVar);
udiVar := DINT_TO_UDINT(diVar);
siVar := BYTE_TO_SINT(byVar);
wVar := INT_TO_WORD(iVar);
lwVar := SINT_TO_LWORD(siVar);

--> SA0134: Explicit signed/unsigned conversion from type 'ULINT'
to type 'LINT'
--> SA0134: Explicit signed/unsigned conversion from type 'DINT' to
type 'UDINT'
--> SA0134: Explicit signed/unsigned conversion from type 'BYTE' to
type 'SINT'
--> SA0134: Explicit signed/unsigned conversion from type 'INT' to
type 'WORD'
--> SA0134: Explicit signed/unsigned conversion from type 'SINT' to
type 'LWORD'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Use of Direct Addresses

SA0005: Invalid addresses and data types

Detects invalid addresses and data type specifications. Valid size prefixes in addresses: X for `BOOL` B for 1-byte data types, W for 2-byte data types, and D for 4-byte data types.

Justification: Variables located on direct addresses should preferably be associated with an address that corresponds to their data type width. It can be confusing for the reader of the code, for example, if a `DWORD` is assigned to a `BYTE` address.

Importance: Low

Example

```
PROGRAM Check_Address_Type_PRG
VAR
    iVar AT %QB0 : INT ;   // OK e. g.: %QW0
    xTest AT %QW1 : BOOL ; // OK e. g.: %QX1.0
END_VAR

iVar := iVar + INT#1;
xTest := NOT xTest;

--> SA0005: Invalid address for data type 'iVar'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0047: Accesses to direct address

Detects direct address access in the implementation code

Justification: Symbolic programming is always preferable. A variable has a name that can also have a meaning. An address cannot indicate what it is used for.

Importance: High

PLCopen rule: N1 / CP1

Example

```
PROGRAM PLC_PRG
VAR
    xVar : BOOL;
    byVar : BYTE;
END_VAR

xVar := %IX0.0;
%QX0.0 := xVar;
%MX0.1    := xVar;
%MB1    := byVar;

--> Access to direct address '%IX0.0'
--> Access to direct address '%QX0.0'
--> Access to direct address '%MX0.1'
--> Access to direct address '%MB1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0048: AT-declarations on direct addresses

Detects `AT` declarations on direct addresses

Justification: The use of direct addresses in the code is problematic because the address then appears in multiple locations: first in the controller configuration where the assignment of a physical object to an address is defined, and second in the program where variables are assigned to these addresses. If the addresses are relocated because the configuration is changed, then you have to reassign variables to addresses at a completely different location in the program. This is a cause of error and results in poorer readability and maintainability of the code. Therefore, it is best to perform all assignments in the I/O mapping of the device editor.

Importance: High

PLCopen rule: N1 / CP1

Note: We recommend that you use direct addresses ONLY in the *"I/O Mapping"* tab of the device editor.

Example

```
PROGRAM PLC_PRG
VAR
    xVar1    AT    %IX0.0    : BOOL;
    byVar1    AT    %IB1    : BYTE;
    xVar2    AT    %QX0.0    : BOOL;
END_VAR

--> SA0048: Declaration uses direct address '%IX0.0'
--> SA0048: Declaration uses direct address '%IB1
--> SA0048: Declaration uses direct address '%QX0.0'
```

See also

- *Chapter 1.8.3.1 "Configuring and Running Static Analysis" on page 4130*

Rules for Operators

SA0051: Comparison operations on BOOL variables

Detects comparison operations on variables of type `BOOL`

Justification: CODESYS permits these kinds of comparison, but they are very unusual and can be confusing. The IEC 61131-3 standard does not provide for these comparisons. By avoiding them, you increase the portability of the code to other development systems.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    xBool1, xBool2 : BOOL;
    xResult : BOOL;
END_VAR
xResult := xBool1 > xBool2;  // SA0051
xBool1 := NOT xBool1;           // OK!
xBool2 := xBool2 XOR xBool1;    // OK!

--> SA0051: Comparison operations on BOOL varables
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0052: Unusual shift operation

Detects shift operations (bit shift) on signed variables. In the case of shift operations on bitfield data types (`Byte`, `DWORD`, `LWORD`, `WORD`), an error is not reported.

Justification: CODESYS permits shift operations on signed data types. However, these operations are unusual and can be confusing. The IEC 61131-3 standard does not provide for these kinds of operations. Therefore, they should be avoided in order to increase the portability of the code to other development systems.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    iTemp : INT;
    dwTemp1 : DWORD;
    byTemp2 : BYTE;
    diTemp3 : DINT;
    siTemp4 : SINT;
    liTemp5 : LINT;
END_VAR

//the following lines each will cause an  SA0052:
iTemp := SHL(iTemp, BYTE#2);
diTemp3 := SHR(diTemp3, BYTE#4);
siTemp4 := ROL(siTemp4, BYTE#2);
liTemp5 := ROR(liTemp5, BYTE#2);

//no error SA0052 because DWORD and BYTE are bit field data types:
dwTemp1 := SHL(dwTemp1, BYTE#3);
byTemp2 := SHR(byTemp2, BYTE#1);

---> SA0052: Unusual shift operation
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0053: Too big bitwise shift

Detects whether or not the data type width of the operand has been exceeded in the case of a bitwise shift (bit shift) of operands

Justification: If a shift operation exceeds the data type width, then a constant `0` is generated. If a rotation shift exceeds the data type width, then it is difficult to read. Therefore, the rotation value should be shortened.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    byTemp1 : BYTE;
    wTemp2 : WORD;
    dwTemp3 : DWORD;
    lwTemp4 : LWORD;
END_VAR
byTemp1 := SHR(byTemp1, BYTE#25);
wTemp2 := SHL(wTemp2, BYTE#45);
dwTemp3 := ROR(dwTemp3, BYTE#78);
lwTemp4 := ROL(lwTemp4, BYTE#111);

--> SA0053: Too big bitwise shift
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0054: Comparisons of REAL/LREAL for equality / inequality

Detects whether or not the comparison operators = (equality) and <> (inequality) compare the operands of type `REAL` or `LREAL`

Justification: `REAL`/`LREAL` values are implemented as floating-point numbers according to the IEEE 754 standard. This standard implies that specific, apparently simple decimal numbers cannot be represented with precision. As a result, there may be different representations as `LREAL` for the same decimal number.

Consider the following lines of code:

```
lr11 := 1.1;
lr33 := 3.3;
lrVar1 := lr11 + lr11;
lrVar2 := lr33 - lr11;
botest := lrVar1 = lrVar2;
```

In this case, `botest` returns `FALSE`, even if the variables `lrVar1` and `lrVar2` both return the monitoring value of 2.2. This is not an error of the compiler, but a property of the floating point units of all conventional processors. You can avoid this by specifying a minimum value by which the values may differ: `botest := ABS(lrVar1 - lrVar2) < 0.1;`

Exception: A comparison with `0.0` is not reported by this analysis. For the 0, there is an exact representation in the IEEE 754 standard, and therefore the comparison functions normally as expected. Therefore, for better performance, it makes sense to permit a direct comparison here.

Importance: High

PLCopen rule: CP54

Example

```
PROGRAM PLC_PRG
VAR
    rTest1 : REAL;
    rTest2 : REAL;
    lrTest3 : LREAL;
    lrTest4 : LREAL;
    xResult : BOOL;
END_VAR

//the following lines each will cause an  SA0054:
xResult := rTest1 = rTest1;
xResult := rTest1 = rTest2;
xResult := rTest1 <> rTest2;
xResult := lrTest3 = lrTest3;
xResult := lrTest3 = lrTest4;
xResult := lrTest3 <> lrTest4;
//the following lines each will not cause an  SA0054:
xResult := rTest1 > rTest2;
xResult := lrTest3 < lrTest4;

--> SA0054: Comparisons of REAL/LREAL for equality / inequality
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0055: Unnecessary comparisons of unsigned operands

Detects unnecessary comparisons with unsigned operands. An unsigned data type is never less than zero. This can be used as a sign check.

Justification: A comparison detected with this check yields a constant result and is an indication of an error in the code.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
 byTest: BYTE;
END_VAR

WHILE byTest >= 0 DO
    byTest := byTest - 1;
END_WHILE;

--> SA0055: Unnecessary comparisons of unsigned operands
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0056: Constant out of valid range

Detects literals (constants) outside of the valid range of the operator

Justification: The message is issued in cases when a value is compared with a constant that is outside of the range of this value. Then the comparison constantly returns `TRUE` or `FALSE`. This is an indication of a programming error.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    byTestVar: BYTE;
END_VAR

WHILE byTestVar >= 260 DO
 byTestVar := byTestVar + 1;
END_WHILE

--> SA0056: Constant out of valid range
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0057: Possible loss of decimal places

Detects statements with possible loss of decimal places

Justification: A piece of code of the following type (`diTemp2 := 1 rTemp1 := TO_REAL(diTemp2 / DINT#2)`) can cause a misinterpretation. The author or reader of this line of code can assume that the division would be performed as a `REAL` operation, and in this case the result would be `REAL#0.5`. However, this is not true. It is an integer operation. The result is cast to `REAL` and `rTemp1` gets the value `REAL#0`. To avoid this, use a cast to make sure that the operation is performed as a `REAL` operation: `rTemp1 := TO_REAL(diTemp2) / REAL#2`.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    rTemp1 : REAL;
    diTemp2 : DINT;
    liTemp3 : LINT;
END_VAR

diTemp2 := diTemp2 + DINT#11;
rTemp1 := DINT_TO_REAL(diTemp2 / DINT#3);    // SA0057
rTemp1 := DINT_TO_REAL(diTemp2) / REAL#3.0;
liTemp3 := liTemp3 + LINT#13;
rTemp1 := LINT_TO_REAL(liTemp3 / LINT#7);    // SA0057
rTemp1 := LINT_TO_REAL(liTemp3) / REAL#7.0;


--> SA0057: Possible loss of decimal places
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0058: Operations on enumeration variables

Detects operations on variables of the enumeration data type Assignments are permitted.

Justification: Enumerations should not be used as ordinary integer values. You can also define an alias data type or use a subrange type.

Importance: Medium

Exception: If an enumeration is tagged with the pragma `{attribute 'strict'}`, then the compiler already reports this kind of operation.

If an enumeration is declared as a flag by the pragma `{attribute 'flags'}`, then an error is not issued for `AND`, `OR`, `NOT`, or oder `XOR` operations.

Example

```
TYPE My_Enum :
(
    red := 1, blue := 2, green := 3, black := 4
);
END_TYPE

PROGRAM PLC_PRG
VAR
    iTemp1 : INT;
    abc : My_Enum;
END_VAR
iTemp1 := iTemp1 + INT#1;
abc := My_Enum.red;                         // OK
iTemp1 := My_Enum.black / My_Enum.blue;  //  SA0058
iTemp1 := My_Enum.green / My_Enum.red;   //  SA0058

--> SA0058: Operations on enumeration variables
```

Example with a pragma {attribute 'flags'}

```
{attribute 'flags'}  // declaring the enumeration as a "flag"
TYPE Flags :
(
    Unknown := 16#00000001,
    Stopped := 16#00000002,
    Running := 16#00000004
) DWORD;
END_TYPE

PROGRAM PLC_PRG
VAR
 iTemp1 : INT;
 abc : Flags;
    batate : BYTE;
    dwFlags : DWORD;
    dwState : DWORD;
END_VAR

// OK for the following
IF (dwFlags AND Flags.Unknown) <> DWORD#0 THEN
    dwState := dwState AND Flags.Unknown;
ELSIF (dwFlags OR Flags.Stopped) <>DWORD#0 THEN
    dwState := dwState OR Flags.Running;
END_IF
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0059: Comparison operations always returning TRUE or FALSE

Detects comparisons with literals that always have the result `TRUE` or `FALSE`, and can already be processed during at the compile.

Justification: An operation that consistently yields `TRUE` or `FALSE` is an indication of a programming error.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    byTemp1 : BYTE;
END_VAR

WHILE byTemp1 <= 255 DO
    byTemp1 := byTemp1 + 1;
END_WHILE;

--> SA0059: Relational operator '<=' always evaluates 'TRUE'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0060: Zero used as invalid operand

Detects operations where an operand with the value "0" causes an invalid or a nonsense operation

Justification: This kind of expression could be an indication of a programming error. In any case, it unnecessarily wastes runtime.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    byTemp1 : BYTE;
    wTemp2 : WORD;
    dwTemp3 : DWORD;
END_VAR

byTemp1 := byTemp1 + 0;
wTemp2 := wTemp2 - WORD#0;
dwTemp3 := dwTemp3 * DWORD#0;

--> SA0060: Zero used as invalid operand
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0061: Unusual operation on pointer

Detects operations one variables of type `POINTER TO` which are not = (equality), <> (inequality), + (addition), or `ADR`.

In CODESYS, pointer arithmetic is generally permitted and can also be used appropriately. Therefore, the addition of a pointer with an integer value is considered a common operation on pointers. This makes it possible to use a pointer to process an array of variable length. All other (unusual) operations with pointers are reported with SA0061.

Importance: High

PLCopen rule: E2 / E3

Example

```
PROGRAM PLC_PRG
VAR
    piTemp : POINTER TO INT;
    iTemp : INT;
END_VAR

iTemp := iTemp + INT#1;
piTemp := ADR(iTemp);
piTemp := piTemp * DWORD#5;   // SA0061
piTemp := piTemp / DWORD#2;   // SA0061
piTemp := piTemp MOD DWORD#3; // SA0061
piTemp := piTemp + DWORD#1;
piTemp := piTemp - DWORD#1;   // SA0061

--> SA0061: Unusual operation on pointer
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0062: Uses of TRUE or FALSE in expressions

Detects the use of the literals `TRUE` or `FALSE` in expressions

Justification: This kind of expression is obviously unnecessary and may indicate an error. In any case, the expression unnecessarily affects the runtime.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    xTemp1, xTemp2 : BOOL;
END_VAR
xTemp1 := xTemp1 AND NOT TRUE;
xTemp2 := xTemp1 OR TRUE;
xTemp2 := xTemp1 OR NOT FALSE;
xTemp2 := xTemp1 AND FALSE;

--> Uses of TRUE or FALSE in expressions
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0063: Possibly not 16-bit-compatible operations

Detects 16-bit operations with temporary results. Background: On 16-bit systems, 32-bit temporary results can be truncated. Example: `(int+10)` can exceed 16 bits.

Justification: In the very rare case that you have to write code which should run on a 16-bit processor as well as on a 32-bit processor, this message should help to prevent any problems.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    iVar : INT;
END_VAR
iVar := (iVar + 10) / 2;

--> SA0063: Compatibility for 16 Bit - Possible truncated
intermediate result
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0064: Addition of pointer

Detects the addition of pointers

Justification: In CODESYS, pointer arithmetic is generally permitted and can also be used appropriately. However, it is also a source of errors. Therefore, programming rules exist that generally prohibit pointer arithmetic. This test can check such a requirement.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    iTest:INT;
    ariTest:ARRAY[0..10] OF INT;
    {attribute 'analysis':='-111'}
    piTest:POINTER TO INT;
    i:INT;
END_VAR

piTest := ADR(ariTest[0]);            // OK
piTest^:= 0;
piTest := ADR(ariTest) + SIZEOF(INT); // SA0064
piTest^:= 1;
piTest := ADR(ariTest) + 6;           // SA0064
piTest^:= 3;
piTest := ADR(ariTest[10]);
FOR i:=0 TO 10 DO
    piTest^ := i;
    piTest := piTest + 2;                // SA0064
END_FOR

--> SA0064: Addition of pointer
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0065: Incorrect pointer addition to base size

Detects pointer additions for which the value to be added does not match the base size of the pointer. Only literals of the base size can be added. Also multiplication products of the base size cannot be added.

Justification: In CODESYS (in contrast to C and C++), when adding a pointer with an integer value, only this integer value is added as the number of bytes, and not the integer value multiplied by the base size. Example in ST:

```
pINT := ADR(array_of_int[0])
pINT := pINT + 2 ;   // In CODESYS, pINT then points to
array_of_int[1]
```

This code would function differently in C:

```
short* pShort
pShort = &(array_of_short[0])
pShort = pShort + 2;  // In C, pShort then points to array_of_short[2]
```

Therefore, in CODESYS, you should always add a multiple of the base size of the pointer to a pointer. Otherwise, the pointer may point to non-aligned memory which (depending on the processor) can lead to an alignment exception when accessing it.

Importance: High

Example

```
VAR
 pudiTest:POINTER TO UDINT;
    udiTest:UDINT;
    prTest:POINTER TO REAL;
    rTest:REAL;
END_VAR

pudiTest := ADR(udiTest) + 4;                  // OK
pudiTest := ADR(udiTest) + ( 2 + 2 );          // OK
pudiTest := ADR(udiTest) + SIZEOF(UDINT);      // OK
pudiTest := ADR(udiTest) + 3;                // SA0065
pudiTest := ADR(udiTest) + 2*SIZEOF(UDINT);    // SA0065
pudiTest := ADR(udiTest) + ( 3 + 2 );          // SA0065
prTest := ADR(rTest);
prTest := prTest + 4;                          // OK
prTest := prTest + ( 2 + 2 );                  // OK
prTest := prTest + SIZEOF(REAL);               // OK
prTest := prTest + 1;                          // SA0065
prTest := prTest + 2;                          // SA0065
prTest := prTest + 3;                          // SA0065
prTest := prTest + ( SIZEOF(REAL) - 1 );       // SA0065
prTest := prTest + ( 1 + 4 );                  // SA0065

--> SA0065: Incorrect pointer addition to base size
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0066: Uses of temporary results

Detects the use of temporary results in statements with a data type that is less than the registry size. The implicit cast in this case may lead to unwanted results.

Justification: For performance reasons, CODESYS performs operations on the register width of the processor. Intermediate results are not truncated. This can lead to misinterpretations as in the following case: `usintTest := 0; xError := usintTest - 1 <> 255;`. In CODESYS, `xError` is `TRUE` in this case because the operation `usintTest - 1` is typically executed as a 32-bit operation and the result is not cast to the byte size. Then the value `16#ffffffff` (not equal to 255) is located in the registry. To avoid this, you have to cast the intermediate result explicitly: `xError := TO_USINT(usintTest - 1) <> 255;`

NOTICE!

If this message is activated, then many less problematic locations in the code will be reported. Although a problem can only occur when the operation produces an overflow or underflow in the data type, the static analysis cannot differentiate between the individual locations.

If you include an explicit typecast in all reported locations, then the code will be much slower and less readable.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    byTest:BYTE;
    liTest:LINT;
    xError:BOOL;
END_VAR

//type size smaller than register size;
byTest := 0;
IF (byTest - 1) <> 255 THEN //use of temporary result + implicit
casting ->  SA0066
    xError := TRUE;
ELSE
    xError := FALSE;
END_IF

//type size equal to or bigger than register size;
liTest := 0;
IF (liTest - 1)  <> -1 THEN // use of temporary result and no
implicit casting -> OK
    xError := TRUE;
ELSE
    xError := FALSE;
END_IF

--> SA0066: Use of temporary result: (byTest - USINT #1)
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

Rules for Statements

SA0072: Invalid uses of counter variable

Detects the use of a counter variable in a `FOR` loop

Justification: Manipulation of the counter variable in a `FOR` loop can easily result in an infinite loop. To prevent the execution of the loop for specific values of the counter variable, use `CONTINUE` or simply an `IF`.

Importance: High

PLCopen rule: L12

Example

```
PROGRAM PLC_PRG
VAR_TEMP
    iIndex : INT;
END_VAR
FOR iIndex := INT#0 TO INT#20 BY INT#1 DO
    iIndex := iIndex - INT#1;
END_FOR

--> SA0072: Invalid use of counter variable 'iIndex'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0073: Uses of inadequate counter variable

Detects the use of non-temporary variables in `FOR` loops.

Justification: This is a performance warning. A counter variable is always initialized each time a POU is called. You can create this variable as a temporary variable (`VAR_TEMP`). Access to it may be faster and the variable does not take up any permanent memory.

Importance: Medium

PLCopen rule: CP21 / L13

Example

```
PROGRAM PLC_PRG
VAR
    nIndex : INT;
    iVar : INT;
END_VAR
FOR nIndex := INT#0 TO INT#20 BY INT#1 DO
    iVar := iVar + nIndex;
END_FOR

--> SA0073: Inadequate counter variable
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0080: Loop index variable for array index exeeds array range

Detects the FOR statements where the index variable is used to access an array index and exceeds the range of the array index

Justification: Arrays are typically processed in `FOR` loops. The start and end value of the counter variable should typically match (or at least not exceed) the upper and lower bounds of the array. A typical cause of error is detected here when array bounds are changed and constants are not carefully used; or when a different value is used accidentally in the `FOR` loop than in the array declaration.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
 iIndex1,iIndex2,iIndex3 : INT;
 arWord : ARRAY[1..100] OF WORD;
 arararINT : ARRAY[1..9,1..9,1..9] OF INT;
 arUSINT : ARRAY[0..99] OF USINT;
END_VAR

//1 violation of the rule(lower range is exeeded): SA0080
FOR iIndex1 := INT#0 TO INT#100 DO
    arWord[iIndex1] := INT_TO_WORD(iIndex1);
END_FOR

//6 violations (lower and upper range is exceeded for each array
dimension): 3SA0080
FOR iIndex2 := INT#0 TO INT#10 DO
    arararINT[iIndex2, iIndex2, iIndex2] := iIndex2;
END_FOR

//1 violation (upper range is exceeded by the end result of the
index), previous expressions on index are not evaluated -> OK
FOR iIndex3 := INT#0 TO INT#50 DO
    arUSINT[iIndex3 * INT#2] := INT_TO_USINT(iIndex3);
END_FOR

--> SA0080: Loop index range of 'Index1' exceeds array range
--> SA0080: Loop index range of 'Index2' exceeds array range
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0081: Upper border is not a constant

Detects the FOR statements where the upper bound is not defined with a constant value

Justification: If the upper bound of a loop is a variable value, then it is no longer possible to see how often a loop is executed. This can result in serious problems at runtime. The worst case is an infinite loop.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    i:INT;
    iBorder1: INT := 10;
    iBorder2: INT := 10;
    iCounter: INT;
END_VAR
VAR CONSTANT
    ciBorder:INT := 10;
END_VAR

FOR i:=0 TO 10 DO    //OK
    iCounter := i;
END_FOR

FOR i:=0 TO ciBorder DO // OK
    iCounter := i;
END_FOR

FOR i:=0 TO iBorder1 DO    // SA0081
    iCounter := i;
END_FOR

FOR i:=0 TO iBorder2 DO    // SA0081
    iCounter := i;
    IF iCounter = 10 THEN
        iBorder2 := 50;
    END_IF
END_FOR

--> SA0081: Upper border of a for loop must be a constant value
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0075: Missing ELSE

Detects `CASE` statements without an `ELSE` branch

Justification: Defensive programming requires the inclusion of an `ELSE` branch in every `CASE` statement. If there is nothing to do in the `ELSE` branch, then include a comment to indicate this. It is then clear to the reader of the code that the case was not simply forgotten.

Importance: Low

PLCopen rule: L17

Example

```
PROGRAM PLC_PRG
VAR
    iVar : INT;
    xTemp : BOOL;
END_VAR

iVar := iVar + INT#1;
CASE iVar OF
    INT#1:
        xTemp := FALSE;
    INT#2:
        xTemp := TRUE;
END_CASE

--> SA0075: Missing ELSE in CASE statement
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0076: Missing enumeration constant

Detects whether or not an enumeration variable is used as a condition and not all enumeration values are treated as `CASE` branches

Justification: Defensive programming requires the processing of all possible values of an enumeration. If an action is not required for a particular enumeration value, then you should add a comment to indicate this explicitly. It is then clear to the reader of the code that the value was not simply forgotten.

Importance: Low

Example

```
TYPE My_Enum :
(
    red := 1, blue := 2, green := 3, black := 4
);
END_TYPE

PROGRAM PLC_PRG
VAR
    iVar : My_Enum;
    xTemp : BOOL;
END_VAR
iVar := My_Enum.black;

CASE iVar OF
    My_Enum.red:
        xTemp := FALSE;
    My_Enum.blue, My_Enum.green:
        xTemp := TRUE;
    ELSE
        xTemp := NOT xTemp;
END_CASE

--> SA0076: Missing enumeration constant 'black' in CASE statement
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0077: Type mismatches with CASE expression

Detects code locations where the data type of a condition does not match that of the `CASE` branch

Justification: If the data types between the `CASE` variable and the `CASE` itself do not match, then this could indicate an error.

Importance: Low

Example

```
TYPE My_Enum :
(
    eins := 1, zwei := 2, drei := 3, vier := 4
);
END_TYPE

PROGRAM PLC_PRG
VAR
    diVar : DINT;
    xTemp : BOOL;
END_VAR
diVar := diVar + DINT#1;
CASE diVar OF
    DINT#1:
        xTemp := FALSE;
    My_Enum.zwei, DINT#3:    //SA0077
        xTemp := TRUE;
    ELSE
        xTemp := NOT xTemp;
END_CASE

--> SA0077: Type mismatches with CASE expression
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0078: Missing CASE branches

Detects `CASE` statements without `CASE` branches and only one `ELSE` statement

Justification: A `CASE` statement without cases wastes execution time and it is difficult to read.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    iVar : INT;
    xTemp : BOOL;
END_VAR

iVar := iVar + INT#1;
//in the following the case descriptions are missing:
CASE iVar OF
    ELSE
        xTemp := NOT xTemp;
END_CASE

--> SA0078: CASE-Missing CASE branches
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0090: Return statement before end of function

Detects whether or not the `RETURN` statement is not the last statement in a function, method, property, or program.

Justification: A `RETURN` in the code results in worse maintainability, testability, and readability of the code. A `RETURN` in the code is easily overlooked. Before each `RETURN`, it is often forgotten to insert code that should always be executed when exiting a function.

Importance: Medium

PLCopen rule: CP14

Example

```
FUNCTION FUN : DINT
VAR_INPUT
    bTest : BOOL;
END_VAR

IF bTest THEN
    RETURN;
END_IF
FUN := 99;

--> SA0090: Return statement before end of function
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0095: Assignments in conditions

Detects assignments in conditions of `IF`, `CASE`, or `REPEAT` constructs

Justification: An assignment (:=) and a comparison (=) can easily be mistaken. As a result, an assignment in a condition can easily be unintentional, and it is therefore reported. This can also confuse the reader of the code.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    iCond1:INT := INT#1;
    iCond2:INT := INT#2;
    xCond:BOOL := FALSE;
    iVar : INT;
END_VAR

IF INT_TO_BOOL(iCond1 := iCond2) THEN                       // SA0095
    iCond1 := INT#1;
    iCond2 := INT#2;
ELSIF (iCond1 := 11) = 11 THEN                              // SA0095
    iCond1 := INT#1;
    iCond2 := INT#2;
END_IF

IF xCond := TRUE THEN                                       // SA0095
    xCond := FALSE;
END_IF

IF (xCond := FALSE) OR (iCond1 := iCond2) = 12 THEN         // SA0095
    xCond := FALSE;
    iCond1 := INT#1;
    iCond2 := INT#2;
END_IF

IF (iVar := iVar + 1) = 120 THEN                            //
SA0095 (can be valid, but is not reparable very well
iVar := 0;
END_IF

WHILE (xCond = TRUE) OR (iCond1 := iCond2) = 12 DO          // SA0095
    xCond := FALSE;
END_WHILE

//Error: assignment in repeat loop
REPEAT
    xCond := FALSE;
UNTIL
    (xCond = TRUE) OR (iCond1 := iCond2) = 12               //
SA0095
END_REPEAT

--> SA0095: Assignment in condition: '...'
```

See also

- ↳ *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0100: Variables greater than <n> bytes...

Detects variables that use more than n bytes, where n is defined by the current configuration. Default value: 1024 bytes. The value can be changed by double-clicking the line.

Justification: Some programming guidelines specify a maximum size for a single variable. This can be checked with this.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    aobyTest : ARRAY [0..1024] OF BYTE;
END_VAR

aobyTest[INT#0] := aobyTest[INT#0] + BYTE#1;

--> SA0100: Variable 'aobyTest' greater 1024 bytes
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0101: Names with invalid length

Detects names with invalid lengths. In the *“Project Settings”*, double-click the rule entry to open a dialog where you can define the length of the name and define any exception.

Justification: Some programming guidelines specify a minimum length for variable names. This analysis can be used to check compliance.

Importance: Low

PLCopen rule: N6

Example

```
PROGRAM PLC1                    // SA0101
VAR
    iVar1: INT;                 // SA0101
END_VAR

--> SA0101: Incorrect length of name 'PLC1'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0102: Access to program/fb variables from the outside

Detects external access to local variables of programs or function blocks

Justification: CODESYS permits external read access to local variables of programs or function blocks. This contradicts the principle of data encapsulation (hiding data) and does not comply with the IEC 61131-3 standard.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    iCounter : INT;
    afb_Instance : AFB;
    bfb_Instance : BFB;
END_VAR
iCounter := A_PRG.iLocal;               //  SA0102
iCounter := bfb_Instance.iLocal;        //  SA0102
A_PRG();

FUNCTION_BLOCK AFB
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iLocal: INT;
END_VAR
METHOD METH : INT
VAR_INPUT
END_VAR
iLocal := iLocal + 1;

FUNCTION_BLOCK BFB EXTENDS AFB
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
METHOD METH : INT
VAR_INPUT
END_VAR
iLocal := iLocal + 1;

PROGRAM A_PRG
VAR
    iLocal: INT;
END_VAR
iLocal := iLocal + 1;

--> SA0102: Access to program/fb variable 'iLocal' from the outside
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0103: Concurrent access on not atomic data

Detects whether or not non-atomic variables (for example, with data type `STRING`, `WSTRING`, `ARRAY`, `STRUCT`, FB instances, 64-bit data types) are used in more than one task

Justification: When there is no synchronization during access, inconsistent values can be read when reading in one task and writing in another task at the same time.

Importance: Medium

> *For some data types, especially 64-bit integers, it depends on the platform whether or not access is atomic. Static analysis reports a problem only when the controller does not support atomic access to 64-bit integer data types.*

This rule does not apply in the following cases:

- If the target system has a floating point unit (FPU), then access of multiple tasks to `LREAL` variables is not detected
- If the target system is a 64-bit processor or the corresponding target setting is set for the target device, then the rule does not apply to 64-bit data types

Example

The project contains both programs, `PRG1` and `PRG2`: The program `PRG1` is called by the task `MainTask_1`. The program `PRG2` is called by the task `MainTask_2`.

```
GVL
VAR_GLOBAL
    lrTest : LREAL;   // Since the target system has an FPU, SA0103
does apply.
    lint1 : LINT;
    sTest : STRING;   // SA0103
    wsTest : WSTRING; // SA0103
END_VAR

PROGRAM PRG1
GVL.lrTest := 5.0;
GVL.sTest := 'welt';
GVL.wsTest := "welt";
GVL.lint1 := 99;

PROGRAM PRG2
GVL.lrTest := 5.0;
GVL.sTest := 'hallo';
GVL.wsTest := "hallo";
GVL.lint1 := 88;

--> SA0103: Concurrent access on not atomic data 'sTest'
--> SA0103: Concurrent access on not atomic data 'wsTest'
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0105: Multiple instance calls

Detects the instances of function blocks that are called multiple times. To do this, the function blocks haves to be marked with the pragma `{attribute 'analysis:report-multiple-instance-calls'}`.

Justification: Some function blocks are designed in such as way that they can be called only one time in the cycle. This test checks whether or not a call is made in multiple locations.

Importance: Low

PLCopen rule: CP16 / CP20

Example

```
// {attribute 'analysis:report-multiple-instance-calls'} Deactivated
FUNCTION_BLOCK FB_DoA
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iA : INT;
END_VAR
iA := iA + 1;

{attribute 'analysis:report-multiple-instance-calls'}
FUNCTION_BLOCK FB_DoB
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iB : INT;
END_VAR
iB := iB +1;

PROGRAM PLC_PRG
VAR
    fbA : FB_DoA;
    fbB : FB_DoB;
ND_VAR

fbA();
fbB();    // SA0105
fbA();
fbB();    // SA0105

--> SA0105: Instance 'fbB' called more than once
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0106: Virtual method calls in FB_INIT

Detects method calls in the `FB_Init` method of a base function block, which are overwritten by a function block derived from a base function block

Justification: In these cases, it could be that the variables in the overwritten methods are not initialized in the base FB.

Importance: High

Example

The function block `FB_A` includes the methods `FB_Init` and `Meth_MyInit`. `FB_Init` calls `Meth_MyInit` for initialization. The function block `FB_B` is derived from `FB_A`. `PLC_PRG` calls `FB_B` and therefore uses its `mbMyDintB` variable before it has been initialized. `FB_B.Meth_MyInit` overwrites `FB_A.Meth_MyInit`.

```
FUNCTION_BLOCK FB_A
VAR
    mbMyDintA : DINT;
END_VAR FUNCTION_BLOCK FB_B EXTENDS FB_A
VAR
    mbMyDintB : DINT;
END_VAR METHOD FB_Init : BOOL
VAR_INPUT
    bInitRetains:BOOL;
    bInCopyCode:BOOL;
END_VAR
VAR
    diDummy:DINT;            // SA0106
END_VAR
mbMyDintA := 123;
diDummy := Meth_MyInit(); METHOD Meth_MyInit : DINT
VAR_INPUT
END_VAR
mbMyDintB := 123; // access to member of FB_B PROGRAM PLC_PRG
VAR
    g_BInst : FB_B;
 xVar : BOOL;
END_VAR
xVar := g_BInst.fb_init(TRUE, TRUE);
//this instruction causes the following order of initializations:
//FB_A.fb_init
//FB_B.Meth_MyInit // SA0106
//FB_B.fb_init
//FB_B.Meth_MyInit

--> SA0106: Virtual method call 'Meth_MyInit' in FB_INIT
```

See also

- ↳ *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0107: Missing formal parameters

Detects whether or not formal parameters are missing

Justification: Code becomes more readable when formal parameters are specified in the call.

Importance: Low

Example

```
FUNCTION FUNA : BOOL
VAR_INPUT
    bDo: BOOL;
    bInit: BOOL;
    bManual : BOOL;
END_VAR
VAR
    iInit: INT;
    iLocal: INT;
    iManual: INT;
END_VAR

IF bInit = TRUE THEN
    iInit := iInit + 1;
END_IF
IF bDo = TRUE THEN
    iLocal := iLocal + 1;
END_IF
IF bManual = TRUE THEN
    iManual:= iManual + 1;
END_IF
FUNA := TRUE;

PROGRAM PLC_PRG
VAR
END_VAR

FUNA(bInit := TRUE, bDo := TRUE, bManual := FALSE);     // OK
FUNA(TRUE, TRUE, bManual:= FALSE);                      // SA0107

--> SA0107: Missing formal parameter for input 'TRUE'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

Checking Strict IEC Rules

SA0111: Pointer variables

Detects variables of type `POINTER TO`

Justification: The IEC 61131-3 standard does not permit pointers.

Importance: Low

Example

```
VAR
    piTemp : POINTER TO INT;
    pbyTemp : POINTER TO BYTE;
END_VAR

--> SA0111: Data type POINTER not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0112: Reference variables

Detects variables of type `REFERENCE TO`

Justification: The IEC 61131-3 standard does not permit references.

Importance: Low

Example

```
VAR
 ref_int : REFERENCE TO INT;
 ref_dw : REFERENCE TO DWORD;
END_VAR

--> Data type REFERENCE not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0113: Variables with data type WSTRING

Detects variables of type `WSTRING`

Justification: Not all systems support `WSTRING`. The code is more easily portable without `WSTRING`.

Example

```
VAR
    wstrTemp : WSTRING;
END_VAR

--> SA0113: Data type WSTRING not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0114: Variables with data type LTIME

Detects variables of type `LTIME`.

Justification: Not all systems support `LTIME` The code is more easily portable without `LTIME`.

Importance: Low

Example

```
VAR
    ltVar : LTIME; // SA0114
END_VAR

--> SA0114: Data type LTIME not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0115: Variables with data type UNION

Detects declarations of a `UNION` data type and variable declarations of the `UNION` type

Justification: The IEC 61131-3 standard does not include unions. The code is more easily portable without unions.

Importance: Low

Example

```
TYPE u1: UNION
 lrTemp : LREAL;
 liTemp : LINT;
END_UNION
END_TYPE

PROGRAM PLC_PRG
VAR
    uVar: u1;
END_VAR

--> SA0115: Unions not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0117: Variables with data type BIT

Detects variable declarations of data type BIT (possible within structure definitions)

Justification: The IEC 61131-3 standard does not include the data type `BIT`. The code is more easily portable without `BIT`.

Importance: Low

Example

```
TYPE Struct1 :
STRUCT
    bitVar : BIT;
    iVar : INT;
    bVar : BOOL;
END_STRUCT
END_TYPE

--> SA0117: Variables with data type BIT
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0119:Object-oriented features

Detects the use of object-oriented features, such as function block declarations with `EXTENDS` and `IMPLEMENTS`, or property and interface declarations. This rule is useful when you write code that is intended to be ported to other IEC 61131-3-compliant systems.

Justification: Not all systems support object-oriented programming. The code is more easily portable without object-orientation.

Importance: Low

Example

```
//Function block extended by another and implementing an interface:
FUNCTION_BLOCK POU EXTENDS CTD IMPLEMENTS ITF        //SA0119
...

// Declaration parts of property methods assigned to a function
block:
POU.Prop.Get        //SA0119
POU.Prop.Set        //SA0119

--> SA0119: Object-oriented features not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0120: Program calls

Detects program calls

Justification: According to the IEC 61131-3 standard, programs can be called in the task configuration only. The code is more easily portable when you do not call programs from other locations.

Importance: Low

Example

```
PROGRAM prog_control
VAR
END_VAR

PROGRAM PLC_PRG
VAR
END_VAR

prog_control();

--> SA0120: Program call to 'prg_control' not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0121: Missing VAR_EXTERNAL declarations

Detects the use of a global variable in function blocks without them being declared there as *“VAR_EXTERNAL”*

Justification: According to the IEC 61131-3 standard, access to global variables is permitted only by an explicit import by means of a `VAR_EXTERNAL` declaration.

Importance: Low

PLCopen rule: CP18

Example

```
VAR_GLOBAL
    iGlob1:INT;
END_VAR

PROGRAM PLC_PRG
VAR
 ivar:INT;
END_VAR

ivar:=iGlob1;        // SA0121

--> SA0121: EXTERNAL declaration required for variable ''iGlob1'
```

Example:
Avoid error

```
VAR_GLOBAL
    iGlob1:INT;
END_VAR

PROGRAM PLC_PRG
VAR
 ivar:INT;
END_VAR
VAR_EXTERNAL
 iGlob1:INT;
END_VAR

ivar:=iGlob1;        // OK
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0122: Array index defined as expression

Detects the use of expressions in the declaration of array indexes

Justification: Not all systems permit expressions as array limits.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR CONSTANT
 c_iValue : INT := INT#15;
END_VAR
VAR
 arr: ARRAY[0..c_iValue + 1] OF INT;
END_VAR

--> SA0122: Only constants allowed for array definition 'arr'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0123: Usages of INI, ADR or BITADR

Detects the use of the CODESYS-specific operators `INI`, `ADR`, and `BITADR`

Justification: CODESYS-specific operators prevent the portability of code.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    uiTemp: UINT;
    TempVarInFUNC: DWORD;
END_VAR

TempVarInFUNC := ADR(uiTemp);        //SA0123

--> SA0123: Operator 'ADR' not allowed
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0147: Unusual shift operation - strict

Detects bit shift operations that are not made to bitfield data types (`BYTE`, `WORD`, `DWORD`, `LWORD`)

Justification: The IEC 61131-3 standard permits bit access only to bitfield data types. However, the CODESYS compiler also permits bit shift operations with unsigned data types.

Importance: Low

See also the strict rule SA0018.

Example

```
PROGRAM PLC_PRG
VAR
  in_byte : BYTE := 16#45;   // 2#01000101
  in_word : WORD := 16#0045; // 2#0000000001000101
  in_uint : UINT;
  in_dint : DINT;
  erg_byte : BYTE;
  erg_word : WORD;
  erg_uint : UINT;
  erg_dint : DINT;
  n: BYTE := 2;
END_VAR

erg_byte := SHL(in_byte,n);  // no error because BYTE is a bit field
erg_word := SHL(in_word,n);  // no error because WORD is a bit field
erg_uint := SHL(in_uint,n);  // SA0147
erg_dint := SHL(in_dint,n);  // SA0147

--> SA0147: Unusual shift operation - strict
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*
- *Chapter 1.8.3.3.2.17 “SA0018: Unusual bit access” on page 4164*

SA0148: Unusual bit access - strict

Detects bit access that is not made to bitfield data types (`BYTE`, `WORD`, `DWORD`, and `LWORD`). The IEC 61131-3 standard permits only bit access to bitfield data types. However, the CODESYS compiler also permits bit access to unsigned data types.

Example

```
PROGRAM PLC_PRG
VAR
    iTemp1 : INT;
    diTemp3 : DINT;
    uliTemp4 : ULINT;
    siTemp5 : SINT;
    usiTemp6 : USINT;
    byTemp2 : BYTE;
END_VAR

iTemp1.3 := TRUE;     // SA0148
diTemp3.4 := TRUE;    // SA0148
uliTemp4.18 := FALSE; // SA0148
siTemp5.2 := FALSE;   // SA0148
usiTemp6.3 := TRUE;   // SA0148
byTemp2.5 := FALSE;   // no error because BYTE is a bit field

--> SA0148: Unusual bit access - strict
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0118: Initialisations not using constants

Detects initializations that do not assign constants

Justification: Initializations should be constant if possible and should not refer to other variables. In particular, you should avoid function calls during initialization because this can allow access to uninitialized data.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    dwTemp : DWORD := 22;
    dwTest : DWORD := dwTemp;          //  SA0118
    dwVar : DWORD := TempVarInFUNC(); //  SA0118
END_VAR

--> SA0118: Initialisations not using constants
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0124: Pointer dereferences in declarations

Detects pointer dereferences that are used for initialization in the declaration part

Justification: Pointers and references should not be used for initializations because this can lead to access violations if the pointer has not been initialized.

Importance: Medium

Example

```
FUNCTION_BLOCK FB_Test
VAR_INPUT
    refStruct: REFERENCE TO ST_Test;
END_VAR
VAR
    xPointer : BOOL := refStruct.a; // SA0124
    iCount : INT;
END_VAR

--> SA0124: Dereference access in initialisation
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0125: References in initializations

Detects reference variables that are used for initialization in the declaration part

Justification: Pointers and references should not be used for initializations because this can lead to access violations if the pointer has not been initialized.

Importance: Medium

Example

```
PROGRAM PLC_PRG
VAR
    xRef: REFERENCE TO INT;
    iCount: INT := xRef;
END_VAR

--> SA0125: Reference used in initializations
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0140: Statements commented out

Detects commented-out statements

Justification: Code is often commented out for debugging purposes. When this kind of comment is released, it is not always clear at a later time whether the code should be deleted, or whether it has been commented out for debugging purposes and unintentionally not uncommented.

Importance: High

PLCopen rule: C4

Example

```
PROGRAM PLC_PRG
VAR
    iValue1: INT;
    iValue2: INT;
END_VAR

iValue1 := 100;
iValue2 := 200;
// iValue2 := 300;

--> SA0140: Statement commented out:: iValue2 := 300
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

Possible Use of Uninitialized Variables

SA0039: Possible null-pointer deferences

Detects code locations where a null pointer is possibly dereferenced

Justification: A pointer should be checked before each dereferencing to make sure it is not equal to zero. Otherwise an access violation may occur at runtime.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR
    ptiVar1:POINTER TO INT;
    ptiVar2:POINTER TO INT;
    ptiVar3:POINTER TO INT;
    iVar:INT;
    iCount :INT;
    iCondition: INT;
END_VAR

iCount := iCount + INT#1;
ptiVar1 := ADR(iVar);
ptiVar1^ := iCondition; // OK - valid reference
ptiVar2^ := iCondition; // SA0039 - null pointer dereferenciation
iVar := ptiVar3^;       // SA0039 - null pointer dereferenciation

--> SA0039: Possible null pointer dereference 'ptiVar2^'
--> SA0039: Possible null pointer dereference 'ptiVar3^'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0046: Possible use of not initialised interface

Detects the use of interfaces that were not initialized before being used

Justification: An interface reference should be checked for <> 0 before it is used. Otherwise an access violation may occur during access.

Importance: High

Example

```
//declaration of INTERFACE ITF and assigned METH2:
METHOD METH2 : BOOL
VAR_INPUT
    iInput2:INT;
END_VAR
//declaration of INTERFACE Master_ITF1 and assigned METH:
METHOD METH : BOOL
VAR_INPUT
    iInput:INT;
END_VAR

PROGRAM PLC_PRG
VAR
    instPOU:POU;
    instITF:ITF;
    instMasterITF1:Master_ITF1;
    instMasterITF2:Master_ITF2;
    iDummy:INT;
    xDummy:BOOL;
    instNoInitITF:ITF;
    instNoInitITF2:ITF;
    instNoInitMasterITF1:Master_ITF1;
    instNoInitMasterITF2:Master_ITF2;
END_VAR

instITF := instPOU;
xDummy := instITF.METH(iInput := iDummy);             // OK
instMasterITF1 := instPOU;
xDummy := instMasterITF1.METH(iInput := iDummy);      // OK

xDummy := instNoInitITF.METH(iInput := INT#1);        // SA0046
xDummy := instNoInitITF.METH2(iInput2 := INT#2);      // SA0046
xDummy := instNoInitMasterITF1.METH(iInput := INT#3); // SA0046
iDummy := instNoInitMasterITF2.Prop;                  // SA0046

IF instNoInitITF <> 0 THEN
    instNoInitITF.Prop;          // OK, weil das Interface nicht 0
sein kann
END_IF


--> SA0046: Possible use of not initialised interface
'instNoInitITF'
--> SA0046: Possible use of not initialised interface
'instNoInitITF'
--> SA0046: Possible use of not initialised interface
'instNoInitMasterITF1'
--> SA0046: Possible use of not initialised interface
'instNoInitMasterITF2
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0145: Possible use of not initialised reference

Detects any reference variables that may not be initialized before use and are not checked by the operator `__ISVALIDREF`. This rule is applied in the implementation part of POUs. Rule SA0124 applies to the declaration.

Justification: A reference should be checked for its validity before access because an access violation may occur during access.

Importance: High

Example

```
PROGRAM PLC_PRG
VAR_INPUT
    ref_iTest : REFERENCE TO INT;
END_VAR

ref_iTest := 99;            // SA0145
IF __ISVALIDREF(ref_iTest) THEN
    ref_iTest := 88;
END_IF

--> SA0145: Possible use of not initialised reference 'ref_iTest'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*
- *Chapter 1.8.3.3.2.49.15 “SA0124: Pointer dereferences in declarations” on page 4216*

SA0150: Violations of lower or upper limits or the metrics

Detects the POUs that violate the activated metrics at the lower or upper limits

Justification: Code that complies with certain metrics is easier to read, easier to maintain, and easier to test.

Importance: High

PLCopen rule: CP9

Example

Initial situation: The *“Number of calls”* metric is selected in *“Project Settings ➔ Static Analysis ➔ Metrics”*. Lower limit: `0`; upper limit: `3`. `Prog_1` is called five times.

When running the static analysis, the *“SA0150: Metric violation for Prog_1. Report for metric calls (5) > 2”* error is issued in the message view, in the *“Build”* category.

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0160: Recursive calls

Detects recursive calls in actions, methods, and properties of function blocks. Also detects possible recursions from virtual function calls and interface calls.

Justification: Recursions lead to non-deterministic behavior and are therefore a source of errors.

Importance: Medium

PLCopen rule: CP13

Example

The following method `Call` is assigned to the function block `FB_Test`:

```
FUNCTION_BLOCK FB_Test
VAR
    bParameter: BOOL;
END_VAR

METHOD Call : BOOL
VAR_INPUT
END_VAR
Call := THIS^.Call();        //SA0160
```

The program `PLC_PRG` calls `FB_Test`:

```
PROGRAM PLC_PRG
VAR
    fbTest : FB_Test;
    bValue : BOOL;
END_VAR
bValue := fbTest.bParameter;
fbTest.Call();

--> SA0160: Recursive call detected: 'PLC_PRG -> FB_Test.Call ->
FB_Test.Call
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0161: Unpacked structure in packed structure

Detects unpacked structures that are used in packed structures

Justification: The compiler typically sets an unpacked structure to an address that allows aligned access to all elements within the structure. If you create this structure in a packed structure, then aligned access is no longer possible. Furthermore, access to an element in the unpacked structure can lead to a misalignment exception.

Importance: High

Example

The structure `structSingleDataRecord` is packed, but it contains the unpacked structures `struct4Byte` and `struct9Byte`.

```
{attribute 'pack_mode' := '1'}
TYPE structSingleDataRecord :
STRUCT
    str9ByteData: struct9Byte;    (* 9 BYTE *)
    str4ByteData: struct4Byte;    (* 4 BYTE *)
    udi1: UDINT;
    udi2: UDINT;
    udi3: UDINT;
    usi4: USINT;
END_STRUCT
END_TYPE (* 9 BYTE *)
TYPE struct9Byte :
STRUCT
    usiRotorSlots: USINT;        (* 1 BYTE *)
    uiMaxCurrent: UINT;           (* 2 BYTE  *)
    usiVelocity: USINT;           (* 1 BYTE  *)
    uiAcceleration: UINT;        (* 2 BYTE  *)
    uiDeceleration: UINT;        (* 2 BYTE *)
    usiDirectionChange: USINT;    (* 1 BYTE *)
END_STRUCT
END_TYPE TYPE struct4Byte :
STRUCT
    //udiDummy : UDINT;
    rRealDummy : REAL;
END_STRUCT
END_TYPE

--> SA0161: Declaration of an unpacked struct 'struct9ByteData'
inside a packed struct 'structSingleDataRecord'
--> SA0161: Declaration of an unpacked struct 'struct4ByteData'
inside a packed struct 'structSingleDataRecord'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0162: Missing comments

Detects uncommented locations in the program

Justification: Complete commenting is required by many programming guidelines, and it increases the readability and maintainability of the code.

Importance: Low

PLCopen rule: C2

Comments are required in the following cases:

- Declaration of variables (Comments are located either above the declaration or to the right of the declaration.)
- Declaration of programs, function blocks, or methods (Comments are located above the declaration in the first line.)

Example

```
PROGRAM PLC_PRG
VAR
    iMaxValue:  INT;
END_VAR

--> SA0162: Missing comment for 'PLC_PRG'
--> SA0162: Missing comment for 'iMaxValue'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0163: Nested comments

Detects nested comments

Justification: Nested comments should be avoided because they are difficult to read.

Importance: Low

PLCopen rule: C3

Example

```
{attribute 'do-analysis'}
(* That is
(* nested comment 1 *)
*)
PROGRAM PLC_PRG
VAR
(* That is
// nested comment 2
comment *)
    iVal1: INT;
    iVal2: INT;

(* That is
(* nested comment 3 *) *)
    pVal3: POINTER TO DWORD;
    hugo: INT;
END_VAR

(* That is
// nested comment 4
comment *)

iVal1 := iVal1 + 1;

(* That is
(* nested comment 5 *)
*)

(* Not that one *)

--> SA0163: Nested comment 'nested comment 1'
--> SA0163: Nested comment 'nested comment 2'
--> SA0163: Nested comment 'nested comment 3'
--> SA0163: Nested comment 'nested comment 4'
--> SA0163: Nested comment 'nested comment 5'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0164: Multiline comments

Detects multiline comments that are coded as `(* comment *)`. Only single-line comments that are coded as `// comment` are permitted.

Justification: Some programming guidelines prohibit multiline comments in code because the beginning and end of a comment could get lost and the closing comment bracket could be deleted by accident.

You can deactivate this check by means of the pragma `analysis`, also for comments in the declaration part.

Importance: Low

PLCopen rule: C5

Example

```
{attribute 'do-analysis'}
(*
    This is a multi-line comment                // SA0164
*)
PROGRAM PLC_PRG
VAR
// This is a single line comment
    a: DINT;
END_VAR

(* This is not a single line comment *)         // SA0164
a := a + 1;
```

See also

- *Chapter 1.8.3.3.1.2 “Attribute 'analysis'” on page 4150*

SA0165: Tasks calling other POUs than programs

Detects tasks that call function blocks or functions instead of a program

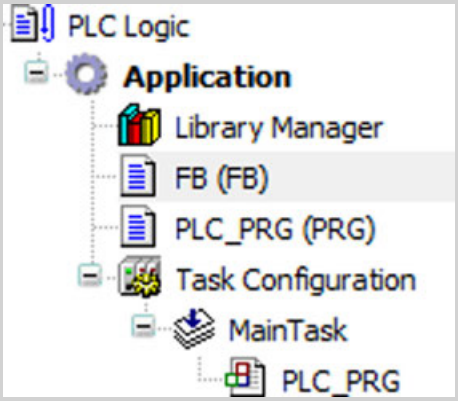
Justification: This rule is part of the PLCopen Coding Guidelines. Therefore, compliance is also checked in CODESYS. We do not see any problems with data consistency in CODESYS if tasks would call POUs other than programs. However, problems can occur if the code is to be ported to other platforms.

Importance: Low

PLCopen rule: CP16

Tasks are inserted below the task configuration. The POUs to be called are configured in the tasks. The POUs must be the *“Program”* type. The *“Function block”* and *“Function”* types are not permitted.

Example



See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0166: Max. number of input/output/in-out variables...

Detects whether or not a defined number of input variables (`VAR_INPUT`), output variables (`VAR_OUTPUT`) or `VAR_IN_OUT` variables is exceeded in a POU. In the *“Project Settings”*, double-click the rule entry to open a dialog where you define the maximum number.

Justification: This is about checking individual programming guidelines. Many programming guidelines provide for a maximum number of POU parameters. Too many parameters make the code unreadable and the POUs difficult to test.

Importance: Medium

PLCopen rule: CP23

Example

In the project settings, for Rule 166, you have defined a maximum number of `1` for `VAR_IN_OUT` variables.

```
FUNCTION_BLOCK FB1
VAR_INPUT
 xIn    : BOOL;
END_VAR
VAR_IN_OUT
 xInOut1 : BOOL;
    xInOut2 : BOOL;
END_VAR

--> SA0166: Too many VAR_IN_OUT variables in POU 'FB1'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0167: Temporary function block instances

Detects function block instances that are declared as temporary variables. This affects instances that are declared in a method or function or as `VAR_TEMP`, and therefore are reinitialized in each processing cycle or for each POU call.

Justification: Function blocks have a state that is usually maintained over multiple PLC cycles. An instance on the stack exists only for the duration of the function call. Therefore, it rarely makes sense to create an instance as a temporary variable. Secondly, function block instances are often large and need a lot of space on the stack (which is usually restricted to controllers). Thirdly, the initialization and often also the scheduling of a function block can take a long time.

Importance: Medium

Examples

```
PROGRAM PLC_PRG
VAR
END_VAR
VAR_TEMP
    yafb: AFB;
END_VAR

FUNCTION Fun : INT
VAR_INPUT
END_VAR
VAR
    funafb: AFB;
END_VAR

METHOD METH : INT
VAR_INPUT
END_VAR
VAR
    methafb: AFB;        // SA0167
END_VAR

--> SA0167: Temporary function block instance: 'methafb'
```

See also

- *Chapter 1.8.3.3.1.5 “Attribute 'analysis:report-multiple-instance-calls'” on page 4152*

SA0168: Unnecessary Assignments

Detects assignments to variables which do not have any effect in the code.

Justification: When values are assigned to a variable multiple times without the variable being evaluated between assignments, the first assignments do not have any effect on the program.

Importance: Low

Example

```
PROGRAM PLC_PRG
VAR
    dwVal1 : DWORD;
    dwVal2 : DWORD;
END_VAR


dwVal1 := 1;                // unnecessary assignment
IF  dwVal2 > 100 THEN
    dwVal2 := 0;
    dwVal2 := dwVal2 + 1;
END_IF
dwVal1 := 2;

--> SA0168: The variable 'dwVal1' is assigned but its value is
never used.
```

See also

- *Chapter 1.8.3.1 “Configuring and Running Static Analysis” on page 4130*

SA0169: Ignored outputs

Detects the outputs of methods and functions that are not specified when calling the method or function.

Justification: Ignored outputs can be a notice about an unhandled error or meaningless function calls because results are not used.

Importance: Medium

Example

```
FUNCTION Fun1
VAR_INPUT
    bIn : BOOL;
VAR_END
VAR_OUTPUT
 bOut : BOOL;
END_VAR

PROGRAM PLC_PRG
VAR
        bValue :BOOl;
END_VAR

Fun1(bIn : TRUE);

 --//SA0169: The output 'bOut' is ignored when called.
```

See also

- *Chapter 1.8.3.3.2.28.5 “SA0036: Unused output variables” on page 4172*

1.8.4 Drive composer pro integration

Drive Composer Pro is a start-up and maintenance tool for ABB's common architecture drives. The tool is used to view and set drive parameters, and to monitor and tune process performance.

Drive Composer Pro provides:

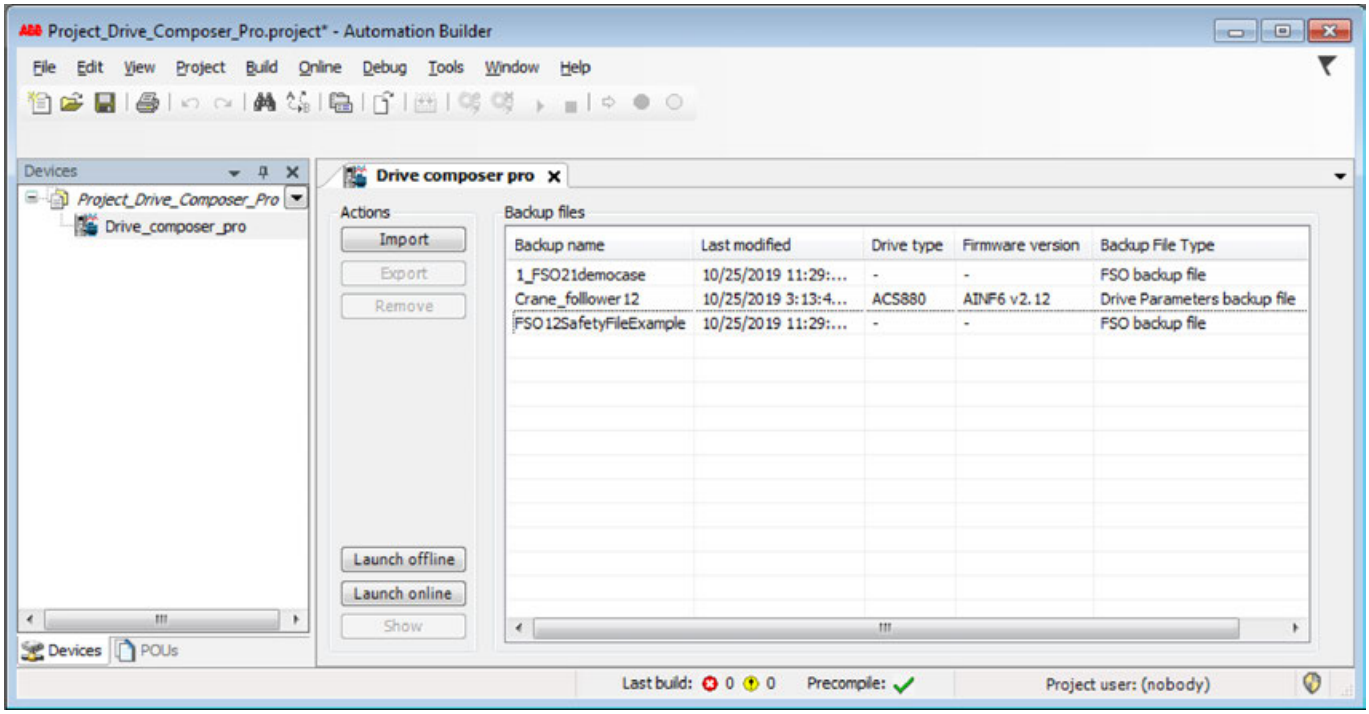
- Setting parameters,
- taking local control of the drive from the PC,
- event logger handling
- control diagrams,
- fast monitoring,
- working with multiple drives on the PC tool network,
- macro script editing for parameters and much more.

1. Add *“Drive Composer Pro”* object into the tree via add object dialog.
2. Open the *“Drive Composer Pro”* with double-click on the object.

In the following section important functions are described.

Import of backup files

1. Import of FSO backup files (*.dcsafety) and Drive Parameters backup files (*.dcparamsbak) into Automation Builder project via the Drive Composer Pro object in the device tree.
2. View of integrated FSO backup files and Drive Parameters backup files in Automation Builder project - refer to figure below.

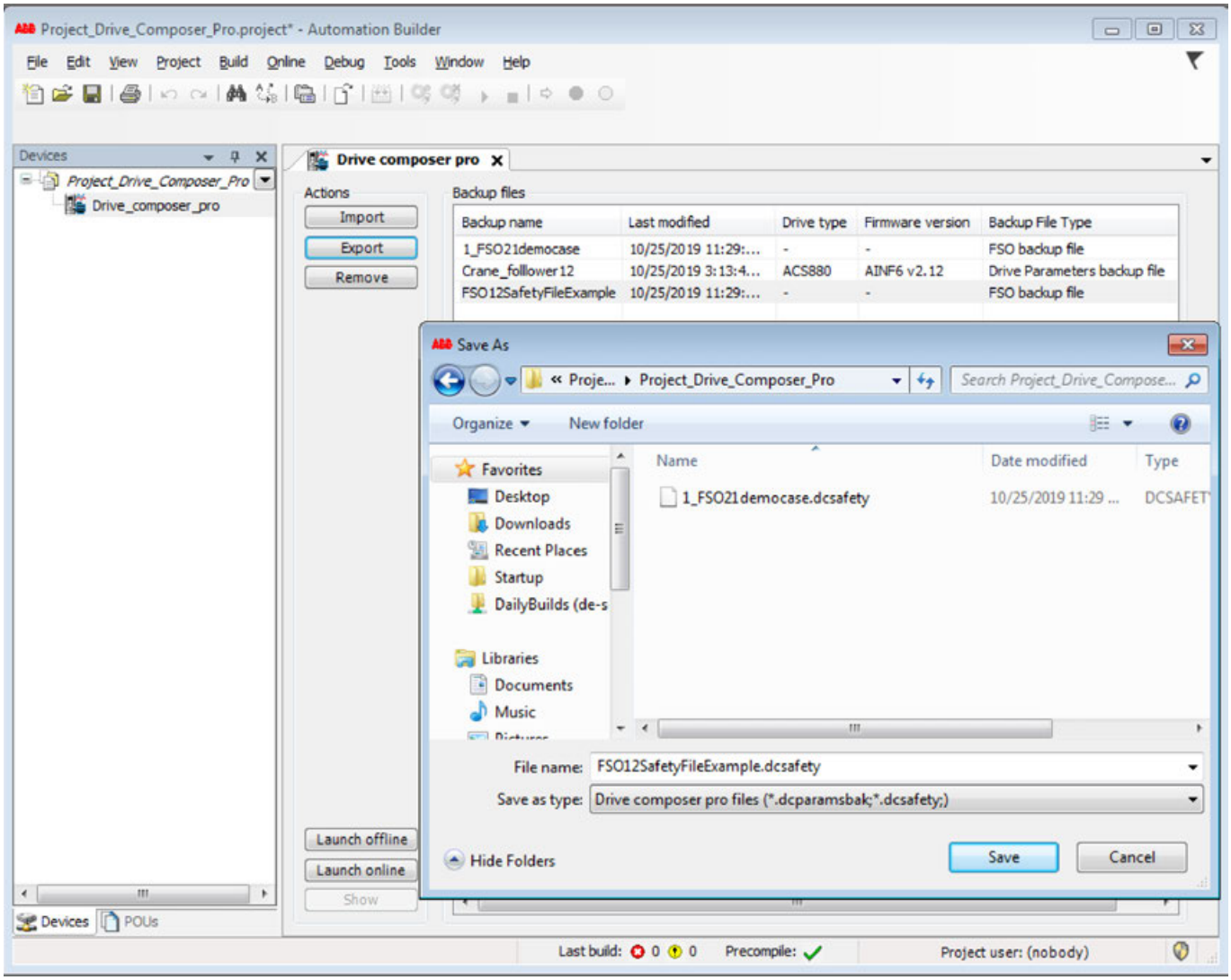


⇨

Drive Composer Pro can't be launched directly with integrated “FSO backup files” but they have to be loaded manually via context menu on the drive in Drive Composer Pro → “Safety Settings”.

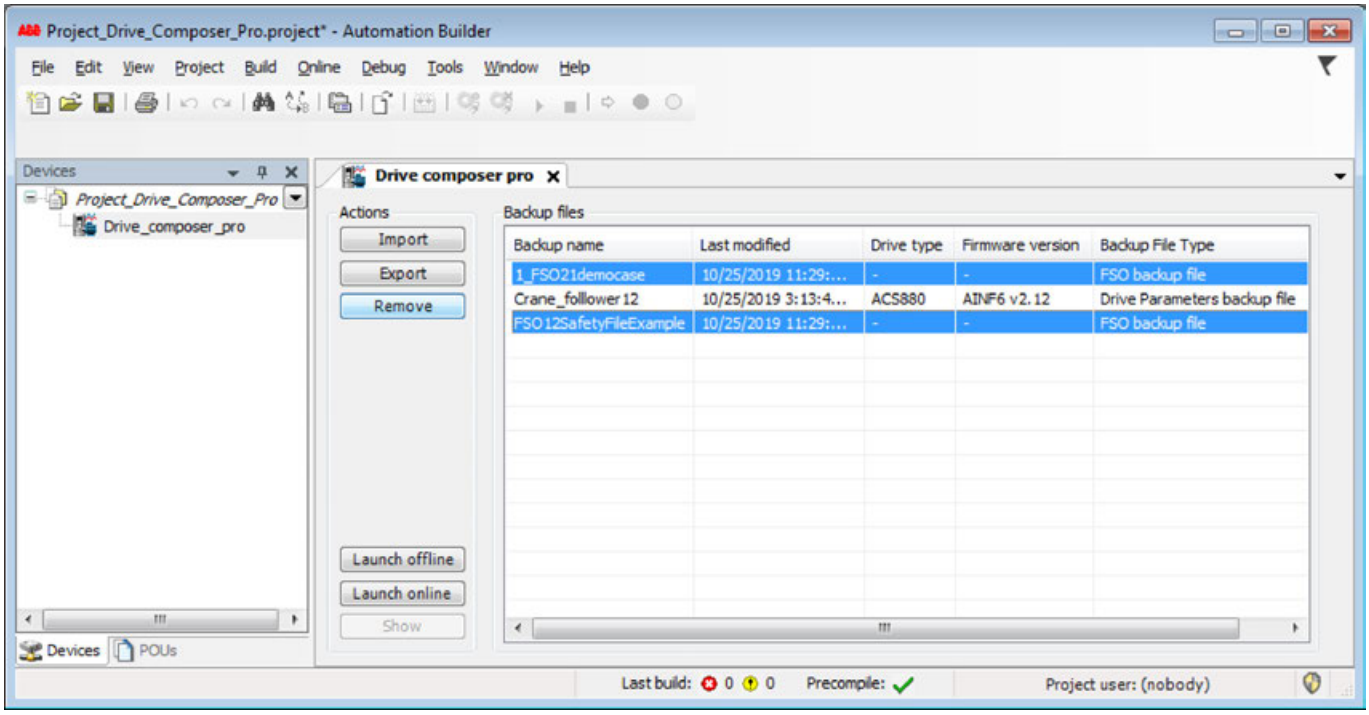
Export of backup files

1. Select the FSO and Drive Parameters backup files.
2. Export the selected file by clicking *[Export]*.
   ⇨ Select the desired storage path.



Remove of backup files

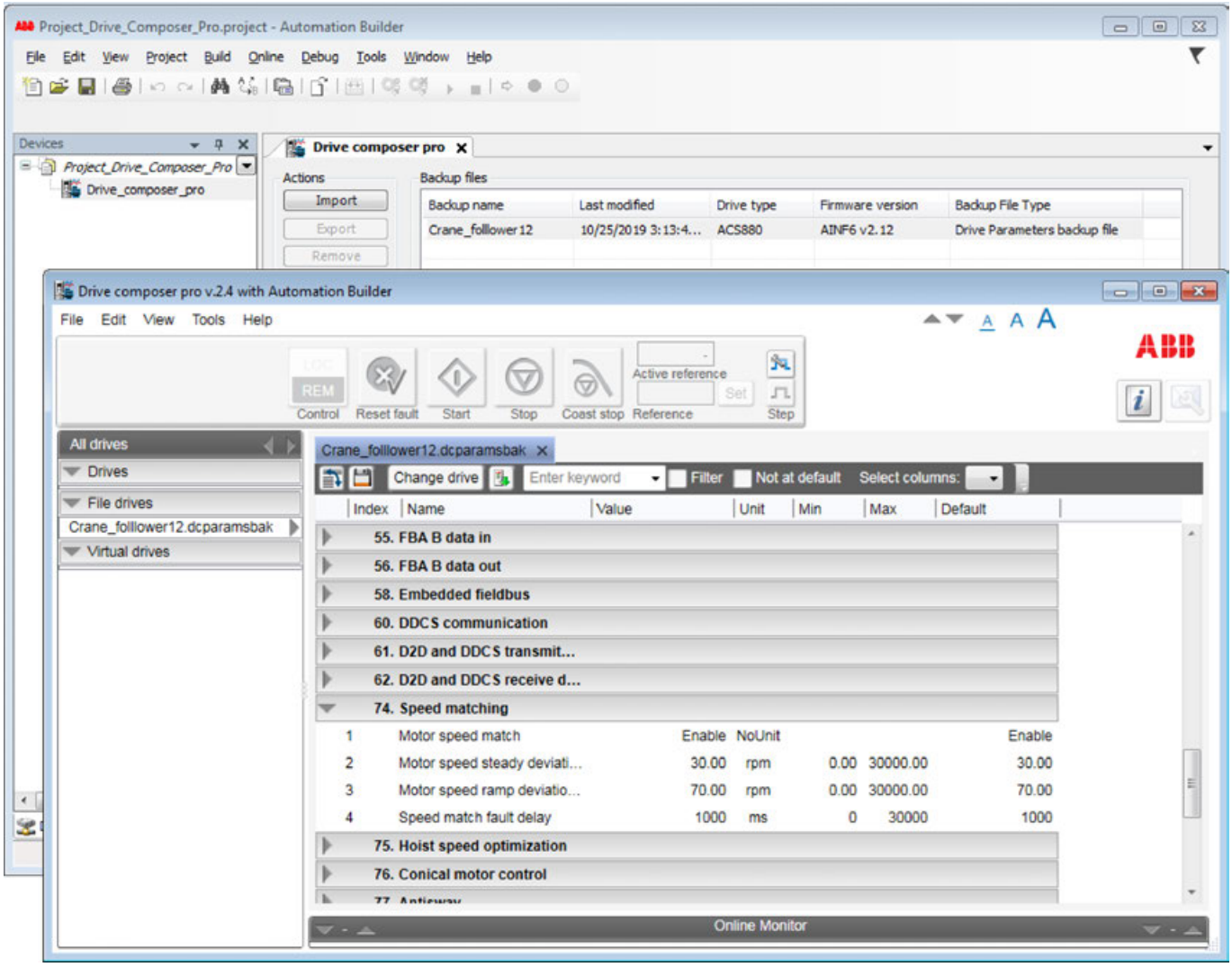
1. Select the FSO and Drive Parameter backup files from Automation Builder project.
2. Remove the selected files by clicking *[Remove]*.



View standard drive parameter backup files

1. Open the *“Crane_follower12.dcparamsbak”* with double-click.
2. The *“Drive Composer Pro”* starts automatically.

> *Standard Drive Parameter backup files (*.dcparamsbak) are automatically displayed under “File Drives”.*



3. Saved changes in the standard drive parameter backup file are automatically updated in the Automation Builder project.

1.8.5 Professional Version Control

SVN integration in CODESYS

Professional Version Control allows for the development of CODESYS projects under version control by Apache™ Subversion®. Professional Version Control provides an SVN client integrated in CODESYS. The objects of your project are versioned in a central *SVN repository*.

As a rule, the SVN repository should be created in a server configuration and located on a server. For testing purposes, you can create a local SVN repository where you can access via `file://`.

Professional Version Control requires a valid license and can be installed using the Automation Builder Installer or the Automation Builder Installation Manager.

1.8.5.1 Getting Started

The following steps are required in order to develop your CODESYS project with Professional Version Control with version control by Apache™ Subversion®:

1. Install the Professional Version Control package in CODESYS.
2. Install an SVN server.
3. Create an SVN repository.
4. Open your CODESYS project in CODESYS.
5. Import the CODESYS project into the SVN project archive.
   ⇨ The CODESYS project is saved in the SVN repository.
6. To edit and further develop the project with SVN version control, the project is edited in CODESYS and then committed to the SVN repository.

A detailed description of these individual steps is located in the following sections.

See also

- *Chapter 1.8.5.3 “Using an SVN Repository” on page 4232*
- *Chapter 1.8.5.4 “Using Working Copies” on page 4234*

1.8.5.2 Version control

What is version control?

Apache™ Subversion® (SVN) is a tool for version and revision management of current and previous versions of files, such as source code, websites, and documentation. Apache™ Subversion® is a registered trademark of the Apache Software Foundation.

Revision management (also known as version control, version management, and source code management) is the management of changes to documents, programs, and other information that is stored as computer files. Version control is employed frequently in software development when a team of employees works on the same files.

Tasks

- Co-writing of changes in revisions: At any time, you can show who made which changes at which time.
- Restoring of old revisions of individual files: At any time, you can reverse accidental changes to files.
- Archiving of special revisions of a project: At any time, you can revert to older versions.
- Coordination of common access of developers to data
- Development of a project simultaneously in multiple branches

Script Engine SVN Add-on API

Professional Version Control provides a scripting-interface for SVN.

1.8.5.3 Using an SVN Repository

An *SVN repository* usually saves information as a file system tree, a hierarchy of files, and directories. Any number of clients connects to the SVN repository and reads or writes changes to the files in revisions.

Creating an SVN repository

NOTICE!

Consult with your IT specialists for more information, for example how to create an SVN repository. For production purposes, we recommend a strictly dedicated administrative SVN server.

We recommend that you create the suggested default directory structure in the SVN repository.

See also

– *http://svnbook.red-bean.com/en/1.8/svn.tour.importing.html#svn.tour.importing.layout*

Creating an SVN repository for testing purposes

NOTICE!

Use the `file://` access method for testing purposes only.

You can reach SVN repositories that were created in format 1.8 or 1.9 via the `file://` protocol.

For testing purposes, you can create a local SVN repository without installing your own server. The SVN repository is accessed via `file://` and provides the same functionality as a server.

Creating a test repository with TortoiseSVN

☑ Requirement: The SVN client TortoiseSVN 1.9 is installed on the development system.

1. Create a new, empty folder on your local file system. The test repository will be created there.
   ⇨ Example: `D:\SVN repository`
2. Click *"TortoiseSVN ➔ Create repository here"*.
   ⇨ The dialog *"Create repository"* opens.
3. Click *"Create directory tree"*.
   ⇨ The `SVN repository` is created.

See also

- Documentation TortoiseSVN *Documentation TortoiseSVN*

Accessing the SVN repository

Table 759: SVN repository URLs

`file:///`	Direct access to an SVN repository (on local hard drive)
`http://`	Access via WebDAV protocol to Apache server that is supported by SVN
`https://`	As `http://`, but with SSL encryption
`svn://`	Access via own protocol to an `svnserve` server
`svn+ssh://`	As `svn://`, but tunneled via SSH

Import the project into the SVN repository.

1. Open the CODESYS project that you want to save in the SVN repository.
   ⇨ Example: `A.project` is open.
2. Click *"Project ➔ SVN ➔ Import project to SVN"*.
   ⇨ The *"Browse SVN repository"* dialog opens.
3. Select the directory `file:///D:/SVN repository/trunk` in the directory tree.
4. Select the command.
   ⇨ The *"Create remote directory"* dialog opens.
5. Specify the URL for the new directory.
   Note: Because the new directory should contain the CODESYS project, specify the project name with extension here.
   ⇨ `file:///D:/SVN%20repository/trunk/A.project`
6. Click *"OK"* to close the dialog.
7. Select the new project and click *"OK"* to exit the *"Browse SVN repository"* dialog.
   ⇨ The *"Import Project to SVN"* dialog opens. The directory `file:///D:/SVN repository/trunk/A.project` is specified in *"URL of SVN repository"*.

See also

- *Chapter 1.8.5.5.1 "Overlay Icons" on page 4235*

1.8.5.4 Using Working Copies

Checking out a project

You can copy CODESYS projects to your development system that are saved in the SVN repository.

Creating a working copy

1. Open CODESYS.
2. Click *"Project ➔ SVN ➔ Checkout"*.
   ⇨ The *"Checkout"* dialog opens.
3. Specify the URL of the SVN repository and select a project in the SVN repository tree.
   If a CODESYS project has the extension `.project` or `_project`, then it is recognized automatically as a project at checkout. If it has the extension `.library` or `_library`, then it is recognized as a library project.
4. In *"Checkout to"*, specify the name and location of the working copy on your development system.
5. Click *"OK"* to close the dialog.
   ⇨ The project opens in CODESYS. In the object tree of the project, the SVN link is shown with overlaid icons. Now the project is saved as a working copy on your development system.

See also

- *Chapter 1.8.5.5.1 "Overlay Icons" on page 4235*

Editing the working copy

Update the working copy before you start editing, especially if the project is revised by a team. This is how you avoid conflicts.

1. Open the working copy.
2. Click *“Project ➔ SVN ➔ Update project”* (symbol:).
   ⇨ You working copy is current.
3. Revise your project.
4. Click *“Project ➔ SVN ➔ Edit SVN working copy”*.
   ⇨ The dialog opens. There you can browse your changes.
5. Close the dialog.
6. If necessary, you can click *“SVN ➔ Revert”* in the context menu.
   ⇨ The file is reverted back to the base revision and your changes are discarded.
7. If necessary, you can click *“Compare”* in the context menu of an edited object.
   ⇨ The compare dialog opens. You can resolve any conflicts here.
8. Close the compare dialog.
9. Click *“Project ➔ SVN ➔ Commit project”* (symbol:).
   ⇨ The *“Commit”* dialog opens.
10. In *“Message”*, specify a log entry that describes your changes. Example: `Changes for customer ABC, request 1234.`
    ⇨ Your changes are saved in the SVN repository as a revision with a revision number.

See also

- *Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 4238*

Changed working copy format in Professional Version Control `V4.1.0.0` and later

For projects in version Professional Version Control `V4.1.0.0` and later, the working directory (working copy) has a new format.

If you open a project that was created with `V4.0.4.0` or earlier, then the project is updated automatically to the new format when it is opened.

If you open a project that was created with `V4.0.4.0` or earlier and the project is based on an older SVN version of `1.7.x` or earlier, then you are prompted whether or not CODESYS should update the format. If you decline the update, then the SVN link of the project is deactivated. You can still load and edit the project.

The update does not have an effect on saving to the SVN server. You can also checkout projects with earlier versions of the client. The new format affects only the local working directory.

See also

- *http://svnbook.red-bean.com/en/1.8/svn.ref.svn.c.upgrade.html*

1.8.5.5 Reference, User Interface

1.8.5.5.1 Overlay Icons

Every object in CODESYS has a status value in the SVN repository. This status value is displayed in the object tree (in the *“POUs”*, *“Devices”*, or *“Modules”* views) for each object by overlay icons.

Table 760: Overlay icons

	Object is planned to be added to the SVN repository.
	Object conflicted
	Object deleted
	Object modified
	Object with modification in the metadata
	Object with modifications in the memory format
	Object normal
	Object write-protected (read-only)
	Object locked
	Object with deleted subobjects
	Object ignored on commit
	External object
	Ignored object
	Unversioned object
	Object with modified subobjects
	The object is not saved in the SVN repository. It will be created again when loaded from SVN.
	`SVN_VERSION_INFO` temporarily unavailable, for example as with interface libraries
	The status of the object is not updated.
	The object was modified on the server (**U**pdate available).
	The object was locked on the server by another user (or in another working directory).
	Tree conflict by changes to the structure of the project

1.8.5.5.2 Commands

Not all commands are available in the logged in state because some SVN commands of the project could be changed.

Table 761: Availability of commands

Command	Not Logged In	Logged In
Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 4238	X	X
Chapter 1.8.5.5.2.2 “Command 'Edit SVN working copy'” on page 4239	X	
Chapter 1.8.5.5.2.3 “Command 'Import project to SVN'” on page 4242	X	X
Chapter 1.8.5.5.2.4 “Command 'Checkout'” on page 4242	X	X
Chapter 1.8.5.5.2.5 “Command 'Commit', Command 'Commit Project'” on page 4244	X	X
Chapter 1.8.5.5.2.6 “Command 'Compare'” on page 4247	X	
Chapter 1.8.5.5.2.7 “Command 'Compare with HEAD revision'” on page 4247	X	
Chapter 1.8.5.5.2.8 “Command 'Compare with revision'” on page 4247	X	
Chapter 1.8.5.5.2.9 “Command 'Compare to remote project...'” on page 4248	X	
Chapter 1.8.5.5.2.10 “Command 'Include externals to project', Command 'Include externals'” on page 4249	X	
Chapter 1.8.5.5.2.11 “Command 'Ignore on commit'” on page 4251	X	X
Chapter 1.8.5.5.2.12 “Command 'SVN Info'” on page 4251	X	X
Chapter 1.8.5.5.2.13 “Command 'Show properties'” on page 4252	X	X
Chapter 1.8.5.5.2.14 “Command 'Get lock'” on page 4252	X	X
Chapter 1.8.5.5.2.15 “Command 'Steal locks'” on page 4253	X	X
Chapter 1.8.5.5.2.16 “Command 'Release lock'” on page 4253	X	X
Chapter 1.8.5.5.2.17 “Command 'Release locks recursively'” on page 4253	X	X
Chapter 1.8.5.5.2.18 “Command 'Show log', Command 'Show project log'” on page 4253	X	X
Chapter 1.8.5.5.2.19 “Command 'Revert', Command 'Revert project'” on page 4255	X	
Chapter 1.8.5.5.2.20 “Command 'Revert to revision', Command 'Revert project to revision'” on page 4256	X	
Chapter 1.8.5.5.2.21 “Command 'Update', Command 'Update project' ” on page 4256	X	
Chapter 1.8.5.5.2.22 “Command 'Update to revision'” on page 4257	X	
Chapter 1.8.5.5.2.23 “Command 'Update only this'” on page 4258	X	

Command	Not Logged In	Logged In
Chapter 1.8.5.5.2.24 “Command 'Disconnect project from SVN'” on page 4258	X	X
Chapter 1.8.5.5.2.25 “Command 'Switch'” on page 4258	X	
Chapter 1.8.5.5.2.26 “Command 'Un-Ignore on commit'” on page 4259	X	X
Chapter 1.8.5.5.2.27 “Command 'SVN Cleanup'” on page 4259	X	X
Chapter 1.8.5.5.2.28 “Command 'Clear authentication data' ” on page 4260	X	X
Chapter 1.8.5.5.2.29 “Command 'Merge changes'” on page 4260	X	
Chapter 1.8.5.5.2.30 “Command 'Connect to existing project'” on page 4261	X	X
Chapter 1.8.5.5.2.31 “Command 'Resolve conflict' ” on page 4262	X	
Chapter 1.8.5.5.2.32 “Command 'Work in offline mode'” on page 4262	X	X
Chapter 1.8.5.5.2.33 “Command 'Copy (Branch/Tag)'” on page 4263	X	
Chapter 1.8.5.5.2.34 “Command 'Pending Changes'” on page 4264	X	X

Command 'SVN Repository Browser'

Symbol: 

Function: This command opens the SVN repository browser. The contents of an SVN repository is shown in a tree structure here. You can search through the repository in the browser.

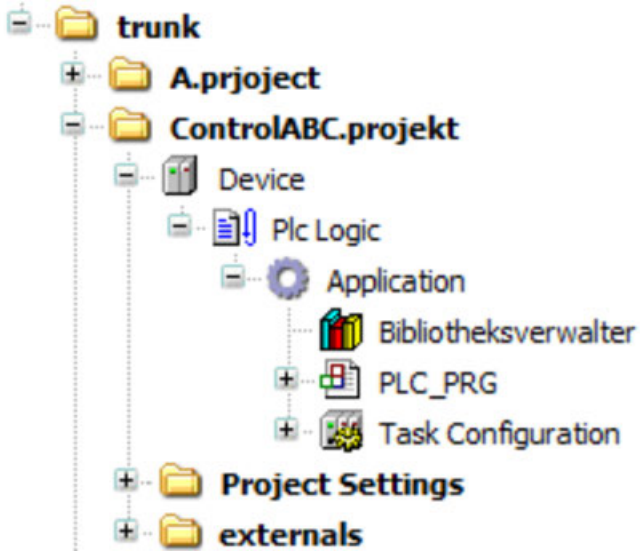
Call: Menu bar: *“Project ➔ SVN”*.

Depending on the selected object, the following commands are available in the context menu:

- *“Show log”*
- *“Checkout”*
- *“Create folder”*
- *“Copy to”*
- *“Rename”*
- *“Delete”*

Double-clicking the object with the right mouse button opens the log dialog.

Dialog 'SVN Repository Browser'

"URL"	URL in SVN repository Example: `https://svnserver/repository/trunk/ControlABC.project` Tip: As soon as a valid SVN repository is specified, you can browse and select a specific project by means of the adjacent button.
15	Opens the dialog *"Select revision"*. The button is labeled with the currently selected revision: ● *"HEAD"*: Top revision (latest). Preset ● *"3"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) Note: The dialog provides the same options as the *"Revision"* group.
	Updates the browser view by rescanning the SVN repository.
	Navigates the URL address up by one folder.
Left area	Directory tree in the SVN repository. Project nodes are shown in bold.  Note: In this view, you can directly edit the project name and the name of the superordinate folder.
Right area	List of objects of the selected directory
"Close"	Closes the dialog

See also

- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 4267*

Command 'Edit SVN working copy'

Symbol:

Function: This command opens the dialog *"Edit SVN working copy"* and displays the working copy in a browser from the SVN view.

Call: Menu bar: *"Project ➔ SVN"*.

The functionality of the browser allows for:

- Access to and actions on objects that are not displayed in the *"Devices"* view.
- Actions on objects that can lead to exceptions in the *"Devices"* view.
- Editing of global objects that are modified, in conflict, or blocked.

Dialog ‘Edit SVN working copy'

Table 762: “Edit SVN working copy: <project name> - <project URL>”

“Path in SVN repository”	Display of working copy from SVN view. The file and folder structure of the objects in the project are presented in a tree view. In this way, the recursion depth of an object is clear. : Object selected for the following menu command
“Name of object”	File name of the object Example: `Application`
“ Node type”	The top node is the project root directory.
“Text status”	Object status: ● *“modified”* ● *“added”* ● *“deleted”* ● *“non-versioned”* ● *“Conflicted”*
“Property status”	Status in SVN repository: ● *“modified”* ● *“added”* ● *“deleted”* ● *“Conflicted”* ● *“normal”*
“Revision”	Revision number
“Conflict information”	File conflict, property conflict, or tree conflict
“Lock”	For locked objects, the user who applied the lock is displayed. Example: `b.mayer`
“Lock comment”	Lock message. Implicit, normal, or stolen lock.
“URL”	URL of the object

Table 763: Menu commands

“Select ➔ All”	Selects all files.
“Select ➔ None”	Deselects all files.
“Select ➔ Modified”	Selects the modified files.
“Select ➔ Conflicted”	Selects the conflicted files.
“Select ➔ Locked”	Selects the locked files.
	Updates the working copy. Changes made by others are added from the SVN repository to your working copy.
“Update ➔ Project”	Updates all files of the project.
“Update ➔ Selected nodes”	Updates only the selected files.
“Update ➔ Selected nodes and children”	Updates the selected files and subordinate files.
“Reset”	Discards your changes to the working copy. Then the object corresponds to the revision in the repository.

"Delete ➔ Selected nodes"	Deletes the selected objects from the working copy.

"Commit"	Commits your changes to the SVN repository. Any locked objects will be unlocked.
"Commit ➔ Project"	Commits all files in the project.
"Commit ➔ Selected nodes"	Commits only the selected files.
"Commit ➔ Selected nodes and children"	Commits the selected files and subordinate files.

	Commands for managing locks.
"Locks ➔ Revalidate all"	Checks the validity of locks in the working copy. Any invalid locks will be unlocked.
"Locks ➔ Release locks"	Releases the lock.
"Locks ➔ Acquire locks"	Locks the object from editing by others.
"Locks ➔ Steal locks"	Locks the file for you and removes the lock of another user. Tip: Avoid stealing a lock because the changes made by another user can be lost.

	Commands to resolve conflicts.
"Conflicts ➔ Mark as resolved"	Indicates a displayed conflict in the SVN repository as marked and resolved. Note: Select the command if you edited and resolved the displayed conflict. Then you can commit changes again.
"Conflicts ➔ Resolve using theirs"	Resolves the conflict: In the SVN repository, the changes are accepted that were committed by other users. Your changes are discarded.
"Conflicts ➔ Resolve using mine"	Resolves the conflict: In the SVN repository, the changes to your working copy are accepted and the changes by other users are discarded.

"Show log"	Opens the dialog *"Log - Application"*. The history of the selected node is shown here. The previous revisions are displayed with the respective actions.

"Change location"	Changes the storage location of the selected object within the working copy. Example: You can resolve a tree conflict by saving the local object to another location. Then update the parent object to apply it to the locked children.

"Update"	Updates the browser view by rescanning the working copy.

"Cleanup"	Executes an SVN cleanup operation on the working copy.

See also

- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 4256*
- *Chapter 1.8.5.5.2.19 "Command 'Revert', Command 'Revert project'" on page 4255*
- *Chapter 1.8.5.5.2.5 "Command 'Commit', Command 'Commit Project'" on page 4244*
- *Chapter 1.8.5.5.2.31 "Command 'Resolve conflict' " on page 4262*
- *Chapter 1.8.5.5.2.14 "Command 'Get lock'" on page 4252*
- *Chapter 1.8.5.5.2.16 "Command 'Release lock'" on page 4253*
- *Chapter 1.8.5.5.2.15 "Command 'Steal locks'" on page 4253*
- *Chapter 1.8.5.5.2.25 "Command 'Switch'" on page 4258*
- *Chapter 1.8.5.5.2.18 "Command 'Show log', Command 'Show project log'" on page 4253*
- *Chapter 1.8.5.5.2.27 "Command 'SVN Cleanup'" on page 4259*

Command 'Import project to SVN'

Symbol: 

Function: This command opens the *"Import Project to SVN"* dialog for importing a CODESYS project to the SVN repository.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement

- You have access to an SVN repository and you know its URL.
- You have read access to the entire project.

> **NOTICE!**
> Projects are always saved unencrypted on the server. Therefore, take appropriate security measures (for example, respective access rights to the SVN server) for protecting your projects.

See also

- User and access management in *Protect and save project*

Dialog 'Import Project to SVN'

"URL of SVN repository"	URL of the SVN repository with the new project folder where the files are imported Example: `https://svnserver/repository/trunk/ControlABC.project` Hint: When importing libraries, specify the extension `.library` or `_library`. For projects, specify the extension `.project` or `_project`. Then the project type is recognized automatically at checkout and the options are set accordingly in the *"Checkout"* dialog.
	Opens the *"SVN Repository Browser"* dialog The previous project structure is displayed and you can edit them here.
"Import message"	Text for use as log message Example: `Control project for customer A`
"Recent messages"	Opens the *"Recent Messages"* dialog. There you can reuse the last log messages.
"Generate SVN_VERSION_INFO"	☐: The object `SVN_VERSION_INFO` is not created automatically during the import operation. Therefore, the project does not get any global constants or variables for the project metadata.
"OK"	Creates the current project in the SVN repository and imports the project objects. The local project in CODESYS Development System is linked to the SVN repository. Overlay icons show this in the object trees.

See also

- *Chapter 1.8.5.5.2.4 "Command 'Checkout'" on page 4242*
- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 4238*
- *Chapter 1.8.5.5.1 "Overlay Icons" on page 4235*

Command 'Checkout'

Symbol: 

Function: This command opens the *"Checkout"* dialog. Here you can checkout a project stored in the SVN repository as a working copy.

Call: Menu bar: *"Project ➔ SVN"*.

Dialog 'Check-out'

Table 764: "URL of SVN repository"

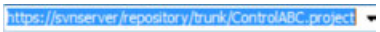
	URL of the project in the SVN repository Example: `https://svnserver/repository/trunk/ControlABC.project` Tip: As soon as a valid SVN repository is specified, you can click the adjacent button or use the options to browse in *"Revision"* and select a specific project.
	Opens the dialog *"Select revision"*. The button is labeled with the currently selected revision: ● *"HEAD"*: Top revision (latest). Preset ● *"15"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) Note: The dialog provides the same options as the *"Revision"* group.
	Opens the *"SVN repository browser"* dialog Here you can browse the SVN repository.

Table 765: "Checkout to"

"Name"	Name of the working copy Example: `ControlABC.project`
"Location"	Storage location of the working copy Example: `/D:/svn/repository/trunk/ControlABC.project`

Table 766: "Checkout as"

"Project"	The project is saved as a CODESYS project "`<project name>.project`".
"Library"	The project is saved as a CODESYS library file "`<project name>.library`".
"Auto-detect"	CODESYS attempts to recognize the project type by means of the extension. The current implementation checks whether the URL of the project ends with "`_library`" or "`.library`". In this case, the project is recognized as a library or a project.

Table 767: "Checkout options"

"Omit externals":	☑: Externals (external objects) are not copied to the working directory.

Table 768: "Revision"

For a description, refer to the section "Dialog 'Select revision'". Note: The group provides the same options as the *"Revision"* dialog.

"OK"	Checks out the project from the SVN repository, saves it locally to the specified location, and opens it in CODESYS as the primary project.

If files were encrypted when imported to the SVN repository, or if they have been committed, then note the following:

When committing to the SVN repository, the information about an encrypted project file is included. However, the type of encryption is not included (password, Wibu security key, X509 certificate). Therefore, it may be necessary to encrypt the working copy again in the project settings. In this case, a dialog opens when exiting the command to notify you of this. Then you are able to switch directly to the project settings.

See also

- *Chapter 1.8.5.5.3.3 “Dialog 'Select revision'” on page 4267*
- *Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 4238*
- *"Version control with Subversion", Section "Revision identifier"*

Command 'Commit', Command 'Commit Project'

Symbol:

Function: The command commits changes that were made in CODESYS to the SVN repository. The *“Commit”* dialog opens for this purpose.

Call:

- Context menu: *“SVN”* to commit exactly this object
- *“Project ➔ SVN ➔ Commit Project”* to commit all changes in the project at the same time

Requirement: At least one object was modified. An object whose contents have been modified is overlaid in the object tree with the , , or symbol.

When you execute the command, the lock on the objects to be committed is lifted automatically.

See also

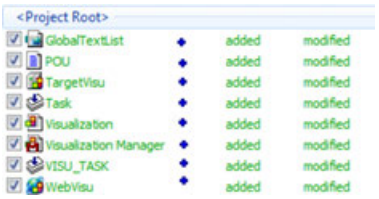
- *Chapter 1.8.5.5.1 “Overlay Icons” on page 4235*

Dialog 'Commit'

Table 769: “Commit to: <URL project/object>”

	URL in SVN repository Example: `file:///D:/SVN repository/trunk/ControlABC.project`
“Log message”	Type in a log message that comments your change. Example: `Bug fix error 123`
“Recent Messages”	Opens the *“Recent Messages”* dialog for displaying the last log messages. You can click a log message to accept it.

Table 770: "Changes made (double-click on object for compare, right-click on object for more operations)"

	List of objects that were changed and can therefore be committed. The SVN URLs mirror the hierarchy of the object in the SVN repository. The objects are highlighted in color according to the object status: ● Blue: Modified ● Green: Added ● Dark red: Deleted ● Red: Conflicted ● Black: Non-versioned (not in SVN repository) Note: These objects are displayed when the *"Show non-versioned objects"* option is selected. ● Gray: Excluded from commit Note: This is the case when the *"Ignore during commit"* option is selected. The list also contains objects which have not been modified but have a lock. This helps to prevent locking from going unnoticed in the repository. Double-click an object in order to open the compare dialog. The revision of the working copy is compared with the base revision. The compare dialog also opens when you click *"Compare"* in the context menu. Right-click an object in order to open the context menu. Note: When the *"Commit Project"* command has been executed, a list of objects is shown here. When the *"Commit"* command is applied to a specific object, only this object is shown (if modified or locked) and its modified or locked child objects.
"Object"	☑: The object is selected for the commit. Example: ☑ Device\Plc Logic\Application\PLC_PRG
"Text status"	Object status in CODESYS ● *"Modified"* ● *"Added"* ● *"Deleted"* ● *"Non-versioned"* ● *"Conflicted"*
"Property status"	Status of the metadata of the object ● *"Modified"* ● *"Added"* ● *"Deleted"* ● *"Conflicted"* ● *"Normal"*
"Lock"	If the object has a lock, then it is shown here the user who applied the lock. Example: `b.mayer`
"Description"	Display of the log message
"Select/Deselect All"	☑: All objects in the list are selected.

"Keep Locks"	☑: Your locked object remains in locked after the commit.
"Keep Change Lists":	☑: The change list also remains after the commit. ☐: The change list is not deleted after the commit.
"Update After Commit (recommended)"	☑: The object/project is updated after the commit. Select this check box to ensure that the project is up-to-date and to prevent conflicts resulting from mixed revisions of working copies.

Button *“Update Project”*	Updates the project Hint: Prevent conflicts by committing a previously updated project/object.
“OK” Keyboard shortcut *[Ctrl]+[Enter]* Keyboard shortcut *[Ctrl]+[Enter]*	Checks the working copy first. Starts the commit of changes when the working copy is current. Opens a dialog when the working copy is outdated. You can then select from the following: ● *“Abort the commit, I want to investigate the issue.”* ● *“Yes, I want to update this project now.”* ● *“Continue with the commit, I know what I do.”* Note: The history of the commit is displayed in the *“Messages”* view. The messages are highlighted in color. ● Blue: Commit a change ● Green: Add an object ● Dark red: Delete/replace an object ● Black: Other messages (summary)

Handling external objects

If the external object is in the same SVN repository, then changes in this external object are listed in the commit dialog and committed together with the internal project. If an external object is in another SVN repository, then you are notified about changes in the external project and you have to commit these separately.

An external object has the “externals” property.

See also

- *Chapter 1.8.5.5.2.6 “Command 'Compare'” on page 4247*
- SVN help: *http://svnbook.red-bean.com/en/1.7/svn.basic.in-action.html#svn.basic.in-action.mixedrevs)*

Context menu (right-click on object)

“Compare”	Opens the compare dialog to compare the working copy with the top-level revision.
“Compare with HEAD version”	Opens the compare dialog to compare the working copy with the HEAD revision.
“Compare with Revision”	The list entries are highlighted in color according to the object status: ● Blue: Modified ● Green: Added ● Dark red: Deleted ● Red: Conflicted ● Black: Non-versioned (not in SVN repository) Note: These objects are displayed when the *“Show non-versioned objects”* option is selected. ● Gray: Excluded from commit Note: This is the case when the *“Ignore during commit”* option is selected for the object.
“Revert”	Discards your changes to the working copy. Then the object corresponds to the revision in the SVN repository.
“Show log”	Shows the version history of the selected object.

"Properties"	Opens the *"SVN Properties"* dialog. The properties are displayed there and you can edit them.
Move to change list	Note: This command has not been implemented yet.

Command 'Compare'

Symbol: 🔍

Function: This command opens a tab that shows the result of the comparison of your working copy and the BASE revision. The base revision is the top-level revision in the SVN repository.

Call:

- Menu bar: *"Project ➔ SVN"*.
- Context menu

Requirement: The object is versioned, it was modified locally, and it does not contain any conflicts.

Multiple tabs can be open at the same time with the comparison of different objects.

> ***Comparison by object type***
> *The comparison dialog makes use of the functionality of the CODESYS command "Project ➔ Compare". In this way, objects are compared according to their object type.*

See also

- *Chapter 1.6.6.1.1.6 "Comparing projects" on page 3640*

Command 'Compare with HEAD revision'

Symbol: 🔍

Function: This command opens a tab that shows the result of the comparison of your working copy and the HEAD revision. The HEAD revision is the top-level revision in the branch. You can revert specific changes that were committed to the HEAD revision.

Call: Context menu: *"SVN"*

Requirement: The object is versioned and not conflicted.

Multiple tabs can be open at the same time with the comparison of different objects.

> ***Comparison by object type***
> *The comparison dialog makes use of the functionality of the CODESYS command "Project ➔ Compare". In this way, objects are compared according to their object type.*

See also

- *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*

Command 'Compare with revision'

Symbol: 🔍

Function: This command opens the *"Project log"* dialog or *"Log - <object> "* where the version history is displayed from the project or an object of the CODESYS project. Here you can select a revision. A tab opens and shows the result of the comparison of your working copy and the revision.

Call: Context menu: *"SVN"*

Requirement: The object is versioned and not conflicted.

Multiple tabs can be open at the same time with the comparison of different objects.

> ***Comparison by object type***
> *The comparison dialog makes use of the functionality of the CODESYS command "Project ➔ Compare". In this way, objects are compared according to their object type.*

See also

- *"Tab 'Project log', Dialog 'Log - <object>'" on page 4254*
- *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*

Command 'Compare to remote project...'

Symbol:

Function: This command opens the dialog *"Select Remote Project for Comparison"*.

Call: Menu bar: *"Project ➔ SVN"*.

See also

- *Chapter 1.4.1.20.3.4.21 "Command 'Compare'" on page 1010*

Dialog 'Select Remote Project for Comparison'

Table 771: "URL of SVN repository"

https://svnserver/repository/trunk/ControlABC.project	URL of the project in the SVN repository that is compared. Example: `file:///D:/SVN repository/trunk/ControlDEF.project` As soon as a valid SVN repository is specified, you can click the adjacent button or use the options to browse in *"Revision"* and select a project.
15	The label on the button corresponds to the selected revision: ● *"HEAD"*: Top revision (latest). ● *"15"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) After clicking the button, the dialog *"Select revision"* opens. Note: The dialog provides the same options as the *"Revision"* group.
...	Opens the dialog *"Browse SVN repository"* to search the SVN repository.

Table 772: "Checkout options"

"Omit externals":	☑: External objects are not compared.

Table 773: “Revision”

Options for selecting a specific revision Note: the current valid selection is also displayed next to the SVN repository URL.	
“HEAD”	: The HEAD revision is selected. This is the latest revision (top revision) within a branch.
“Revision”	: A specific revision is selected by the revision number. Example: `3`
“Date”	: The specific revision is selected by the modification date. Example: `12/23/2016 11:59:59`
“Use UTC Time”:	: Modification date in universal time.

Table 774: “compare options”

“Ignore Whitespace”	: No comparison of whitespace characters. Semantically relevant whitespaces, such as in strings, are compared anyway.
“Ignore Comments”	: No comparison of comments.
“Ignore Properties”	: No comparison of properties. Folders, the property *“Exclude from build”*, and POU images are not compared. See: Dialog 'Properties'

“OK”	Compares the SVN project with the working copy.

See also

- *Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 4238*
- *Chapter 1.4.1.20.4.10 “Dialog 'Properties'” on page 1157*

Command 'Include externals to project', Command 'Include externals'

Symbol:

Function: These commands open the dialog *“Include externals”*.

Call:

- Menu bar: *“Project ➔ SVN”*.
- Context menu: *“SVN”*

Requirement: An object is selected in the object tree. The external objects are linked below that. If you have selected nothing or the project root directory, then the command *“Include externals to project”* is available. If you have selected an object, then the command *“Include externals”* is available.

The same external objects cannot be linked multiple times at different locations in the same project. This leads to problems in CODESYS because of conflicts with the internal identification of the object.

Dialog 'Include externals'

Table 775: "URL of SVN repository"

file:///D:/SVN repo A/trunk/DSTest.project/Source	URL of the external object that is linked. The object to be linked is versioned and can have sub-objects. External objects are located at another location in the SVN repository than the project. It can even be in another SVN repository. Example: `file:///D:/SVN repo A/trunk/DSTest.project/GlobalTextList` Note: The objects that should be linked below the selected object must have a matching object type. For example, only a task can be linked below the *"Task configuration"* object.
15	Opens the dialog *"Select revision"*. Here you can select a revision. The button is labeled with the currently selected revision: ● *"HEAD"*: Top revision (latest). Preset ● *"15"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) Note: The dialog provides the same options as the *"Revision"* group.
...	Opens the *"SVN repository browser"* dialog Here you can browse the SVN repository.

Table 776: "Revision"

Options for selecting a revision Note: the current valid selection is also displayed in the buttons next to the SVN repository URL.	
"HEAD"	: Latest revision (top revision) selected in a branch.
"Revision"	: A specific revision by the revision number. Example: `3`
"Date"	: A specific revision by the modification date. Example: `12/23/2016 11:59:59`
"Use UTC Time":	: Modification date in universal time.

"OK"	Adds the external object and its sub-objects with the property `svn:externals` to your project (below the selected object). The working copy is updated and the external object is overlaid with the ↗ symbol. Example: Source (external device `Source`) Note: If the linking fails (for example when adding a device below a task configuration), then the complete operation fails and reverts back. Note: Renaming or moving individual external objects is permitted inly within an external tree, whereby it is not permitted to move the top object. To move a complete tree, you have to remove it and link it to another location.

> *"... You should seriously consider using explicit revision numbers in all of your externals definitions. Doing so means that you get to decide when to pull down a different snapshot of external information, and exactly which snapshot to pull. Besides avoiding the surprise of getting changes to third-party repositories that you might not have any control over, using explicit revision numbers also means that as you backdate your working copy to a previous revision, your externals definitions will also revert to the way they looked in that previous revision, which in turn means that the external working copies will be updated to match the way they looked back when your repository was at that previous revision. For software projects, this could be the difference between a successful and a failed build of an older snapshot of your complex codebase. ..."*
>
> *This is a quote from:*
>
> *http://svnbook.red-bean.com/nightly/en/svn.advanced.externals.html).*

Command 'Ignore on commit'

Function: This command identifies an object and adds it to the "ignore-on-commit" list. Then it is deactivated in the commit dialog by default.

Call: Menu bar: *"SVN"*

Requirement: At least one object is available that is not in the change list `ignore-on-comment`.

Objects of the "ignore-on-commit" list are overlaid with the symbol in the object tree. By default, they are not selected in the commit dialog, unless a dependency of a selected object requires it. These objects can always be selected manually in the dialog.

See also

- *Chapter 1.8.5.5.2.26 "Command 'Un-Ignore on commit'" on page 4259*

Command 'SVN Info'

Function: This command provides information about the selected object in the SVN repository. The *"SVN Information"* dialog opens for this purpose.

Call: Context menu: *"SVN"*

Requirement: A versioned object (with SVN link) is selected in the object tree.

Dialog 'SVN Information'

Example

```
Name: Device_4\Plc Logic\Application\PLC_PRG
URL: file:///D:/SVN repository/trunk/ControlABC.project/Device/Plc
Logic/Application/PLC_PRG/svnobj
Repository Root: file:///D:/SVN repository/
Repository UUID: 185325d7-73eb-e54b-ab50-206aa23c8b42
Revision: 29
Node Kind: File
Schedule: Normal
Last Changed Author: a.mayer
Last Changed Rev: 8
Last Changed Date: 17.01.2017 12:33:51
Text Last Updated: 17.01.2017 12:33:51
Checksum: d5fb4d91ebaea06f26bcdb15942724d57932b6a3
```

Command 'Show properties'

Symbol: 

Function: This command opens the *"SVN Properties"* dialog. Here you can edit the properties of the versioned object.

Call: Context menu: *"SVN"*

Requirement: A versioned, unlocked object is selected.

Dialog 'SVN Properties'

Table 777: "properties for: <object name>"

"Name"	Name of the property Example: `myprop:customer-number` Note: SVN has some reserved properties. Example: `svn:mime-type`
"Value"	Example: `1234` Double-click in the field to edit the value.
"Add"	Opens a dialog to define another property with its value.
"Remove"	Deletes the selected property.
"Show binary properties"	☑: The binary properties are also displayed.
"Reset"	Resets the changes displayed in green.
"OK"	Accepts the changes.

See also

- *http://svnbook.red-bean.com*.

Command 'Get lock'

Symbol: 

Function: This command locks the object explicitly for you. The *"Lock Message"* dialog opens for this purpose.

Call: Context menu: *"SVN"*

Requirement: The versioned object is not locked (not overlaid with the symbol).

Dialog 'Lock Message'

"Enter the reason why you lock the object:"	Lock message Example: `Locked for processing task 123`
Button *"Recent Message"*	Shows message in the dialog that have already been used. There you select one in order to use the lock message.
"Recursive"	☑: The object is locked with all subordinate child objects.
"OK"	Locks the object When the lock is successful, the object (in the object tree) is overlaid with the  symbol.

Command 'Steal locks'

Symbol: 

Function: This command steal the lock of the object. The *"Lock Message"* dialog opens for this purpose.

Call: Context menu: *"SVN"*

Requirement: The versioned object is locked by someone else (overlaid with the  symbol).

Dialog 'Lock Message'

"Enter the reason why you lock the object:"	Lock message Example: `a.mayer had to steal the lock because the changes need to be implemented so urgently.`
"Recent Message"	Shows message in the dialog that have already been used. There you select one in order to use the lock message.
"Recursive"	 : The lock is stolen by the object and all subordinate child objects.
"OK"	Steals the lock. When the stolen lock is successful, the object (in the object tree) is overlaid with the  symbol.

Command 'Release lock'

Symbol: 

Function: This command releases the lock of an object.

Call: *"Context menu ➔ SVN"*

Requirement: The object is locked.

Command 'Release locks recursively'

Symbol: 

Function: This command releases the lock of an object explicitly with all of its subordinate objects.

Call: *"Context menu ➔ SVN"*

Requirement: The object is locked.

Command 'Show log', Command 'Show project log'

Symbol: 

Function: These commands open the tab *"Project log"* or *"Log - <object>"*. The version history of the project or an object of the CODESYS project is displayed in the tab.

Call:

- Menu bar: *"Project ➔ SVN"*.
- *"Context menu ➔ SVN"*

If you select nothing or the base node in the object tree, then the history of the entire project is displayed (*"Show project log"*). If you select one or more objects, then the history of these elements is displayed (*"Show log"*).

Multiple tabs can be open at the same time with the version history of different objects.

Tab 'Project log', Dialog 'Log - <object>'

Upper area ● *"Revision"*: Revision number ● *"Author"* ● *"Date"* ● *"Message"*: Message entered at commit	List of all revisions of the project or the selected objects in the information. The first 100 revisions are displayed by default. The *"Next 100"* and *"All"* buttons are provided for displaying more or all revisions. Several commands are available in the context menu of each revision. These context menu commands are described below.
Middle area	Display of the *"Message"* of the revision that is selected in the upper area.
Lower area ● *"Action"* ● *"Path"*: Object path in SVN ● *"Copy from path"* ● *"Copy from revision"*	List of actions that were performed on the objects of the project in the selected revision:
"Hide unrelated changed paths"	☑: All changes of this revision are hidden that do not have any relevance to the object.
"Stop on copy/rename"	☑: If the object was copied from another location in the SVN repository, then no more log messages are retrieved. This is especially beneficial when branches or tags are monitored and only changes within the branch are relevant.
"Filter/Range"	Opens the *"Filter"* dialog
"All"	All revisions are listed.
"Next 100"	The next 100 revisions are listed.

Table 778: Dialog "Filter"

"Revision range"	The displayed revisions can be filtered by *"Head"*, *"Revision"*, or *"Date"*. ☑: The option fields for *"Start revision"* and *"End revision"* are editable. *"Use UTC time"*: Date display in universal time. For more detailed information, refer to the description "Dialog 'Select revision'".
"Message contains"	Display of revision logs that contain a special text in the *"Message"*
" Author contains"	Display of revision logs of the specified author
"Path contains"	Display of revision logs of the specified path

Table 779: Context menu commands of the revisions

"Compare with base working copy"	Compares the selected revision of the object with the base working copy (without local changes).
"Com with working copy"	Compares the selected revision of the object with the working copy.
"Compare with HEAD revision"	Compares the selected revision of the object with the HEAD revision.
"Compare with previous revision"	Compares the selected revision of the object with the previous revision.
"Update item to revision"	Updates the object to the selected revision. Note: Changes of the project by this command cannot be committed. For VSS users: This is comparable to loading an older version without checkout. To revert a previous commit, the command *"Revert to this revision"* has to be used.

"Revert to this revision"	Reverts the object to the selected revision. This command does not have an effect on the SVN repository as long as the changes are not committed. Internally, SVN reverts the merges for all changes that were made after the selected revision in order to revert the changes of the preceding commits.
"Edit author"	Opens a dialog for changing the author of the revision.
"Edit log message"	Opens a dialog for changing the log message of the revision.
"Revision properties"	Opens the dialog *"Revision properties"* where the properties are displayed. In the dialog, you can activate the *"Add"* and *"Remove"* properties and the option *"Show binary properties"*.
"Create branch/tag from this revision"	Creates a branch or tag from the selected revision.
"Browse SVN repository"	Opens the *"SVN repository browser"* dialog
"Copy to clipboard"	Copies log details of the selected revision to the clipboard This is the revision number, author, date of revision, log message, and the list of changes objects for each revision.

See also

- *Chapter 1.8.5.5.2.6 "Command 'Compare'" on page 4247*
- *Chapter 1.8.5.5.2.7 "Command 'Compare with HEAD revision'" on page 4247*
- *Chapter 1.8.5.5.2.8 "Command 'Compare with revision'" on page 4247*
- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 4238*
- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 4267*

Command 'Revert', Command 'Revert project'

Symbol: 

Function: This command opens the *"Revert"* dialog. In the dialog, select the objects whose local changes should be reverted, and those that are reverted to the state of the base revision of the working copy.

Call:

- Menu bar: *"Project ➔ SVN"*.
- *"Context menu ➔ SVN"*

If you select nothing or the main node in the device tree, then all modified objects are listed in this dialog (*"Revert project"*). If you selected one or more objects, then only the changes to this object are listed and recursively their sub-objects (*"Revert"*).

Dialog 'Revert'

"Group externals"	☑: The external definitions are grouped by their external storage locations.
"Keep locks"	☑: The lock is retained for all files that are modified by the revert command.
"Select/deselect all"	

When external objects are deleted, Professional Version Control cannot restore this data in SVN offline mode. The user is prompted how to proceed:

- Switch back to SVN online mode and call the external objects.
- Connect now to the SVN server one time in order to complete the current operation, but afterwards switch back to SVN offline mode.
- Skip the retrieval of the external objects. They can be fetched later by updating the project.

See also

- *Chapter 1.8.5.5.2.20 “Command 'Revert to revision', Command 'Revert project to revision'” on page 4256*

Command 'Revert to revision', Command 'Revert project to revision'

Symbol:

Function: This command opens the *“Select revision”* dialog. In this dialog, you select the revision to which the project or the selected objects revert.

Call:

- *“Project ➔ SVN”*
- *“Context menu ➔ SVN”*

If nothing or the base node is marked in the object tree, then the entire project is reverted to a specific revision (*“Revert project to revision”*). If one or more objects are selected, then these objects and their sub-objects are reverted (*“Revert to revision”*).

Dialog 'Select revision'

For a description of the dialog, refer to the section "Select revision".

See also

- *Chapter 1.8.5.5.3.3 “Dialog 'Select revision'” on page 4267*
- *Chapter 1.8.5.5.2.19 “Command 'Revert', Command 'Revert project'” on page 4255*
- *Chapter 1.8.5.5.2.18 “Command 'Show log', Command 'Show project log'” on page 4253*

Command 'Update', Command 'Update project'

Symbol:

Function: This command commits changes in the SVN repository to the project. The update is performed with the HEAD revision.

Call:

- Menu bar: *“Project ➔ SVN”*.
- *“Context menu ➔ SVN”*

If nothing or the main node is selected, then the entire project is updated (*“Update project”*). If one or more objects are selected, then these objects and their sub-objects are updated (*“Update SVN”*).

The following cases are possible:

- Projects are added to the project that are present in the SVN repository, but not in the project. In this case, the message "Added <object>" is issued to the message view.
- Objects that no longer exist in the SVN repository, but are present in the project locally (and not marked as "added"), are treated according to the Subversion standard procedure: If local changes are present, then the object remains in the project as unversioned. If there are no local changes, then the object is also deleted locally because the user can retrieve the object from an older version at any time. In this case, "Deleted object" is issued to the message view.
- Versioned objects that exist in both the SVN repository and the project are updated if they are different. Three cases to observe:
  - No local changes have been made since the last update: In this case, the local object is overwritten by the contents from the SVN repository. The message *"Object updated"* is issued to the message view.
  - Local changes have been made since the last update and the corresponding object type can be merged. When versions have been merged successfully, the message *"Objects merged"* is issued to the message view. If the command is not executed successfully, then the object is marked as "Conflicted object" in the object tree and the message *"Conflicted object"* is issued.
  - Local changes have been made since the last update and the corresponding object type cannot be merged. In this case, the object is marked as "Conflicted object" in the object tree and the message *"Conflicted object"* is issued.

If only some of the objects are updated, it may be that objects with the same name already exist. For example, this situation can come from moving objects to a folder.

For this conflict, you can react in the following ways:

- Do nothing and leave the conflict-causing objects as they are.
- Update (and remove) the conflicting objects in order to correct the conflict.
- Update the entire project in order to remove all conflicting objects and correct the conflict.

See also

- *Chapter 1.8.5.5.2.22 "Command 'Update to revision'" on page 4257*

Command 'Update to revision'

Symbol:

S**Function**: This command opens the *"Update"* dialog. In the dialog, the revision is defined for updating the project.

Call:

- *"Project ➔ SVN"*
- *"Context menu ➔ SVN"*

If you select nothing or the base node in the object tree, then the entire project is updated to a revision (*"Update project to revision"*). If you select one or more objects, then these objects are updated and their sub-objects are updated recursively (*"Update to revision"*). As an option, you can define that the sub-objects are not updated.

The behavior of the updating process (for example merging of conflicts) is similar to the *"Update project"* and *"Update"* commands.

Dialog 'Update'

"HEAD"	: This command behaves the same as the *"Update"* and *"Update project"* commands.
"Revision"	: The revision to which was last updated is selected by the revision number. ...: Opens the dialog *"Log"* for selecting the revision.

"Date"	: The revision to which was last updated is selected by the modification date. *"Use UTC time"*: : The date is displayed in universal time.
"Recursive"	: Default setting. The selected part is updated recursively. This means that all elements below the selected object are also updated.
"Omit external objects"	: External objects are not updated.

See also

- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 4256*
- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 4267*

Command 'Update only this'

Symbol:

Function: The command updates the selected objects. In contrast to the *"Update"* and *"Update to Revision"* commands, the child objects are not updated.

Call: *"Context menu ➔ SVN"*

See also

- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 4256*
- *Chapter 1.8.5.5.2.22 "Command 'Update to revision'" on page 4257*

Command 'Disconnect project from SVN'

Symbol:

Function: This command deletes all connections of the current project to SVN by converting the project into a non-versioned project.

Call: Menu bar: *"Project ➔ SVN"*.

Because this operation cannot be reversed, the operation must be confirmed before the command is executed.

Use the command "Connect to existing project" to connect to the SVN repository again at a later time.

See also

- *Chapter 1.8.5.5.2.30 "Command 'Connect to existing project'" on page 4261*

Command 'Switch'

Symbol:

Function: This command opens the *"SVN switch"* dialog. In this dialog, you specify a URL in the SVN repository to which the current working copy of the project is updated. The command switches a project from a branch or tag to another.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement: The project is versioned.

Dialog 'SVN switch'

"From"	Current SVN URL of the project
"To"	Input field for the target URL in SVN ● *"HEAD"*: The *"Select revision"* dialog opens. ● ...: The *"SVN Repository Browser"* dialog opens. There you select the target URL in the SVN repository.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 4238*

Command 'Un-Ignore on commit'

Function: This command removes an unversioned object from the ignore list so that the object is checked by default on commit.

Call: Context menu: *"SVN"*

Requirement: The command *"Ignore on commit"* was executed for the object. The object is marked with the symbol.

See also

- *Chapter 1.8.5.5.2.11 "Command 'Ignore on commit'" on page 4251*

Command 'SVN Cleanup'

Function: This command opens the *"SVN Cleanup"* dialog. In the dialog, you define actions that are performed when cleaning up the SVN working copy.

Call: Menu bar: *"Project ➔ SVN"*.

Dialog 'SVN Cleanup'

Table 780

"Internal SVN working copy"	
"Update time stamps (speeds up SVN status display)"	☑: Corrects recorded time stamps for unchanged files in the working directory. This leads to a reduction in the compare time for future checks. It is not necessary to execute this in regular intervals in the normal workflow.
"Vacuum cached pristine copies (may reduce the size of your project file)"	☑: Cleans the buffer for the original copies by deleting older versions that are no longer referenced by the current project. Advantage: The size of the project file is reduced. Disadvantage: If you downgrade to older revisions, or if you switch between different branches, then the retrieved data size will become larger.
"Clear work queue and force unlock of SVN internal data structures (emergency only!)"	☑: Cleans up the internal SVN task queues and unlocks internal SVN data structures. This should never be necessary during normal work by Professional Version Control. Note: Use this option only if errors occur for SVN commands due to locked working copies. When this is the case, it refers to an error in Professional Version Control. Then please send us an error report (if possible with steps to repeat) to the CODESYS support. Info: These are administrative locks that are internal locks in the SVN working copy. These locks are not set up by context menu commands. For more information, refer to the section "The three meanings of locks" in: *http://svnbook.red-bean.com/en/1.8/svn.advanced.locking.html*
"Project contents"	

“Revert all local changes (use with care!)”	Reverts all local changes to the original status in the SVN repository.
“Release all locks”	Releases all advisory locks in the project (locks visible to the user). These locks are activated by *“Acquire lock”* and *“Steal lock”*.
“Revalidate all locks against the repository (they could have been stolen)”	Checks whether the locally available advisory locks are still valid or have been stolen by someone else for example. All invalid locks are removed.
“Status caches”	
“Clear all caches and refresh status icons”	Deletes all internal caches that Professional Version Control has and updates the status icons. Required only if it issues an error in Professional Version Control through which the caches or the status display are inconsistent.

Command 'Clear authentication data'

Function: This command opens the *“CODESYS”* dialog. In this dialog, define the caches that will be deleted.

Call: Menu bar: *“Project ➔ SVN”*.

Dialog 'CODE-SYS'

The authentication memory contains the authentication data of all SVN repositories for which the user has selected for saving the authorization data. This memory is deleted completely by this command.	
“Clear the shared on-disk cache.”	☑: The data saved on the computer is deleted.
“Clear the RAM cache of this instance.”	☑: The data saved in the RAM is deleted.

The authentication data saved on the computer is stored in `%APPDATA%\Subversion\auth`. This memory path is also used for most other Subversion client applications (for example, TortoiseSVN and AnkhSVN). Therefore, deleting the authentication data affects these applications as well.

Command 'Merge changes'

Symbol: 

Function: This command opens the *“Merge”* dialog. In this dialog, you determine the revisions with the changes to be merged with the working copy of the project.

Call: Menu bar: *“Project ➔ SVN”*.

Requirement: The project is linked to SVN.

Dialog 'Merge'“”

"Kind of merge"	• *"Sync/Reintegrate/Symmetric merge"*: Synchronizes all missing changes from trunk (or a different branch) into this branch. • *"Cherry pick"*: Integrates specifically selected revisions from one branch to another branch. This is necessary, for example, if any error trapping has to be ported back to an older version.
"Merge source"	SVN URL of the SVN repository • Input field • *"HEAD"*: HEAD revision • ...: Dialog *"SVN Repository Browser"* opens for selecting the SVN repository.
"Define start and end revision"	Select this option to merge a cohesive range of revisions with the working copy.
"Start revision"	Defines the range of revisions that are merged with the working copy: • *"HEAD"*: HEAD revision • *"Revision"*: Start and end revision of the range • *"Date"*: Date of the start and end revisions
"End revision"	
"Define revision range"	Select this option to merge individual revisions with the working copy. You can also highlight the individual revisions in the *"Log"* dialog. Note: When defining ranges, CODESYS SVN behaves like other graphical clients, such as Tortoise SVN), and not like the command-line client. Example: For a range of `4-7`, revisions 4, 5, 6, and 7 are merged. See also: *Merging a Range of Revisions*
"Dry run (simulation)"	☑: This command is executed without changing the working copy. Files that are changed during an actual merge are displayed, as well as ranges where conflicts occur.
"Record only"	☑: The revision is marked as "merged" without actually performing the merge.
"Ignore ancestry "	☑: SVN uses path-based differences only, not history-based differences.

See also

- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 4267*
- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 4238*

Command 'Connect to existing project'

Symbol:

Function: This command opens the *"Connect to SVN repository"* dialog. In the dialog, you define the URL and the revision of the SVN repository with which the unversioned project is connected.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement: The project is disconnected from SVN.

NOTICE!
Only users who have read access to the entire project (see the CODESYS user and access management) can import the project into the SVN repository or can link to an existing database project.

NOTICE!
This command functions reliably only when the project has already been imported into SVN and then disconnected with the command *"Disconnect project from SVN"*.

Dialog 'Connect to SVN repository'

"URL of existing project"	URL of the SVN repository *"HEAD"*: Selection of the revision in the *"Select revision"* dialog ...: Selection of the SVN repository in the *"SVN Repository Browser"*
"Checkout options"	*"Omit externals"*: External objects are not checked out.
"Revision"	● *"HEAD"*: HEAD revision ● *"Revision"*: Number of the revision ● *"Date"*: Date of the revision *"Use UTC time"*: ☑: Date display in universal time.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 4238*

Command 'Resolve conflict'

Symbol: 🔍

Function: This command opens the *"<object>"* dialog. In the dialog, the conflicts are displayed and functions for resolving conflicts are prepared in order to merge changes.

Call: Context menu of the object.

Requirement: The object has a conflict that has occurred by updating the object with local changes.

Dialog '<object>'

"Compare"	The local objects are displayed on the left side, and the version from the SVN repository is displayed on the right side.
"Use mine"	A local change is used.
"Use yours"	A change of the version from the SVN repository is changed.
"Apply"	All changes are accepted that you made in this dialog. The status of the object is changed.
"Cancel"	Cancels all changes that you made in this dialog. But the object keeps the conflicted status.

Command 'Work in offline mode'

Function: This command switches to SVN offline mode. In SVN offline mode, the implicit locking and all commands that access the SVN repository are not possible.

Call:

- Menu bar: *"Project ➔ SVN"*.
- Context menu: *"SVN"*

Requirement: The project is linked to SVN.

When switching back to SVN online mode, all present locks on the working copy are checked against the server. If this locking is invalid, then it is released.

Uses case

The user on a machine wants to make changes to the project without disconnecting the connection. At the moment, there is not connection to the server. Despite this, when automatic locking is activated, work is possible because the SVN offline mode deactivates the automatic lock temporarily.

Command 'Copy (Branch/Tag)'

Symbol: 

Function: This command opens the *"SVN Copy Branch/Tag"* dialog. There you can *"Branch"* or *"Tag"* a revision of your project. A specific revision of your project is saved there at this position. A branch is normally used in order to save changes isolated in one version. A tag is used for marking a specific state, for example a shipping version. Internally, it is copied not in the actual sense, but more refers to the revision.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement: The project is versioned.

Dialog 'SVN Copy (Branch/Tag)'

Table 781: "SVN repository"

"From"	SVN path of the current project Example: `https://svnserver/repository/trunk/ControlABC.project`
"To"	Target path in the SVN repository for the copy operation Example of tag: `https://svnserver/repository/tags/V4.4.4.4/ControlABC.project` ...: Dialog *"SVN Repository Browser"* opens for selecting the target path.

Table 782: "Log message"

Input field	Comment the change in a log message. Example: `Tag for version 4.4.4.4 created.`
" Recent Messages"	Opens the dialog *"Recent Messages"* to display the last log messages. You can click a log message to accept it.

Table 783: "Create copy from"

"Working copy (including local changes)"	The new branch/tag refers to the working copy including all local changes. The local changes are committed to the SVN repository for this purpose.
"Base revision of working copy (<revision number>)"	The new branch/tag refers to the base revision of your working copy whose revision number is displayed in the parentheses. If the working copy already contains local changes, then these are not committed to the SVN repository.
"HEAD revision of the repository"	The new branch/tag refers to the HEAD revision of your project.
"Specific revision in SVN repository" 	The new branch/tag refers to a revision that is displayed on the adjacent button. Click the button to change the revision. The *" dialog opens."*.

"Switch to new location "	☑: After the dialog is confirmed, the working copy switches to the new branch/ tag.

"OK"	The target path is created (as a new tag `../repository/tags/V4.4.4.4` or as a new branch `../repository/branches/new_feature`). Then the revision specified in *"Create copy from"* is copied there.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 4238*

Command 'Pending Changes'

Symbol:

Function: The command opens the *"Pending Changes"* view. All objects are listed there which have changed from the base revision or which are locked.

Call: *"View ➔ Pending Changes"*

View 'Pending Changes'

The modified or locked objects are shown in the lower half of the view. You can use the *"Commit"*, *"Revert"*, and *"Update"* commands on single or multiple objects. You will find commands for comparing and displaying the version history in the context menu of a selected object.

Double-clicking the object opens the project comparison.

"Select"	Selection or clearing of all objects
"Commit"	Commits local changes to the SVN repository
"Revert"	Reverts the local changes to the state of the base revision of the working copy
"Update"	The command commits changes in the SVN repository to the project. The update is performed with the HEAD revision.
"Keep Locks"	☑ Lock is not released automatically after commit
"Recent Messages"	Shows the last used log messages. You can click a log message to accept it.
"Messages"	Type in a log message that comments your change. Example: `Bug fix error 123`

See also

- *Chapter 1.8.5.5.2.5 "Command 'Commit', Command 'Commit Project'" on page 4244*
- *Chapter 1.8.5.5.2.19 "Command 'Revert', Command 'Revert project'" on page 4255*
- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 4256*

1.8.5.5.3 Dialogs

Dialog 'Options' - 'SVN Settings'

Tab 'General' Symbol: 

Function: This tab includes the basic settings for Professional Version Control.

Call: Menu bar: *"Tools ➔ Options"*.

Table 784: "Automatic locking and merging"

"Merge"	Behavior for the commands *"Update"*, *"Merge"*, or *"Switch"*, when both sides (working copy and SVN repository) have changed from the base version. ● *"Mark all colliding changes as conflicts"*: The objects are not merged automatically. All changes with a conflict are marked as "With conflict", even if some of them can be merged automatically. ● *"Merge mergeable changes, mark the others as conflicts"*: Changes that can be merged are merged automatically. All others are marked as "With conflict". ● *"Merge mergeable changes, ask the user for the others"*: Changes that can be merged are merged automatically. The user is prompted for all others. ● *"Always ask the user, even for mergeable changes"*: For all changed objects, the user is prompted, even if some of them can be merged automatically.
"Locks"	Behavior such as Professional Version Control objects when they are changed locally. ● *"Always try to lock before modification"*: All objects are locked before they are changed, even if they can be merged. ● *"Only lock the objects which don't support merging"*: Only those objects are locked that cannot be merged automatically. ● *"Never acquire a lock automatically"*: No objects are locked, not even if they can be merged automatically.
"Marker"	● *"Use conflict markers when merging objects"*: If objects with conflicts exist that cannot be merged, then these conflicts are marked in the source code with conflict markers. In addition, the object itself is marked as being merged successfully (no conflict). ● *"Leave non-mergeable objects as conflicted"*: No conflict marker is set. Objects that cannot be merged remain in the status "With conflict".
"Prompt the user when automatic locking fails."	☑: If it is not possible, to lock the object, then the dialog *"Automatic locking failed"* opens (see dialog description).

Example of conflict markers

```
Foo();
>>>>>>>>>>>>>>>
I := I + 1;
===============
I := I + 2;
<<<<<<<<<<<<<<<
```

Table 785: "Server check"

"Check server for updates and locks"	☑: Professional Version Control checks in the specified time interval that objects have been updated on the server. In addition, it checks whether objects are locked or locks have been stolen.
"Check interval (minutes)"	Example: 10

Table 786: “Ignore for comparison”

Ignore whitespace	☑: Whitespace differences between the current project and the reference project are ignored.
Ignore comments	☑: Comments in the programming code are excluded from the comparison.
Ignore Properties	☑: Object properties are excluded from the comparison.

> *Some of the SVN options can be overwritten by the project-specific settings.*
>
> *Project-specific settings are defined in the menu “Project ➔ Project settings”, category “SVN Settings”.*

See also

- *Chapter 1.8.5.5.3.5 “Dialog 'Automatic locking failed'” on page 4270*
- *Chapter 1.8.5.5.3.2 “Dialog 'Project Settings' - 'SVN Settings'” on page 4266*

Tab 'SSH'

Symbol:

Function: This tab contains the settings for the SSH protocol.

Call: Menu bar: *“Tools ➔ Options”*.

Table 787: “SSH client implementation”

“libssh2 (recommended)”	Professional Version Control uses Libssh2 for establishing a connection via SSH protocol. This is the recommended setting.
“SharpPlink (backwards compatibility)”	Professional Version Control uses plink.exe for establishing a connection with SSH servers. This option is required only for communication with outdated servers that support the deprecated SSH-1 protocol.

> *The SSH configuration can be overwritten by means of the environment variable `SVN_SSH` or server-specific by means of the SVN configuration file.*

See also

- *Tunneling via SSH*

Dialog 'Project Settings' - 'SVN Settings'

Symbol:

Function: The behavior of the integrated SVN version control system is configured in this dialog.

Call: Menu bar: *“Project ➔ Project Settings”* (*“SVN Settings”*).

Requirement: A project is open.

Table 788: “Automatic locking and merging”

With these settings, you can overwrite the default settings that were made in the dialog *“Tools ➔ Options”*, category *“SVN Settings”*.	
“Merge”	Behavior for the commands *“Update”*, *“Merge”*, or *“Switch”*, when both sides (working copy and SVN repository) have changed from the base version.

“Locks”	Behavior such as Professional Version Control objects when they are changed locally.
“Marker”	Behavior for conflicts

Table 789: “Settings SVN version info”

“Create SVN_VERSION_INFO constants for IEC access”	☑: The object `SVN_VERSION_INFO` is created and includes global constants or variables for the project metadata. ☐: The object `SVN_VERSION_INFO` is not available. When you activate the option, the object is created automatically. When you deactivate the option, the object is removed from the project automatically.

See also

- *Chapter 1.8.5.5.3.1 “Dialog 'Options' - 'SVN Settings'” on page 4265*

Dialog 'Select revision'

Function: This dialog shows the currently selected revision. You can edit the selection there.

“Revision”	
“HEAD”	◉: The latest revision (top revision) within a branch is displayed.
“Revision”	◉: A specific revision is displayed by the revision number. Example: `3` Tip: Click [...] to show the revisions. Then the *“Log”* dialog opens to display the revisions and the associated actions. The revision that you select there is applied.
“Date”	◉: A specific revision is checked out by the modification date. This is the highest revision at the given time (the last revision before that time). Example: `12/23/2016 11:59:59` Tip: See section "Revision identifiers" in "Version control with Subversion"
“Use UTC Time”:	☑: Modification date in universal time is used.
“Reset recursively”	☑: All objects below the selected object are also reset. The action fails if ● Objects have been moved in or out of the hierarchy below ● Objects outside of the hierarchy would be changed by implicit dependencies

See also

- *Chapter 1.8.5.5.2.18 “Command 'Show log', Command 'Show project log'” on page 4253*
- *"Version control with Subversion", Section "Revision identifier"*

Dialog 'Subversion Authentication'

The dialogs are used for authenticating the server/client connection. A server or client authentication is performed depending on the initial situation and protocol.

Overview of possible protocols and dialogs

- `svn://`: The SVN protocol; either unencrypted or SSL/TLS encrypted
  - Can prompt for user name and password (even for an unencrypted connection)
  - Can prompt for a server certificate from the dialog for authentication in order to confirm the server if a certificate is unknown, defective, or invalid (for TLS/SSL encryption)
  - As an alternative or in addition to the user name and password prompt, the client can also be authenticated with client certificates (for TLS/SSL encryption). The dialogs for authentication open with the client certificate.
- `http://`: SVN via http, unencrypted
  - Can prompt for user name and password
- `https://`: SVN via http, SSL/TLS encrypted.
  - Can prompt for user name and password
  - Can prompt for a server certificate from the dialog for authentication in order to confirm the server if a certificate is unknown, defective, or invalid.
  - As an alternative or in addition to the user name and password, the client can also be authenticated with client certificates. The dialogs for authentication open with the client certificate.
- `svn+ssh://`: The SVN protocol, encrypted through an SSH tunnel. SSH (Secure Shell) is the usual networking tool in Linux/Unix for accessing other computers.
  - Can prompt for user name and password
  - Prompts for server certificate in the dialog for authentication if the server is still unknown in order to be sure that it is the correct server.

Dialog for authentication with a server certificate

Initial situation: CODESYS (as a client) receives an unknown or defective server certificate.

This dialog shows information about the certificate. There you can confirm the identity of the server.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

Table 790: “Certificate information” (for SSL/TLS connections)

“Host name”	Example: `svn repository`
“Thumbprint”	
“Valid from”	
“Valid to”	
“Issuer”	Example: `ABB AG`
“Certificate”	

Table 791: “SSH server key information” (for SSH connections)

“Key type”	
“Key size (bits)”	
“Key thumbprint”	

“Save information to RAM”	☑: The certificate is saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: The certificate is saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Authenticates and established the connection.

The certificate memory is secured cryptographically and distributed with other SVN clients.

See also

- *Version Control with Subversion*

Dialog for authentication with a client certificate

Initial situation: The SVN server requires a client certificate for authentication.

In this dialog, you select the client certificate in order to confirm the identity.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

Table 792: “The SSL server requires a client certificate file.”

“File”	Client certificate file

“Save information to RAM”	☑: The certificate is saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: The certificate is saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Authenticates and established the connection.

Dialog for authentication with a pass phrase

Initial situation: The SVN server is configured so that it demands a client certificate for authentication. The applied certificate is protected by a pass phrase.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

Table 793: “A pass phrase is needed to unlock the certificate.”

“Pass phrase”	Example: `***`

“Save information to RAM”	☑: The pass phrase is saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: The pass phrase is saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Authenticates with client certificates by means of a pass phrase and establishes the connection.

Dialog for authentication with a user name and password

Initial situation: The SVN server is configured so that it demands a user name and password for authentication.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

“User name”	Example: `a.mayr`
“Password”	Example: `***`

“Save information to RAM”	☑: Saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: Saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Establishes the connection and authenticates it.

Dialog 'Automatic locking failed'

The dialog shows a list of all objects for which an automatic locking was not possible. In the optoins you define how Professional Version Control will resolve the confilict.

Table 794: “Automatic Locking and Merging”

• *“Try to steal the lock for the affected objects”* • *“Activate the "Offline Mode" to temporarily suppress locking”*	These options are displayed if another user has locked the object.
• *“Update the affected objects to the newest revision”* • *“Update the whole project to the newest revision”* • *“Activate the "Offline Mode" to temporarily suppress locking”*	These options are displayed if there exists a more current version of the object on the server.
• *“Activate the "Offline Mode" to temporarily suppress locking”*	These options are displayed if no connection can be established to the server.
“SVN Project Settings”	Opens the SVN project settings dialog (menu *“Project ➔ Project Settings”*). There you can change the settings for the automatic locking.
“SVN Settings”	Opens the general SVN project settings dialog (menu *“Tools ➔ Options”*).

See also

- *Chapter 1.8.5.5.3.1 “Dialog 'Options' - 'SVN Settings'” on page 4265*
- *Chapter 1.8.5.5.3.2 “Dialog 'Project Settings' - 'SVN Settings'” on page 4266*

1.8.5.5.4 Objects

Object 'SVN_VERSION_INFO'

Symbol: 

The object contains the SVN metadata of the project as global constants or variables in a variable list. It is located in the *“POUs”* view. You can specifically retrieve the data of the global constants or variables by the application. By calling specific data, you can also reduce the memory usage on the controller.

The SVN metadata is provided for this purpose, subdivided over multiple global variable lists (GVLs):

- *“SVN_VERSION_INFO”*
- *“SVN_Info_Summary”*
- *“SVN_Info_SummaryW”*
- *“SVN_Info_URI”*
- *“SVN_Info_Revisions”*
- *“SVN_Info_Flags”*
- *“SVN_info_LastChange”*

The SVN_VERSION_INFO object is created automatically when a project is imported to a SVN repository. To do so the option *“Create SVN_VERSION_INFO”* in the dialog *“Import project to SVN”* must be activated.

Furthermore you can create the object or remove it from the project with the option *“Generate SVN_VERSION_INFO constants for IEC Access”* (Dialog *“Project ➔ Project Settings”*, category *“SVN Settings”*).

Table 795: Global Constants

Name	Data type	Description
MINREVISION	LINT	Lowest revision number of the working copy
MAXREVISION	LINT	Highest revision number of the working copy
PARTIAL	BOOL	TRUE: The working copy is incomplete. Example: Cancellation during the last update due to a network error or a checkout.
MODIFIED	BOOL	TRUE: Local changes were made.
SWITCHED	BOOL	TRUE: Parts of the project were branched (with the *"Switch"* command).
VERSION	STRING	Version identification, similar to Apache™ Subversion® (subversion.exe) Example: 12:34M, means MINREVISION = 12, MAXREVISION = 34, MODIFIED = TRUE For more information, refer to the documentation for Apache™ Subversion®.
CLEAN	BOOL	TRUE: The version is clean. This is the case when MINREVISION is equal to MAXREVISION, the working copy is complete, and non-versioned, and is was not switched.
URL	WSTRING	SVN-URL of the project Example: https://svnserver/repository/trunk/ControlABC.project

If a controller does not support the data type WSTRING, then a compiler error is issued when accessing the object SVN_VERSION_INFO.

See also

- *Chapter 1.8.5.5.3.2 "Dialog 'Project Settings' - 'SVN Settings'" on page 4266*
- *Chapter 1.8.5.5.2.3 "Command 'Import project to SVN'" on page 4242*

1.8.6 Subversion

1.8.6.1 Project Version Control with Subversion

Introduction

Automation Builder projects can be stored in Subversion (SVN) repositories by using the Project Version Control. The Project Version Control can be used to track changes on a project and to have access to historic versions of the whole project or objects in the project. It is possible to hold different versions of a project in branches and to compare these versions. The Project Version Control enables multiple engineers to work collaboratively on the same project.

Basic knowledge

Make yourself familiar with the concepts of SVN.

This manual about Project Version Control is additionally to the following information and describes mainly the specific behavior of Subversion in Automation Builder.

- Homepage of Subversion: *http://subversion.apache.org/*
- Online user manual for Subversion: *http://svnbook.red-bean.com/*
- Documentation on SVN integration in Automation Builder: Refer to subfolder .

1.8.6.1.1 Preconditions

Automation Builder

- In Automation Builder, the Project Version Control must be installed.
- A valid license for the Project Version Control must be activated.
- All collaborating users working on the same project need:
  - Automation Builder installed in the same version with the same features.
  - License for same edition.
  - Same set of optional third party device descriptions.
  - Same set of optional customer specific packages.

SVN server

The Project Version Control can be used in combination with an SVN server in version 1.6 or newer, the repository format should be 1.5, 1.6 or 1.7. Newer repository formats are not yet supported.

The usage of local repositories in the local file system or even on a network share is strongly discouraged.

1.8.6.1.2 Working with Project Version Control

- All objects in the device tree or POU tree are represented by an object in the SVN repository, there might be hidden objects that are not visible in the tree but that exist in SVN.
- The smallest unit in the SVN repository is one object including all its data like name, parameters, device identification.
- Objects are identified in the SVN repository by their name. Renaming one object in Automation Builder means to delete it from the SVN repository and add a new one to the SVN repository. Renaming an object causes a break in the history of that object.
- By default objects are locked before they are changed to prevent other users from changing the object. The locking strategy can be changed in the user options.
- Objects can be compared to other versions of the same object, many differences/changes between the current object in the Automation Builder project and the compared object can be merged into the object in the Automation Builder project. Merging changes could be used to resolve conflicts in case concurrent changed can not be avoided.

- To ensure consistency it is required and also enforced that some changes can be committed or reverted only together.
  - All changes to device objects in the hardware tree that are sub-nodes to the same top level device. Note: Objects that are not devices are excluded, e.g. the application node.
  - All changes below the AC500 PLC application node.
- Most SVN operations can not be performed while other external applications like CODESYS or Panel Builder work on files that are embedded in Automation Builder project.
- Some operations like changing the target or updating the project to the latest device (description) versions do a recursive lock of the whole AC500 PLC. If the lock can’t be obtained the operation is aborted.
- Some objects contain internal data that has no meaning to the end user but is also important. Changes on such data are not shown in the compare dialog or are summarized by a placeholder like "There are hidden changes".
- Including externals is not supported.

1.8.6.1.3 Recommendations on Working with Project Version Control

Be collaborative

- Multiple users that work collaboratively on the same project should agree on their responsibility for certain parts of the project where they do changes to avoid conflicts and tree conflicts.
- Agree on locking strategy used by all users working on the same project.
- Distribute the work between multiple users meaningful.
  - It is suggested to setup the hardware structure at first before other users checkout a project to work on it and limit structure changes in the hardware tree to the minimum.
  - Before adding objects, especially top level objects, users should agree that only one user adds objects at top level or below the same parent, or agree on unique names for the objects to add. The default naming scheme for new objects bears the risk of name conflicts. These conflicts could be resolved only by reverting the changes of the user who later tried to commit the changes.

Be careful

- The SVN integration (and also project compare) gives lot of power to the user, users should be sure to do only things they fully understand. Especially by merging changes incomplete it is possible to create inconsistent data.
- Adding devices, removing devices or even changing parameters can have side effects to other devices, do not change objects/parameters to their original state by merging that were not done explicitly.
- Commit changes frequently to SVN.
  - To release locks that you don’t longer need.
  - To reduce the risk of conflict with co-workers.
  - To keep the sets of changes to commit small.
- Do frequent updates when collaborating in a team.
  - To be up-to date.
  - To keep the sets of changes to get from SVN small.
  - To reduce the risk of losing work results in case of conflicts.
- To avoid conflicts, it is suggested to stay with the default setting to automatically lock objects before doing changes. Consider explicit recursive locks of sub-trees where you plan bigger changes.
- Prefer a clean checkout over using the switch command to change between different branches.
- Do not use the switch command to change between unrelated projects, this could corrupt the Automation Builder project (local copy, not in SVN) easily.
- Commit local changes to the SVN repository before creating a branch.

Be effective

- Give objects good/correct names after adding them and use renaming of objects already committed to SVN sparely to maintain a continuous history in the SVN repository.
- The goal to revert only single changes of all changes done that must be committed/reverted together, could be achieved by using project compare or the object compare dialog.
- If changes can’t be committed to the SVN repository because of locks hold by other users, it is possible to create a branch, use the switch command to change to this branch and commit the changes there. The branch and base line could be merged together later.

1.8.6.1.4 Known Issues and Troubleshooting

Not all changes are shown for all objects, but hidden changes are also important.

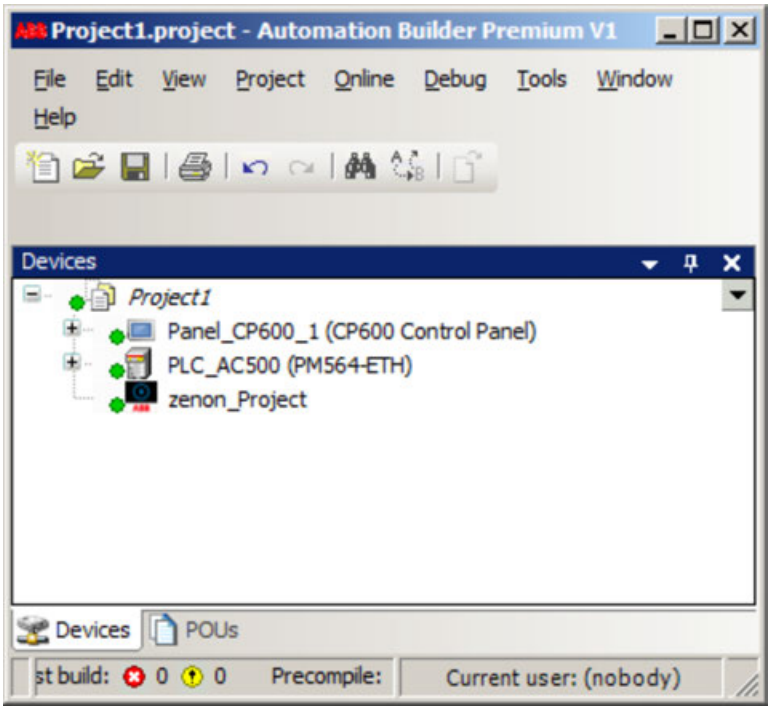
The device pool may be changed as side effect of several operations, including opening the project.

When a project was corrupted (by performing an update that tried to add an AC500 Communication Module) it is possible to save this project and merge changes to a project that has been cleanly checked out by project compare.

1.8.6.2 SVN Support Examples

1.8.6.2.1 Importing Automation Builder Project to SVN Repository

1. In the Automation Builder main menu, go to *"Project ➔ SVN ➔ Import Project to Subversion"*.
2. Enter user credentials and click *"OK"*.
3. Select SVN server repository to import Automation Builder project and click *"OK"*.
   ⇨ The Automation Builder project is imported into the selected repository and connected automatically to the repository. The imported project nodes are identified with green indicators.



1.8.6.2.2 Logging in User2

1. In the Automation Builder main menu, go to *"Project ➔ SVN ➔ Checkout"*.
2. Enter user credentials and click **OK**.
3. Select the repository location, project folder and revision if any and click *"OK"*.
   ⇨ The project will checked out of the repository, saved in the selected location and opened as a primary project.

The tables below provide the descriptions of the options availlable in the check-out dialog.

Checkout options	
Omit externals	Do not checkout external objects.
As library project	Saves the project as a CODESYS library file.

Revisions	
HEAD	Checks out the Head revision.
Revision	Allows to select the required revision of the project.
Date	Allows to select a revision date of the project.

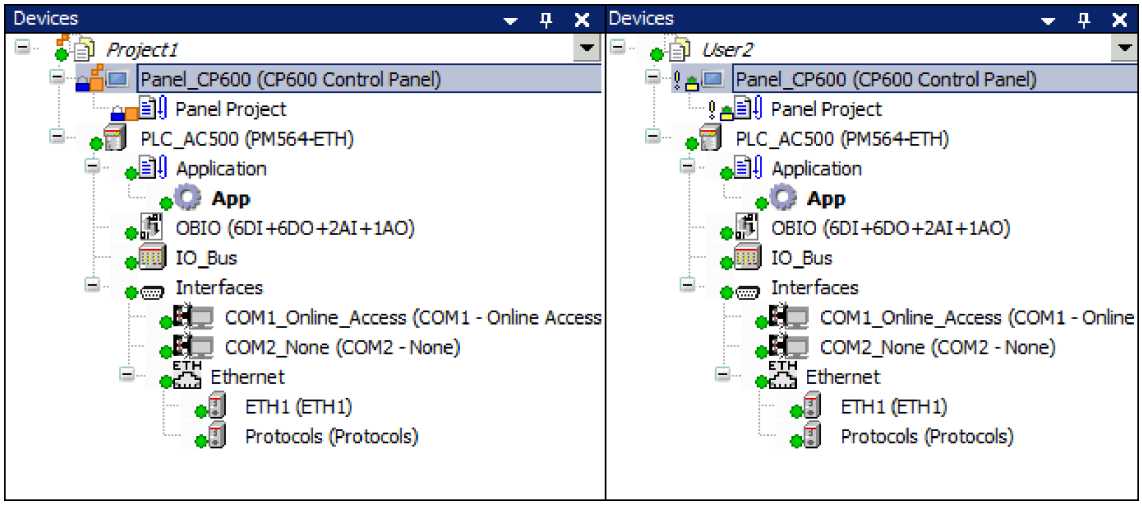
The following instances can occur in the workflow.

- If the project contains any updates, the specific project level node is indicated with .
- If a new object is added to the project, the newly added node is indicated with .
- If the project node is deleted, the specific node is indicated with .

1.8.6.2.3 Examples

Example 1

If User1 modifies Panel_CP600 project, then the node indicator turns to orange with lock symbols. If User2 need to modify the same Panel_CP600 project, the Panel_CP600 project appears with a lock symbol.

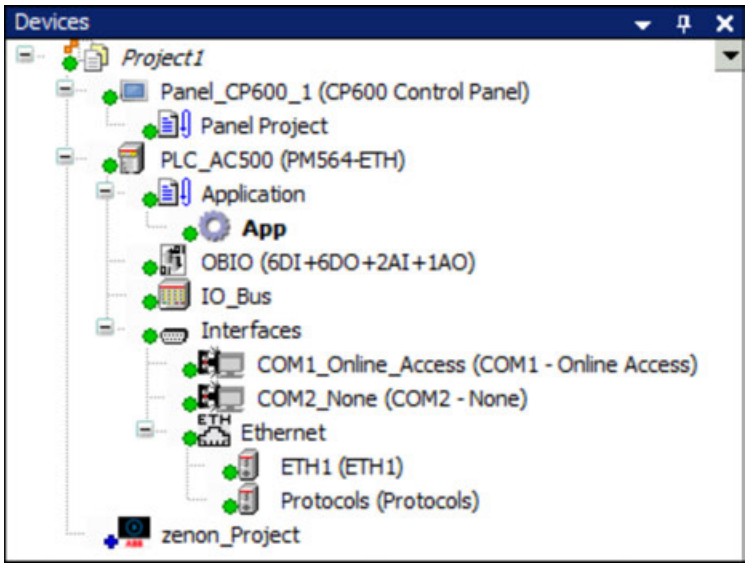


To steal the lock of an affected object, proceed as follows:

1. Double-click *"Panel_CP600"* project.
   ⇨ Automatic lock failed dialog is displayed.
2. Enable *"Try to steal the lock for the affected objects"* and click *"OK"* to steal the lock.
3. In the Lock Message window, enter the reason to steal the lock and click *"OK"*.
4. User2 can modify and commit the project.

Example 2

If User1 adds a new object to the project and commit the changes, then User2 can update the project to see the latest modifications.



Example 3

The user can revert to any of the available project revisions.

1. Right-click on object node and select *"SVN ➔ Revert to Revision"*.
2. Select or enter the revision number and click *"OK"*.
   ⇨ The revision command reverts local changes of this object back to the specific revision of the working copy.
3. Right-click on the object node and select *"SVN ➔ Commit"*.
4. In the commit window, enter the reason to change the project and click *"OK"* to make the changes.
   ⇨ The project node is updated with the latest changes.

Example 4

SVN server allows to select the required revisions of Automation Builder project. You can checkout the project using *"Project ➔ SVN ➔ Checkout"* and then enter the credentials and click *"OK"*.

In the check-out dialog, do the following:

1. Select the project repository.
2. Activate *"Revision"* and select or enter the revision number and click *"OK"*.

The user can work on the selected revision. To commit the changes to the project, right-click on project and select *"SVN ➔ Commit Project"*.

1.8.7 Python

1.8.7.1 Python script support

Using scripts

Scripting allows python scripts to be used to automate project configuration in Automation Builder. Parameters can be added to scripts, so that a generic script can be customized before execution. The user can add a script to most parts of the device tree. A script can be started either from the user interface (by a command or with the python scripting editor) or from the Windows command line and is saved with the project.

With the scripting feature commands or complex program operations can be automated.

Examples of use cases:

- Integration of Automation Builder in automatic build server environments:
  - continuous integration (CI)
  - continuous delivery (CD)
  - continuous testing
- Integration with third-party software, for example:
  - code generators
  - creation of projects that are custom tailored to a specific machine configuration
- Creation of documentation
- Updating of libraries: Setting of project information during the release process
- Automatic testing: Mostly in connection with the Professional Test Manager
- Outputting variables via monitoring APIs

Licensing

A valid license is required to use the scripting. If you open a project with the existing script object without a valid license, you are not allowed to add or edit the scripts. However, the scripts are not removed from the project.

Scripting language

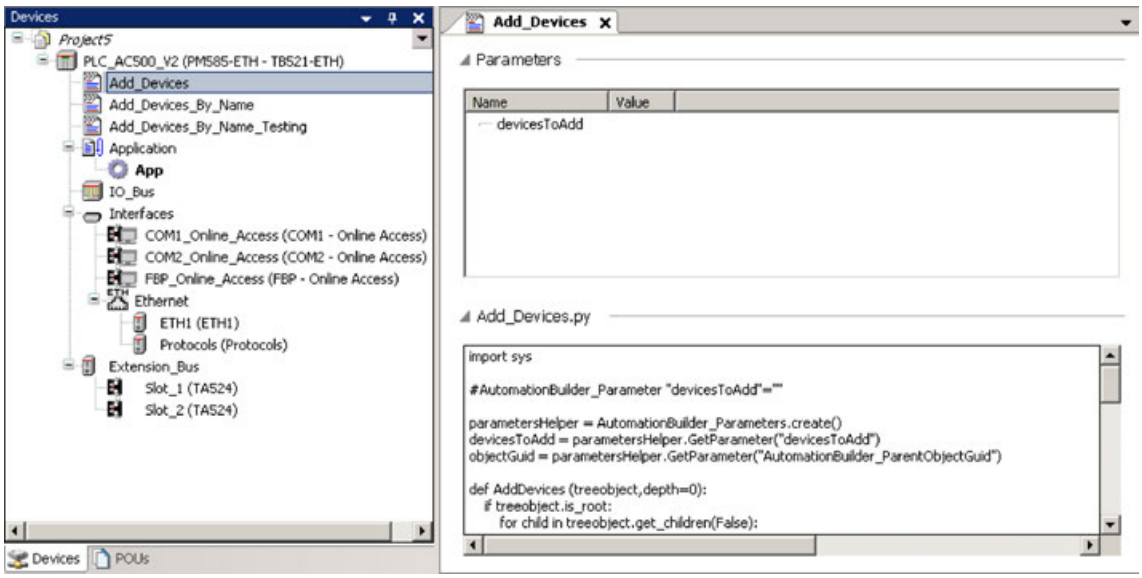
The Automation Builder scripting language is modular and based on IronPython. For this purpose, the Automation Builder “ScriptEngine” component combines the IronPython interpreter with the Automation Builder development environment which makes the extensive python framework libraries available including file access in networks and much more.

1.8.7.2 Working with script objects

Scripts to execute can be added to and stored in the Automation Builder project. Additionally, parameters can be added to scripts, so that generic scripts can be customized before execution.

Adding a script object to the project

1. In the device tree, right-click on a node (e.g. a PLC node) and click *“Add object”*.
2. Under *“Scripting category”* select *“Script ➔ Add object”*.
   ⇨ The 'Add Script' dialog is displayed.
3. Browse and select a script from the file system or create a new script by clicking *[Add]*.
   ⇨ A script is added below the selected node and the editor is opened.



4. The default parameter values are read from the script. The user can edit the default values as required.

> *Editing scripts within Automation Builder is not supported. You can use an external editor to edit the script and then import it to Automation Builder.*

> *The script objects can be reused within the project via copy-and-paste around the device tree.*

Execution

The user can execute the script with the parameter values via the execute button in the editor or via right-click on the script object in the device tree by selecting *"Script ➔ Execute"*.

Import

The user can import the script from the file system. This will replace the contents of the current script object with the contents of the imported file. Optionally, parameter values will be preserved if the imported script has a matching named parameter. In the device tree, right-click on a script object and select *"Script ➔ Import"*.

Export

The user can export the selected script and saved it as a new file in the file system. The exported file does not include any edited parameter values. In the device tree, right-click on a script object and select *"Script ➔ Export"*.

Parameters

The following instructions help the user to create parameters in the python script:

- Parameters must be defined in the script.
- Parameters and values are optional.
- The `ParameterName` and the `ParameterValue` must be delimited with symbols. The format must be as follows:
  "#AutomationBuilder_Parameter {"ParameterName"} {= "ParameterValue"}
- {ParameterName} is the name given to the parameter. This allows the values to be referenced in the python script.
- {ParameterValue} is the default value given to the parameter. This value can be modified in the editor.

The example below shows the format of the `ParameterName` and `ParameterValue` in the script.

- #AutomationBuilder_Parameter "numWidgets": creates a new parameter called `numWidgets`.
- #AutomationBuilder_Parameter "numWidgets" = "4": creates a new parameter called `numWidgets` and initializes to the value 4.

Using parameters within the python script:

- Parameters can be used in the script by creating an instance of the parameter helper: parameterHelper = AutomationBuilder_Parameters.create()
- Individual parameters are retrieved by calling: GetParameter(parameterName). devicename = parameterHelper.GetParameter("Name")

Python script examples

A set of python script examples are available in the path %Public%\Documents\Automation-Builder\Examples\Python scripts.

1.8.7.3 Python script editor

In Automation Builder a browser-based python script editor is integrated. This allows the user to modify the existing python script, to create a python script from the scratch and to finally execute the script. Moreover, it assists the user in writing the script with the following features:

- Auto suggest
  - IntelliSense suggestions for the python syntax during typing.
  - IntelliSense for CODESYS script engine and Automation Builder injected script objects.
  - Built-in language service that provides complete code intelligence for objects, properties and methods.
  - Details of the object with *[CTRL]* + *[spacebar]*.
- Auto completion
  Press the Enter key on a function suggested by IntelliSense in order to insert it.
- Python syntax highlighting (basic syntax colorization)
  The function and its respective namespace is automatically colored in order to match colors.
- Matching brackets
  Matching brackets are highlighted as soon as the cursor is near to one of them using the command palette.
- Zoom
  Changes the font size of the editor's content.
- Find and replace
  Support of 'Find' (search for a keyword) and 'Find and replace' (search and replace a keyword). This feature is supported in the editor, however not integrated in Automation Builder platform.
- Minimap
  High level overview of the script for a quick navigation and code understanding.
- Copy/paste
  Support of 'copy and paste' of the script text within and into the editor.
- Undo/redo
  Support of 'undo/redo' for editing actions. This feature is supported in the editor, however not integrated in Automation Builder platform.
- Keyboard shortcuts
  Keyboard shortcuts allow to perform most tasks directly from the keyboard (e.g. *[CTRL]* + *[Z]*, *[CTRL]* + *[Y]*) including copy and paste. For further keyboard shortcuts refer to the command palette (*[F1]*).
- Folding
  Support of folding and expanding script regions.
- Comment/uncomment the code
  Support of commenting (*[CTRL]* + *[K]*) and uncommenting (*[CTRL]* + *[C]*) code through shortcuts.
- 'Execution' button
  Executes the script directly in the editor window.
- In order to start a new script from the scratch the user can start with an empty editor. This can be done via the 'Add script' dialog without script file selection.

For further features that can be used in the python script editor refer to the command palette (*[F1]*).

Limitations with CODESYS script engine IntelliSense

- No IntelliSense available for return type of a property.
- No support of IntelliSense for keyword “None”.
- No IntelliSense support for method overloading.
- No IntelliSense support for methods that return an object.
- Private methods are also part of IntelliSense. Refer to the CODESYS script engine document to verify the access modifier.

1.9 Human machine interface

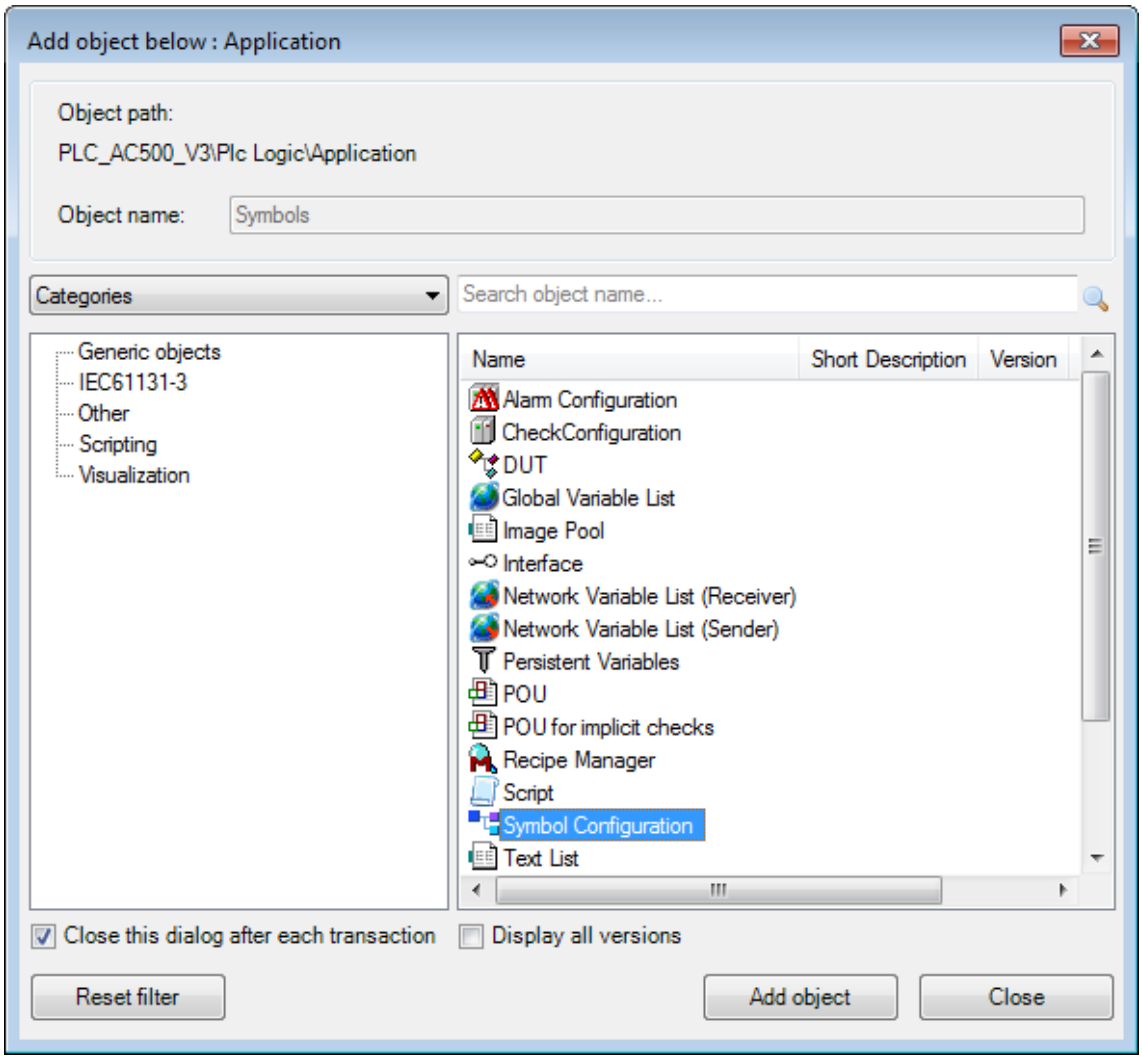
1.9.1 Panel Builder interface

This document describes HMI CP600 Control Panel configuration in Automation Builder and starting HMI configuration and programming software Panel Builder 600 from Automation Builder. The Panel Builder project created for the HMI CP600 is stored within the Automation Builder project.

1.9.1.1 Adding desired AC500 PLC to the project

Configuring the Symbol File

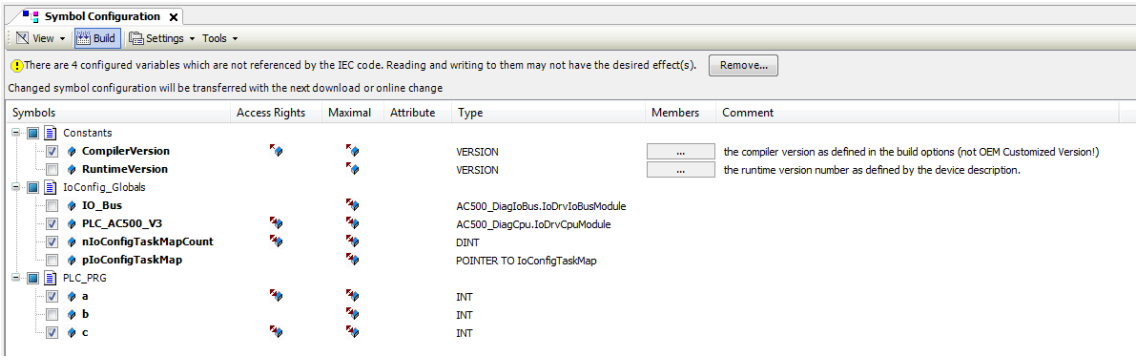
1. In the Automation Builder device tree, right-click the *"Application"* node and click *"Add object"*



2. Click on *"Symbol Configuration"* and click *"Add object"*

   ⇨ A *"Symbol Configuration"* object is added to the *"Application"* node.

3. Double-click on the *"Symbol Configuration"* object , then click on *"Build"*



⇨ A list of all variables in the project is generated. Single variables or groups of variables can be selected by checking the corresponding item in the list.

4. After the symbols have been configured, download the project or click *"Build ➔ Generate code"* in the Automation Builder to create an .xml file containing all the variables read to be imported in the Tag Editor.

1.9.1.2 Creating a Panel Builder project

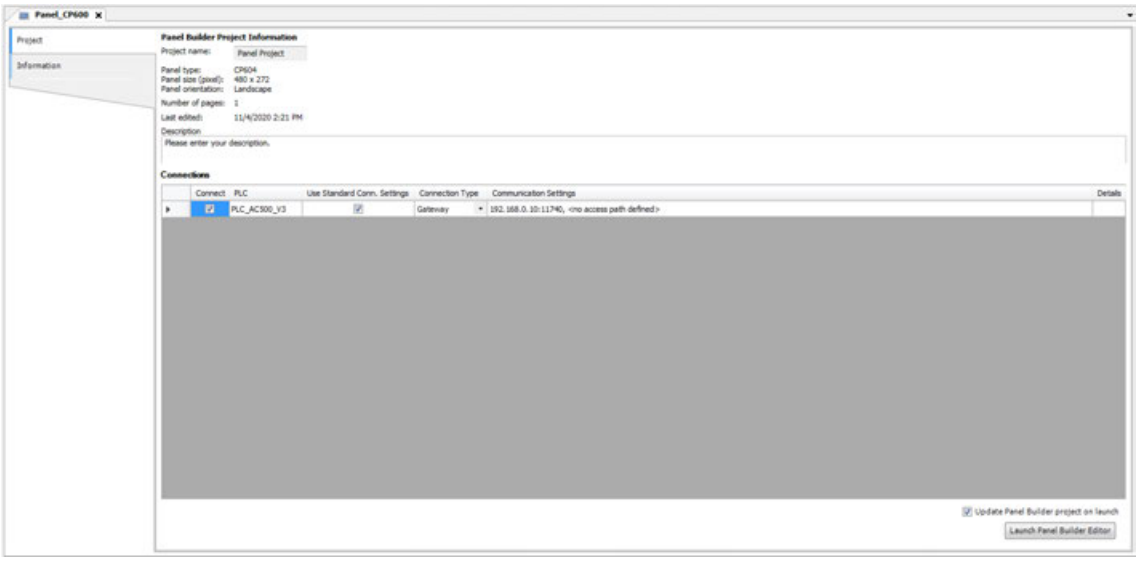
Adding a panel object

1. Right-click in the Automation Builder device tree and click *"Add object ➔ Panel-CP600"*
2. Click on *"CP600 Control Panel"* and click *"Add object"*

⇨ A Control Panel object is added to the Automation Builder device tree.

Starting a Panel Builder project

1. In the device tree, double-click *"Panel CP600"* object to start Panel CP600 screen.



2. Select the required PLC and enable the checkbox in the 'Use Standard Connection Settings' column to use it as a standard gateway connection.

   You can set communication settings using the application program or by creating custom communication settings. Custom communication settings can be configured by clicking the button in the 'Details' column.

3. Enable the *"Update Panel Builder project on launch"* checkbox and click *[Launch Panel Builder Editor]*.

> *If you update Automation Builder project with new variables and data types or if there are changes in existing Automation Builder project variables and data types (new, modified, deleted), recompile CODESYS application to refresh the symbol file, then launch Panel Builder editor.*

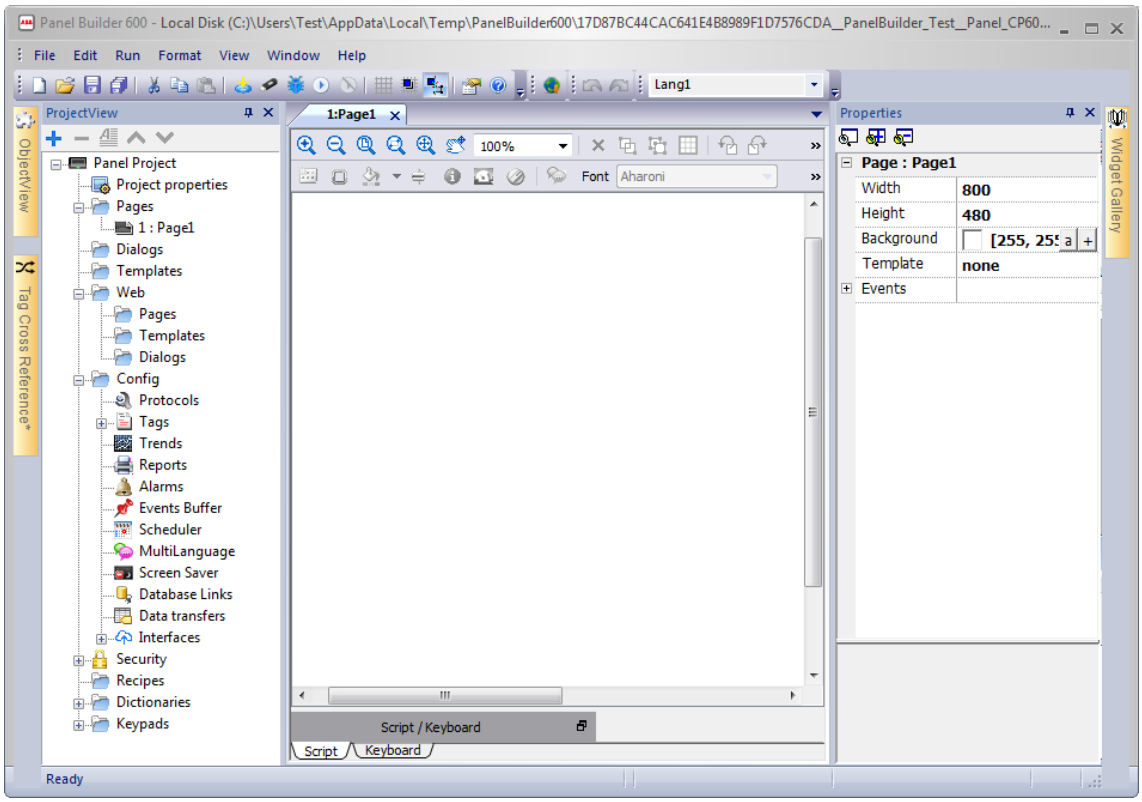
4. Select *"New"* and click *"Open"* to create a new HMI project.

   ⇨ A project wizard is displayed.

> *If you want to import an already existing Panel Builder project file from the file system, select "Import existing project file" and proceed.*

5. Select the required panel type and orientation and click *"Finish"*.

   ⇨ A new project wizard starts only if the Panel project is empty.

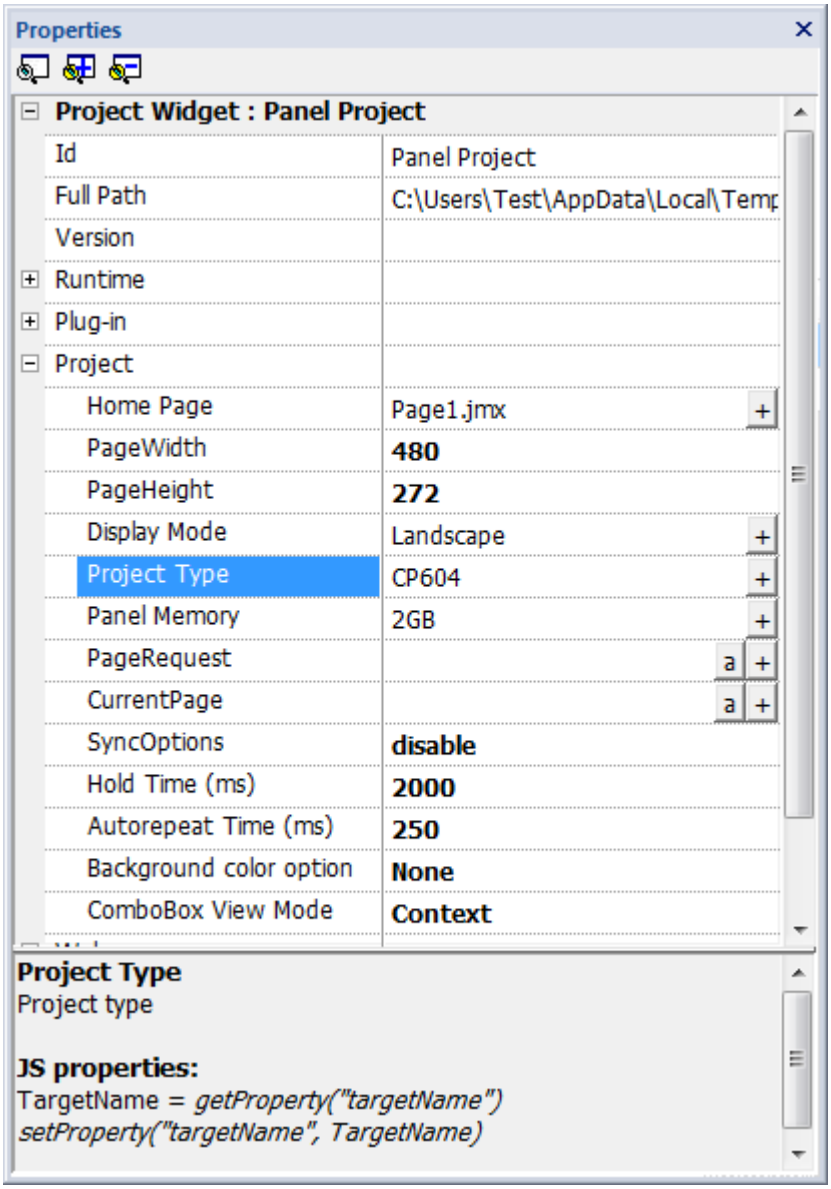


> *The panel projects can be compared in Automation Builder using the "Compare Objects" option.*

Changing panel type

1. In the Panel project, double-click *"Project properties"* to change the panel type to the panel which is used.

   ⇨ The Properties dialog is displayed.



2. In the Properties dialog, expand *"Project"* and click *"Project Type "*.

   ⇨ A project wizard dialog is displayed.

3. Select the desired panel type and click *"Finish"*.

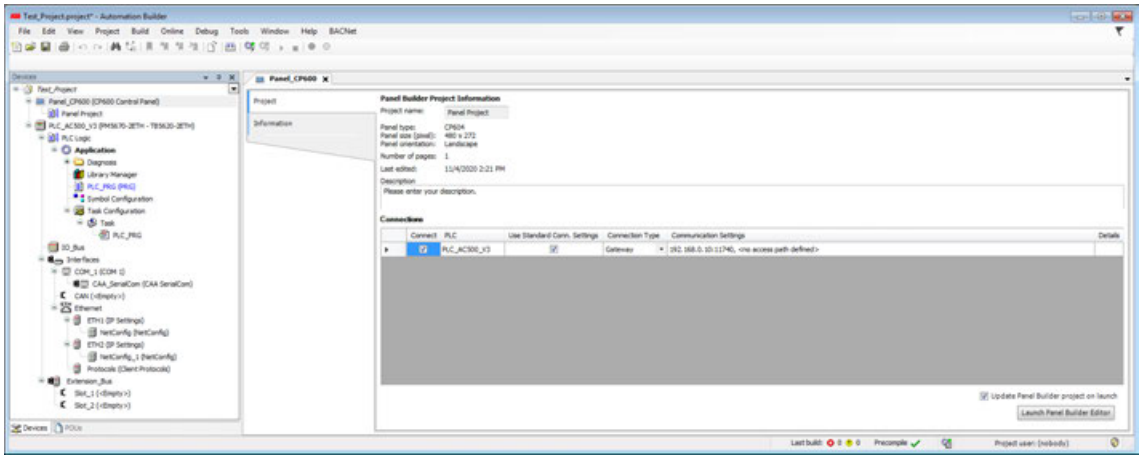
Project information

The project information view provides an overview of the Panel Builder project without opening the project. To open the project information, double-click the *"Panel_CP600"* object.

The project information is updated every time the Panel Builder project is edited. You can rename the Panel Builder project via context menu.

> *The project name is internally used as a base for the Panel Builder project file name. Therefore, the project name has to comply with general file name restrictions.*

The Panel Builder project information shows the list of PLCs added to the project.

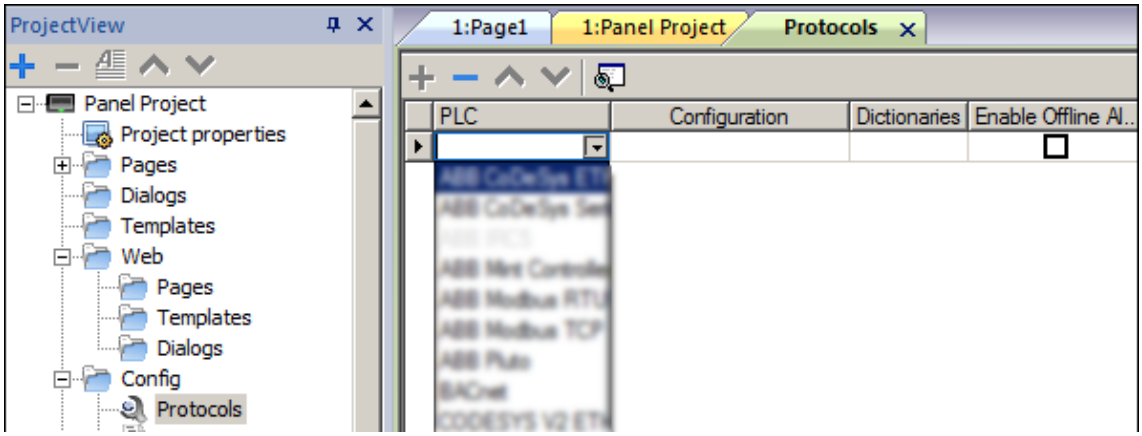


1.9.1.3 Configuring Panel Builder

Configuring communication protocols

> *The user can configure a panel project manually in Panel Builder editor when there is a need to create individual panel projects. Otherwise, the configuration is updated in the panel project while launching Panel Builder editor in Automation Builder.*

1. In the Panel Builder project structure, double-click *"Config ➔ Protocols"*.
2. Click + to add a protocol.



3. Select *"OPC UA Client"* to ensure an encrypted communication between AC500 V3 devices and the control panels. This is necessary to protect passwords and other data in terms of cyber security.

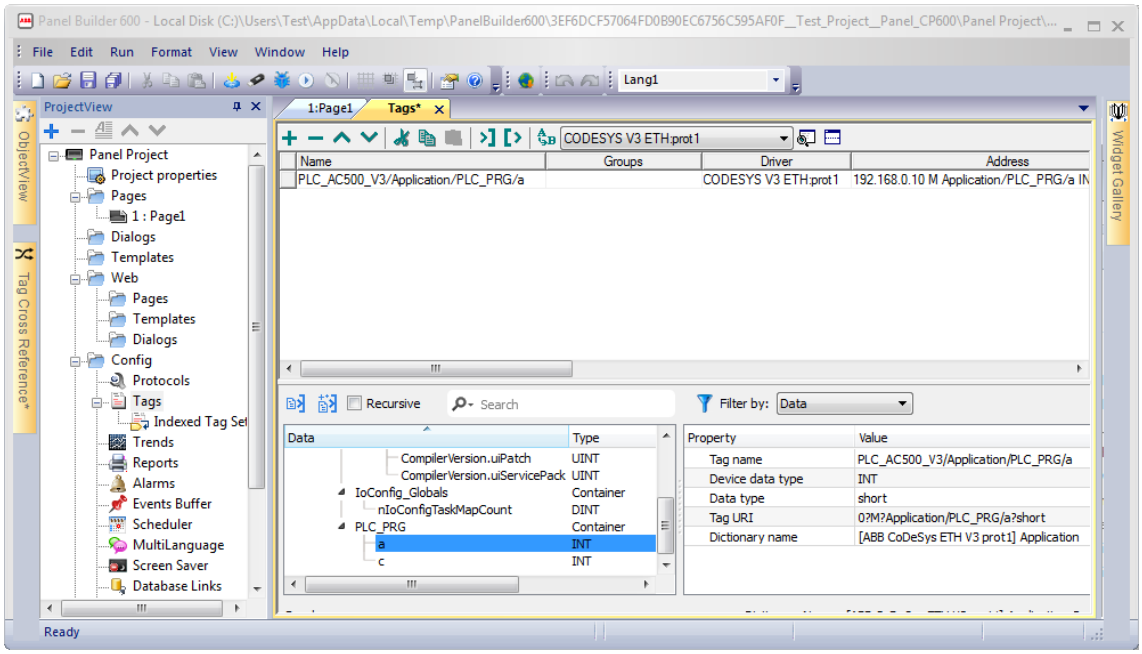
   Set the IP address, port, protocol type and PLC models. Click *[OK]*.

Importing tags

1. In the Panel project view, click *"Config ➔ Tags"*.
2. Select the protocol from the drop-down list and click ›] to import tags.

> *If the Panel Builder contains multiple tag importers, a dialog is displayed to select the required importer type.*

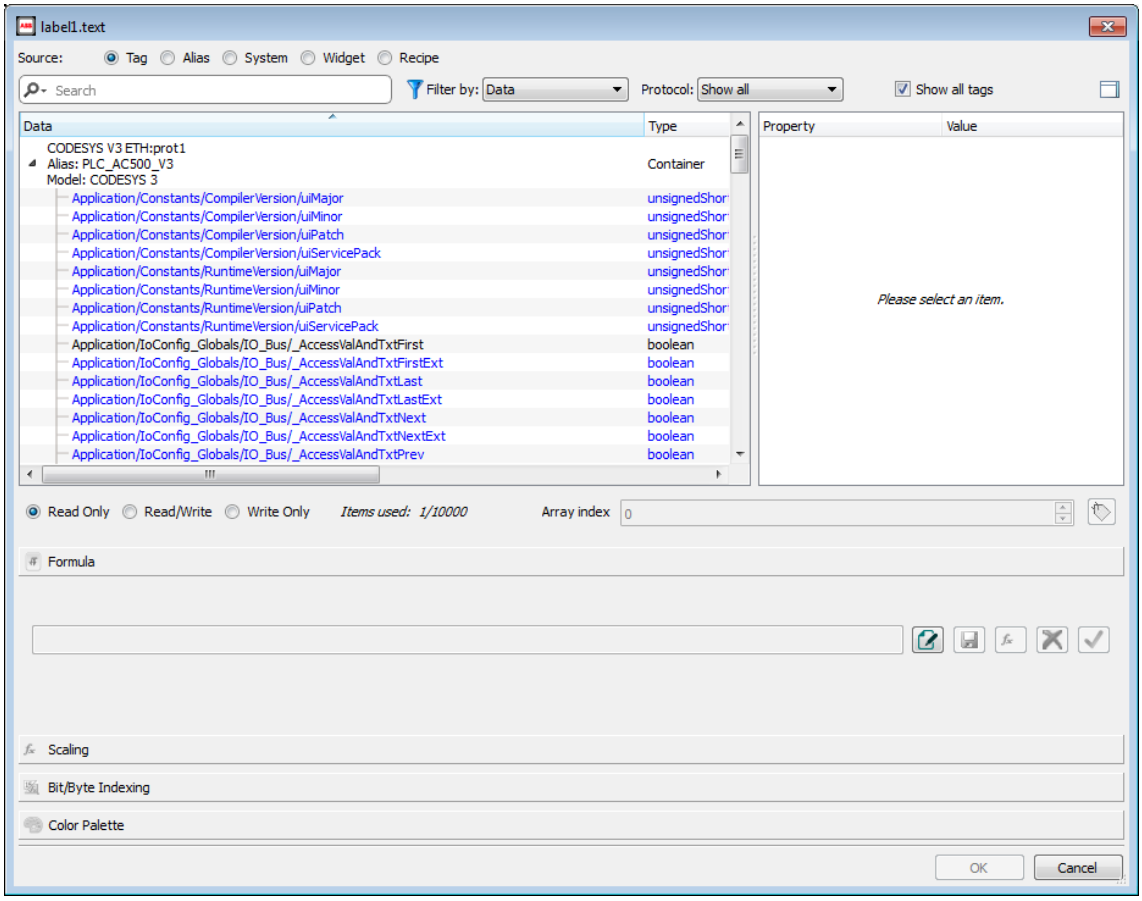
3. Select the symbol file which was exported to the file system.
4. In the lower part of the tag editor, mark the desired tags and click *“Import Tag (s)”* to import the tags to the Panel Builder project.



Attaching tags to widgets

1. In the project view, expand *“Pages”* and double-click **Page1**.
2. In the Panel Builder 600 main menu, select *“View ➔ Toolbars and Docking Windows ➔ Widget Gallery”*.
3. Drag-and-drop the desired widget to the page editor.

4. Right-click on the widget value and select *"Attach To"* to attach a tag to the widget.



5. Select the desired tag and select the desired option for the authorization *"Read Only"* or *"Read/ Write"* or *"Write Only"*. Then, click *[OK]*.

Downloading a project to panel

1. In the Panel Builder main menu, click *"Run ➔ Download To Target"*.
2. Select the CP600 project from the drop-down list and click *"Download"*.

Importing an existing Panel Builder project

1. In the Automation Builder device tree, right-click the Panel project and click *"Import ➔ Panel Builder Project"*.

   System prompts to overwrite the exiting project object data.
2. Click *"Yes"* to confirm.
3. Select the existing Panel Builder 600 project from the file system and click *"Open"*.

   ⇨ The imported project is displayed.

Exporting Panel Builder project

1. In the Automation Builder device tree, right-click the Panel Builder 600 project and click *"Export ➔ Panel Builder Project"*.
2. Click *"Browse"* and select the desired location in the file system and save the project file.
   ⇨ A success message is displayed, if the project file exports successfully.

> *When you double-click the Panel Builder project node, the compressed information of the node is extracted into a temporary folder and then the external Panel Builder program is started. After the external Panel Builder program is closed, the corresponding Panel Builder files can be compressed back into the node and saved in the Automation Builder project.*

> *We recommend to edit the Panel Builder project by starting Panel Builder through the Automation Builder. You can also export a Panel Builder project to the file system to edit the project by using the external Panel Builder. Then, reimport it to Automation Builder.*

1.9.2 SCADA Integration

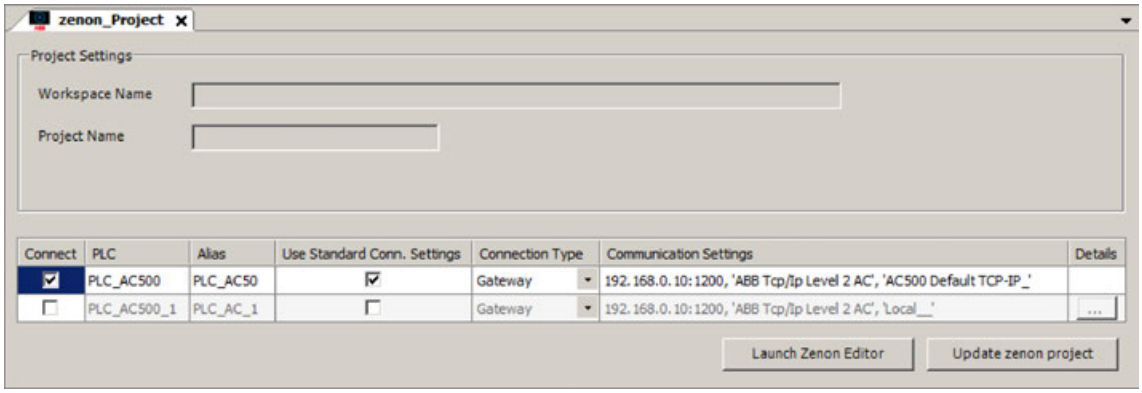
Overview

This document describes SCADA integration configuration in Automation Builder using zenon editor. The configured device network address information and variables are synchronized with zenon editor to avoid double entry.

The Automation Builder supports both standard and multi-user functionality.

1.9.2.1 Creating Workspace and Project

1. In the device tree, double-click *"zenon_Project"*.



⇨ To launch the zenon editor, click *[Launch Zenon Editor]*.

To update the zenon project with latest changes of application program, click *[Update zenon project]*.

2. Select the required PLC and select the *“Use Standard Conn. Settings”* option to use as a standard gateway connection.

   This enables the user to use the same communication settings that Automation Builder uses to communicate to the PLC.

   > *The configured gateway communication settings made in Automation Builder are displayed in the column 'Connection Type'.*

   As an alternative you can create custom communication settings: Deselect the *“Use Standard Conn. Settings”* option and click the button in the 'Details' column.

3. Click *[Launch Zenon Editor]* to create a new workspace and project.

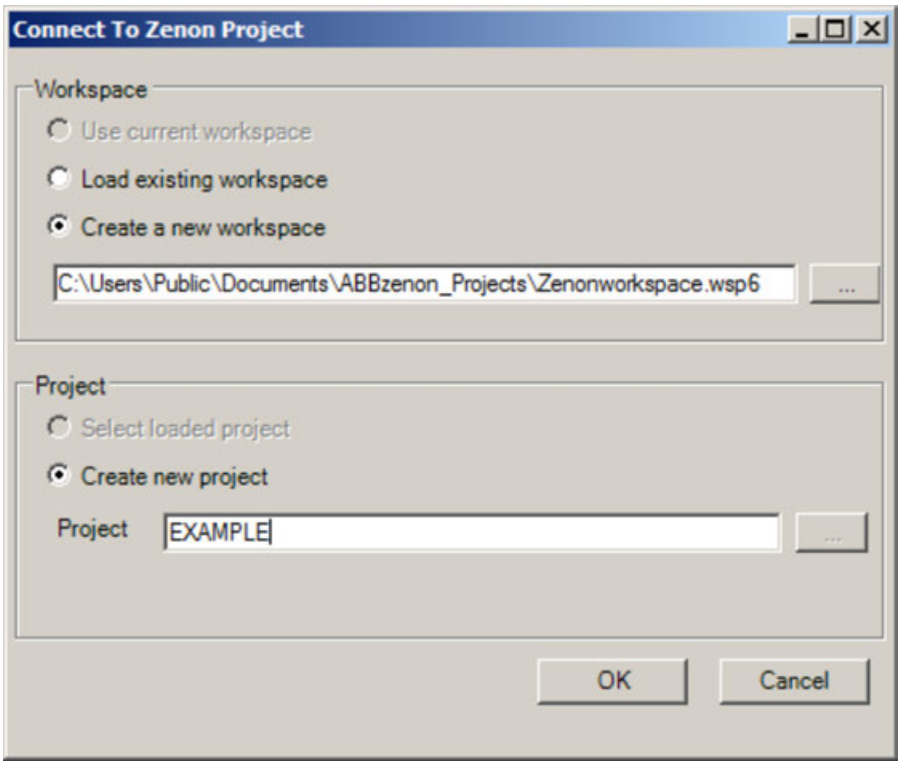


Fig. 345: Connect to zenon project

⇨

> *If Zenon Editor is already running, then select the “Use current workspace” option.*

4. Select the *“Create a new workspace”* option and select the file location to create a new workspace.
5. Select the *“Create new project”* option to create a project.

   ⇨ ABB zenon editor is displayed.

> *If you update or change an Automation Builder project with new variables or data types (new, modified, deleted), recompile the application to refresh the symbol file and click [Update zenon project].*

After creating the project and workspace in Automation Builder, it is not required to set it again for the zenon object. A double-clicking on the zenon project shows the previously configured zenon project and the workspace.

1.9.2.2 Loading existing Workspace and Project

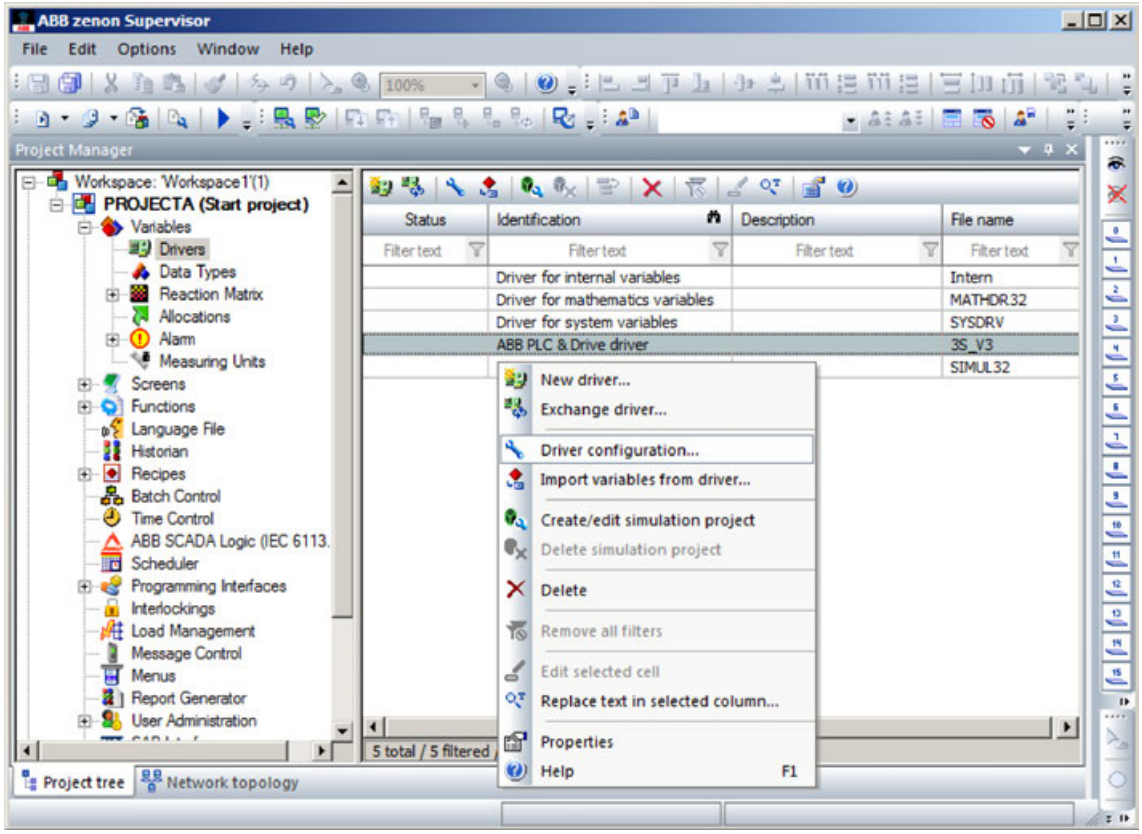
You can load an existing workspace and project to ABB zenon supervisor.

1. In the zenon_Project screen, click *[Update zenon project]*.
   ⇨ Connection to the zenon project dialog is displayed.
2. In the workspace area, enable *"Load existing workspace"* and select the location.
3. In the project area, enable *"Select loaded project"* and click *[OK]*.
   ⇨ Zenon editor loads the selected existing workspace and the project.

1.9.2.3 Checking the Gateway Settings in a Zenon Project

The gateway settings configured in Automation Builder can be checked in a zenon project. The IP address configured in Automation Builder are displayed in the zenon driver configuration.

In the Project Manager structure of the zenon editor, click *"Variables ➔ Drivers"* to configure the driver configuration.



The *"Settings"* tab shows all gateway settings based on the number of configured PLCs in Automation Builder. The IP address should be similar to the project gateway settings in Automation Builder.

> *In the zenon project window, the **Connect** column should be checked to transfer the desired number of PLC connection settings to the zenon editor.*

1.9.2.4 Generating a Symbol File

Before generating the symbol file, define the variables in theCODESYS application.

1. In the CODESYS application main menu, click *“Project ➔ Options”*.
2. In the *“Options”* dialog, click *“Symbol configuration”*. Enable *“Dump symbol entries”* and *“Dump XML symbol table”* and click *“Configure symbol file”*.

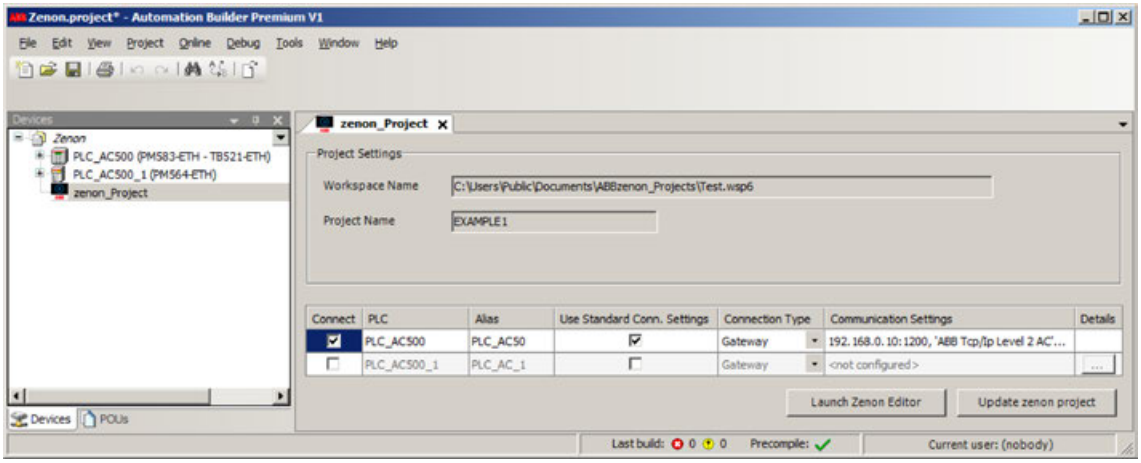
   ⇨ Set object attributes dialog is displayed.
3. Enable *“Export variables of object”*. If this option has a gray background, double-click on it to activate.
4. In the CODESYS application window, click at the bottom of the window and click *“Resources ➔ Tools ➔ Target settings”*.
5. In the target settings dialog, open the *“General”* tab and enable *“Download symbol file”*.
6. From the CODESYS application main menu, select *“Project ➔ Build”* to compile the project.

> *Precondition to generate a symbol file is to create the application and perform a PLC program build in CODESYS application.*

The symbol file is generated after the build. The data exchange can be transferred to the zenon project by clicking *[Update zenon project]* in Automation Builder.

1.9.2.5 Updating Standard Data Types

The standard data types created in CODESYS application can be updated to the zenon project by clicking on *“Update zenon project”*.



> *Data types and variables can be updated from the desired number of PLCs configured in the zenon project of Automation Builder.*

In the zenon project, double-click *“Variables”* and check the updated standard data type.

1.9.2.6 Creating Data Types

1. In the CODESYS application open the *“Data types”* tab. Right-click *“Data types* ➔ *Add object”* to create a new data type.
2. Enter the user defined data type name.
3. In *“POUs”* tab, add the user defined variable data type and compile.
   ⇨ The user defined data type is created and can be imported in the zenon editor.

> *If you modify or delete the data types in CODESYS application, compile with “Rebuild all option”.*

1.9.2.7 Importing Data Types in zenon Editor

1. In the zenon project, click *[Update zenon project]* to update the data types.
2. Click *“Update”* to update the variables and data types to the zenon project.
   ⇨ The user defined variables and data types are imported to the zenon project.

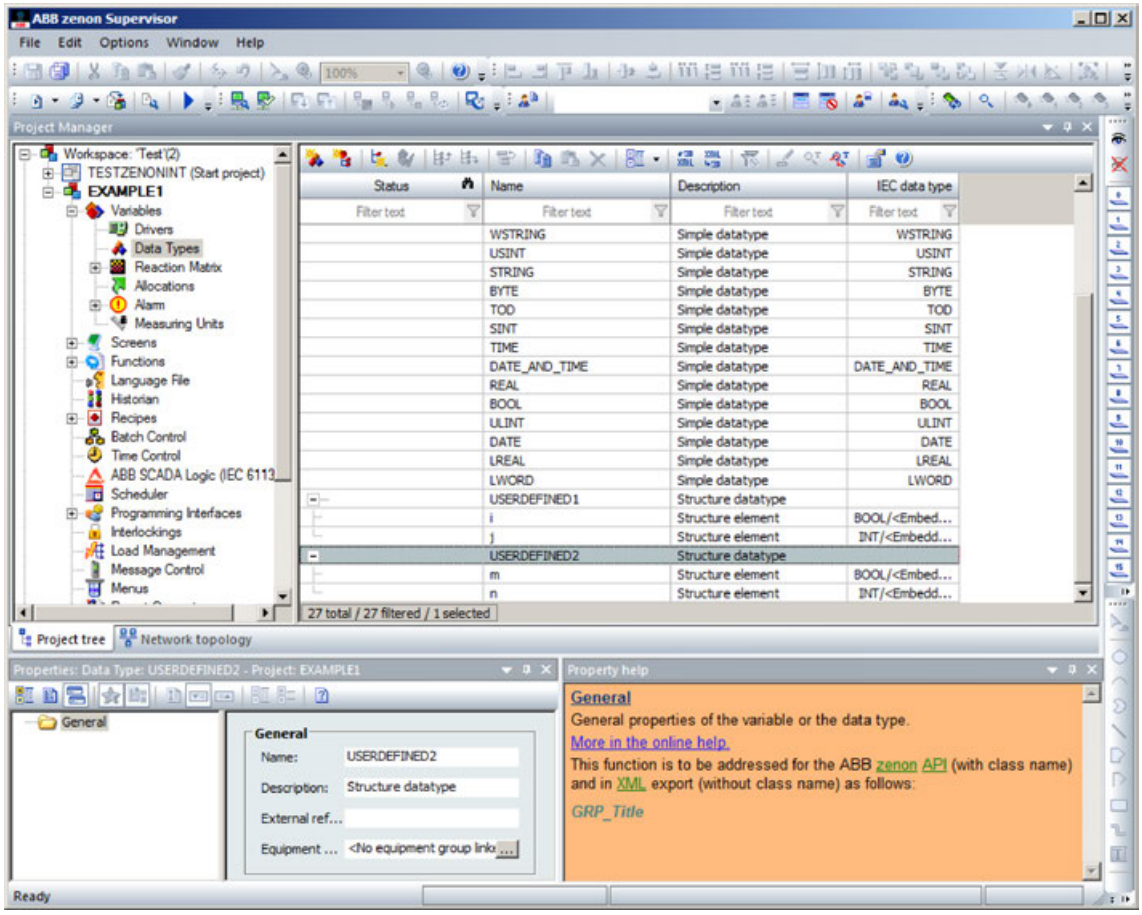


Fig. 346: User defined variables

1.10 Reference, function blocks

Reference documentation

- *_BlockGetData*
- *_BlockGetPool*
- *_CloneMessage*
- *_CreateArrayReceiver*
- *_CreateIdAreaReceiver*
- *_CreateMaskReceiver*
- *_CreateMessage*
- *_CreateSingleIdReceiver*
- *_DeleteReceiver*
- *_DisableSyncService*
- *_DriverClose*
- *_DriverGetSize*
- *_DriverOpenH*
- *_DriverOpenP*
- *_EnableSyncService*
- *_FlatCreateH*
- *_FlatCreateP*
- *_FlatDelete*
- *_FlatDisable*
- *_FlatEnable*
- *_FlatGetSize*
- *_FlatRead*
- *_FlatTest*
- *_FlatUpdate*
- *_FreeMessage*
- *_GetBaudrate*
- *_GetBusAlarm*
- *_GetBusload*
- *_GetBusState*
- *_GetCiAState*
- *_GetDiagnosis*
- *_GetLostCounter*
- *_GetMessageDataPointer*
- *_GetMessageId*
- *_GetMessageLength*
- *_GetMsgCount*
- *_GetNetId*
- *_GetReceiveCounter*
- *_GetReceiveErrorCounter*
- *_GetReceivePoolSize*
- *_GetReceiveQueueLength*
- *_GetTimeStamp*
- *_GetTransmitCounter*
- *_GetTransmitErrorCounter*
- *_GetTransmitPoolSize*
- *_GetTransmitQueueLength*
- *_Is29BitIdMessage*
- *_IsRTRMessage*
- *_IsSendingActive*
- *_IsTransmitMessage*
- *_JobAbort*
- *_JobClose*
- *_JobExecute*
- *_JobGetId*
- *_JobGetParams*

- *_JobGetState*
- *_JobOpen*
- *_JobOpenEx*
- *_JobReset*
- *_JobSetState*
- *_LostMessages*
- *_MsgAddRef*
- *_MsgClone*
- *_MsgGetData*
- *_MsgReceive*
- *_MsgRelease*
- *_MsgReleaseEx*
- *_MsgSend*
- *_PoolCreateH*
- *_PoolCreateP*
- *_PoolDelete*
- *_PoolExtendH*
- *_PoolGetBlock*
- *_PoolGetBlockSize*
- *_PoolGetCurCapacity*
- *_PoolGetNumBlocksLeft*
- *_PoolGetSize*
- *_PoolPutBlock*
- *_Read*
- *_ReadArrayReceiver*
- *_RegisterIdArea*
- *_ResetBusAlarm*
- *_RLstAddPrio*
- *_RLstCheckPrio*
- *_RLstCreateH*
- *_RLstCreateP*
- *_RLstDelete*
- *_RLstGetHighestPrio*
- *_RLstGetSize*
- *_RLstRemovePrio*
- *_SDOServerClose*
- *_SDOServerDoCycle*
- *_SDOServerOpen*
- *_SetCiAState*
- *_StorageGetIndexId*
- *_StorageGetTableId*
- *_UnregisterIdArea*
- *_WorkerRegister*
- *_WorkerUnregister*
- *_Write*
- *_XChgCreateH*
- *_XChgCreateP*
- *_XChgDelete*
- *_XChgExtendH*
- *_XChgGetSize*
- *_XChgIsEmpty*
- *_XChgMsgLeft*
- *AbbETrig*
- *AbbLCon*
- *AbbLConA*

- *AbbLConC*
- *AbbLConCA*
- *ABORT_CODE*
- *AbstrTreeNode*
- *AC500DeviceDiag*
- *ACAlarmExtender*
- *ACCESS_MODE*
- *AccessRights*
- *ACCESSTYPES*
- *AcknowledgeRequestBuilder*
- *ActionController*
- *active_low*
- *AdapterDiagnosis*
- *ADAPTERSTATE*
- *AdapterState*
- *AddBrowseInfo*
- *AddLogger*
- *AddMultiplicatedVector*
- *AddPoints*
- *ADDR*
- *ADDR_TO_ID*
- *ADDR_TYPE*
- *AddressArea*
- *AddressLeafTreeNode*
- *AdjustData_LocalToOpc*
- *AdjustData_OpcToLocal*
- *AffectedSourcesHelp*
- *AINFO_TYPE*
- *ALARM_ID*
- *AlarmFctWriteLatchVariable*
- *ALARMGROUP_ID*
- *AlarmIndices*
- *AlarmInfo*
- *AlarmingCall*
- *AlarmLatchAdapter*
- *AlarmSelectionInfo*
- *AlarmSelectionInfoDefault*
- *AlarmState*
- *AlarmStateTransition*
- *AlarmStorageConvertFromTimestamp*
- *AlarmStorageConvertToTimestamp*
- *AlarmStorageConvertValueToLREAL*
- *AlarmStorageConvertValueToREAL*
- *AlarmStorageConvertValueToString*
- *AlarmStorageGetDefaultText*
- *AlarmStorageGetMessageCount*
- *AlarmStorageGetTextId*
- *AlarmStorageGetTextListName*
- *AlarmStorageLatchVariable*
- *AlarmStorageReader*
- *AlarmStorageRow*
- *AlarmStorageStaticData*
- *AlarmType*
- *AllocAndCopyPString*
- *AllScalarsUnion*

- *AnalogDeviceDescType*
- *AnalyzeExpression*
- *AnalyzeExpressionCombined*
- *AnalyzeExpressionTable*
- *APP_MEMORY_SEGMENT*
- *APP_NAME*
- *AppCallGetProperty*
- *AppCallGetProperty2*
- *AppCallGetProperty2Release*
- *AppCallGetProperty3*
- *AppCallSetProperty*
- *AppCallSetProperty2*
- *AppendToString*
- *AppFindApplicationByName*
- *AppGenerateException*
- *AppGetApplicationByAreaAddress*
- *AppGetApplicationFlags*
- *AppGetApplicationInfo*
- *AppGetAreaAddress*
- *AppGetAreaOffsetByAddress*
- *AppGetAreaPointer*
- *AppGetAreaSize*
- *AppGetCurrent*
- *AppGetFirstApp*
- *AppGetNextApp*
- *AppGetProjectInformation*
- *AppGetSegment*
- *AppGetSegmentAddress*
- *AppGetSegmentSize*
- *APPLICATION*
- *APPLICATION_INFO*
- *ApplicationSoftwareVersion*
- *Apply_Attributes*
- *AppNumOfActiveSessions*
- *AppRegisterPropAccessFunctions*
- *AppReset*
- *AppRestoreRetainsFromFile*
- *AppStartApplication*
- *AppStopApplication*
- *AppStoreRetainsInFile*
- *AR_Info*
- *AREA_TYPE*
- *AreaRegister*
- *ARInfo*
- *ARP_Packet*
- *ARRAY_RECV_ENTRY*
- *ArrayDimension*
- *AsciiUpper*
- *AskCredentialsHelper*
- *ASM_IWORKER*
- *ASM_STATE*
- *Assert*
- *AssignerBase*
- *AsyncAdd*
- *AsyncBase*

- *AsyncBaseClass*
- *AsyncGetJobReturnValue*
- *ASYNCJOB_EVENTPARAM*
- *ASYNCJOB_HOOKPARAM*
- *ASYNCJOB_PARAM*
- *ASYNCJOB_TASKPARAM*
- *AsyncKill*
- *AsyncProperty*
- *AsyncRemove*
- *AsyncRemoveAll*
- *AsyncWrapper*
- *atan2*
- *AtomicReadLInt*
- *AtomicReadLReal*
- *AtomicReadLTime*
- *AtomicReadLWord*
- *AtomicReadULInt*
- *AtomicWriteLInt*
- *AtomicWriteLReal*
- *AtomicWriteLTime*
- *AtomicWriteLWord*
- *AtomicWriteULInt*
- *ATTRIB*
- *AutomaticTimeSync*
- *BackgroundTask*
- *BackgroundTaskFactoryArgs*
- *BackgroundTaskFactoryBase*
- *BackupRestore*
- *BACNET_READ_FILE_RESULT_RECORD*
- *BACNET_READ_FILE_RESULT_STREAM*
- *BACnetAccessCredentialDisableReasonString*
- *BACnetAccessCredentialDisableString*
- *BACnetAccessDoorPVString*
- *BACnetAccessEventString*
- *BACnetAccessPassbackModeString*
- *BACnetAccessUserTypeString*
- *BACnetAccessZoneOccupancyStateString*
- *BACnetAccumulator*
- *BACnetAccumulatorStatusString*
- *BACnetAcknowledgeAlarm*
- *BACnetAcknowledgeInternalAlarm*
- *BACnetActionString*
- *BACnetAddListElement*
- *BACnetAlarmSummResponseCbCompletion*
- *BACnetAnalogInput*
- *BACnetAnalogOutput*
- *BACnetAnalogValue*
- *BACnetAsyncTransactionToken*
- *BACnetAttachUserDataToObjectHandle*
- *BACnetAttachUserDataToObjectPropertyOverObjectHandle*
- *BACnetAuthenticationStatusString*
- *BACnetAuthorizationModeString*
- *BACnetAveraging*
- *BACnetBackupBACnetDevice*
- *BACnetBackupStateString*

- *BACnetBinaryInput*
- *BACnetBinaryOutput*
- *BACnetBinaryPVString*
- *BACnetBinaryValue*
- *BACnetBitStringGetBit*
- *BACnetBitStringSetBit*
- *BACnetCalendar*
- *BACnetCancelPendingConfirmedRequest*
- *BACnetChangeListErrorCbCompletion*
- *BACnetClientAcknowledgeAlarm*
- *BACnetClientAddListElement*
- *BACnetClientBackupBACnetDevice*
- *BACnetClientBase*
- *BACnetClientConfPrivateTransfer*
- *BACnetClientConfTextMessage*
- *BACnetClientCreateObject*
- *BACnetClientDeleteObject*
- *BACnetClientDeviceCommControl*
- *BACnetClientGetAlarmSummary*
- *BACNetClientGetEnrollmentSummary*
- *BACnetClientGetEventInfo*
- *BACnetClientLifeSafetyOperation*
- *BACnetClientReadAllPropertyDataContents*
- *BACnetClientReadProperty*
- *BACnetClientReadPropertyMultiple*
- *BACnetClientReadRange*
- *BACnetClientReadStreamFile*
- *BACnetClientReinitializeDevice*
- *BACnetClientRemoveListElement*
- *BACnetClientRestoreBACnetDevice*
- *BACnetClientSubscribeCOV*
- *BACnetClientSubscribeCOVProperty*
- *BACnetClientTimeSynchronization*
- *BACnetClientUTCTimeSynchronization*
- *BACnetClientWriteProperty*
- *BACnetClientWritePropertyMultiple*
- *BACnetClientWriteStreamFile*
- *BACnetClose*
- *BACnetCloseClientCustomer*
- *BACnetCommand*
- *BACnetConfCOVNotification*
- *BACnetConfEventNotification*
- *BACnetConfPrivateTransfer*
- *BACnetConfTextMessage*
- *BACnetConstructObject*
- *BACnetControlStackInternalObjectActions*
- *BACnetCopyPropertyContents*
- *BACnetCreateObject*
- *BACnetCreateObjectErrorCbCompletion*
- *BACnetCreateObjectResponseCbCompletion*
- *BACnetCreateObjectResult*
- *BACnetDataTypeString*
- *BACnetDateRange*
- *BACnetDateTime*
- *BACnetDateTimeCmp*

- *BACnetDateTimeToString*
- *BACnetDayOfWeekBitsString*
- *BACnetDayOfWeekString*
- *BACnetDeleteDeviceAddressBindings*
- *BACnetDeleteObject*
- *BACnetDeleteObjectIdNameBindings*
- *BACnetDestroyObject*
- *BACnetDevice*
- *BACnetDeviceAddressToInstNumber*
- *BACnetDeviceCommControl*
- *BACnetDevObjPropReference*
- *BACnetDevStatusString*
- *BACnetDoesObjectExist*
- *BACnetDoesObjectNameExist*
- *BACnetDoorAlarmStateString*
- *BACnetDoorSecuredStatusString*
- *BACnetDoorStatusString*
- *BACnetDoorValueString*
- *BACnetDumpStackInformation*
- *BACnetEnableStackLogging*
- *BACnetEnrollmentSummResponseCbCompletion*
- *BACnetEnumString*
- *BACnetEventEnrollment*
- *BACnetEventInfoResponseCbCompletion*
- *BACnetEventLog*
- *BACnetEventStateString*
- *BACnetEventTransitionString*
- *BACnetEventTypeString*
- *BACnetFile*
- *BACnetFileAccessString*
- *BACnetFindUpdateDeviceAddressBindings*
- *BACnetFindUpdateObjectIdNameBindings*
- *BACnetFreeStackAllocatedMemory*
- *BACnetGetAlarmSummary*
- *BACnetGetBACstackTaskPriority*
- *BACnetGetCheckInvalidDateResponses*
- *BACnetGetCheckInvalidDateWrites*
- *BACnetGetCheckInvalidEnumResponses*
- *BACnetGetCheckInvalidEnumWrites*
- *BACnetGetCheckInvalidUnsignedResponses*
- *BACnetGetCheckInvalidUnsignedWrites*
- *BACnetGetClientDeviceCommunication*
- *BACnetGetDatabaseObjectDescription*
- *BACnetGetDatabaseObjectPropertyDescription*
- *BACnetGetDccValue*
- *BACnetGetDeviceAddressBindingList*
- *BACnetGetDeviceAddressBindingsCacheOptions*
- *BACnetGetEnrollmentSummary*
- *BACnetGetEventInfo*
- *BACnetGetObjectHandle*
- *BACnetGetObjectIdentifierFromHandle*
- *BACnetGetObjectIdNameBindingList*
- *BACnetGetObjectIdNameBindingsCacheOptions*
- *BACnetGetPropertyAccessRight*
- *BACnetGetPropertyCallbackAttachment*

- *BACnetGetPropertyCallbackAttachmentByHandle*
- *BACnetGetStackApiVersion*
- *BACnetGetStackApiVersionParts*
- *BACnetGetUserDataFromObjectHandle*
- *BACnetGetUserDataFromObjectPropertyOverObjectHandle*
- *BACnetGlobalGroup*
- *BACnetGroup*
- *BACnetIam*
- *BACnetIAmEx*
- *BACnetIHave*
- *BACnetIHaveEx*
- *BACnetInitMidnightTimer*
- *BACnetInstnumberToDeviceAddress*
- *BACnetIntegerValue*
- *BACnetIPdatalink*
- *BACnetIPdatalinkExt*
- *BACnetIsPropertyWriteable*
- *BACnetLargeAnalogValue*
- *BACnetLifeSafetyModeString*
- *BACnetLifeSafetyOperation*
- *BACnetLifeSafetyOpString*
- *BACnetLifeSafetyPoint*
- *BACnetLifeSafetyStateString*
- *BACnetLifeSafetyZone*
- *BACnetLimitEnableString*
- *BACnetLockStatusString*
- *BACnetLoggingTypeString*
- *BACnetLoop*
- *BACnetMaintenanceString*
- *BACnetMonthString*
- *BACnetMSTPdatalink*
- *BACnetMultistateInput*
- *BACnetMultistateOutput*
- *BACnetMultistateValue*
- *BACnetNodeTypeString*
- *BACnetNotificationClass*
- *BACnetNotifyTypeString*
- *BACnetObjectBase*
- *BACnetObjectIdToText*
- *BACnetObjTypeString*
- *BACnetOpenClientCustomer*
- *BACnetPDUtypeToText*
- *BACnetPersistenceInfo*
- *BACnetPolarityString*
- *BACnetPositiveIntegerValue*
- *BACnetPrivateTransferErrorCbCompletion*
- *BACnetPrivateTransferResponseCbCompletion*
- *BACnetProgram*
- *BACnetProgramErrorString*
- *BACnetProgramRequestString*
- *BACnetProgramStateString*
- *BACnetPropertyAttributeExistent*
- *BACnetPropertyAttributePersistent*
- *BACnetPropertyAttributes*
- *BACnetPropertyAttributeWritable*

- *BACnetPropertyIdToText*
- *BACnetPropIDString*
- *BACnetPulseConverter*
- *BACnetReadAllPropertyDataContents*
- *BACnetReadFile*
- *BACnetReadFileResponseCbCompletion*
- *BACnetReadProperty*
- *BACnetReadPropertyMultiple*
- *BACnetReadPropMultipleResponseCbCompletion*
- *BACnetReadPropResponseCbCompletion*
- *BACnetReadRange*
- *BACnetReadRangeResponseCbCompletion*
- *BACnetRegisterAddressBindingsChangeCallback*
- *BACnetRegisterClientCommunicationStateCallback*
- *BACnetRegisterClientDataPoint*
- *BACnetRegisterClientEventNotification*
- *BACnetRegisterClientUnsubscribeCompletionCallback*
- *BACnetRegisterInternalActionErrorCallback*
- *BACnetRegisterObjectIdNameBindingsChangeCallback*
- *BACnetRegisterTimeProviderFunction*
- *BACnetReinitializeDevice*
- *BACnetReliabilityString*
- *BACnetRemoveListElement*
- *BACnetRestartAllClients*
- *BACnetRestoreBACnetDevice*
- *BACnetRetrievePropertyInstance*
- *BACnetRetrievePropertyInstanceByHandle*
- *BACnetSchedule*
- *BACnetSecurityLevelString*
- *BACnetSegmentString*
- *BACnetSendNetworkManagementMessage*
- *BACnetServer*
- *BACnetServerConfCOVNotification*
- *BACnetServerConfEventNotification*
- *BACnetServerInit*
- *BACnetServerPluginBase*
- *BACnetServerPluginCallbackBase*
- *BACnetServerPluginHookBase*
- *BACnetServiceChoiceToText*
- *BACnetServiceString*
- *BACnetSetBACstackTaskPriority*
- *BACnetSetCallback*
- *BACnetSetCheckInvalidDateResponses*
- *BACnetSetCheckInvalidDateWrites*
- *BACnetSetCheckInvalidEnumResponses*
- *BACnetSetCheckInvalidEnumWrites*
- *BACnetSetCheckInvalidUnsignedResponses*
- *BACnetSetCheckInvalidUnsignedWrites*
- *BACnetSetClientDeviceCommunication*
- *BACnetSetClientDeviceFixAddress*
- *BACnetSetClientGlobalCommTimingParameters*
- *BACnetSetClientGlobalMaxDeviceActions*
- *BACnetSetComponentLoggingLevel*
- *BACnetSetDccValue*
- *BACnetSetDeviceAddressBindingsCacheOptions*

- *BACnetSetHook*
- *BACnetSetObjectIdNameBindingsCacheOptions*
- *BACnetSetpointReference*
- *BACnetSetPropertyAccessRight*
- *BACnetSetPropertyCallbackAttachment*
- *BACnetSetPropertyCallbackAttachmentByHandle*
- *BACnetShedStateString*
- *BACnetSilencedStateString*
- *BACnetSrvcAbortCbCompletion*
- *BACnetSrvcErrorCbCompletion*
- *BACnetSrvcIgnoreCbCompletion*
- *BACnetSrvcRejectCbCompletion*
- *BACnetSrvcResponseCbCompletion*
- *BACnetStackControl*
- *BACnetStatusFlagString*
- *BACnetStorePropertyInstance*
- *BACnetStorePropertyInstanceByHandle*
- *BACnetStructuredView*
- *BACnetSubscribeCOV*
- *BACnetSubscribeCOVProperty*
- *BACnetTimeProviderTimeChangedTrigger*
- *BACnetTimeStamp*
- *BACnetTimeStampUnion*
- *BACnetTimeSynchronization*
- *BACnetTrendLog*
- *BACnetTrendLogMultiple*
- *BACnetUnconfCOVNotification*
- *BACnetUnconfEventNotification*
- *BACnetUnconfPrivateTransfer*
- *BACnetUnconfTextMessage*
- *BACnetUnitsString*
- *BACnetUnregisterClientDataPoint*
- *BACnetUnregisterClientEventNotification*
- *BACnetUpdateAccumulatorDataSourceValue*
- *BACnetUTCTimeSynchronization*
- *BACnetVtClassesSupportedString*
- *BACnetWhoHas*
- *BACnetWhoIs*
- *BACnetWriteFile*
- *BACnetWriteFileResponseCbCompletion*
- *BACnetWriteGroup*
- *BACnetWriteProperty*
- *BACnetWritePropertyInstance*
- *BACnetWritePropertyInstanceByHandle*
- *BACnetWritePropertyMultiple*
- *BACnetWritePropMultipleErrorCbCompletion*
- *BASE64*
- *BaseMap*
- *BaseVector*
- *BBMD_Info*
- *BCD_TO_BYTE*
- *BCD_TO_DWORD*
- *BCD_TO_INT*
- *BCD_TO_WORD*
- *BehaviourModel*

- *BehaviourModelBase*
- *BIT_AS_BYTE*
- *BIT_AS_DWORD*
- *BIT_AS_WORD*
- *BitCpy*
- *BitmapEntry*
- *BitmapProcessing*
- *BLINK*
- *BlkClass*
- *BLOB*
- *BlobAlloc*
- *BlobCopyToData*
- *BlobFree*
- *BlockGetData*
- *BmpPoolBegin*
- *BmpPoolEnd*
- *BmpPoolRegister*
- *BmpPoolUnRegister*
- *BOLT*
- *BoolElement*
- *BoolElementFactory*
- *BranchNamedTreeNode*
- *BranchTreeNode*
- *BranchTreeNodeOpcUA*
- *BrushStyle*
- *BTagAlignment*
- *BTagElementType*
- *BTagReaderCreate*
- *BTagReaderCreateDynamic*
- *BTagReaderDestroy*
- *BTagReaderGetComplexContent*
- *BTagReaderGetContent*
- *BTagReaderGetString*
- *BTagReaderGetTagId*
- *BTagReaderGetTagLen*
- *BTagReaderInit*
- *BTagReaderIsDataTag*
- *BTagReaderMoveNext*
- *BTagReaderPeekNext*
- *BTagReaderSkipContent*
- *BTagSwapHeader*
- *BTagWriterAppendBlob*
- *BTagWriterAppendDummyBytes*
- *BTagWriterAppendFillBytes*
- *BTagWriterAppendRaw*
- *BTagWriterAppendWString*
- *BTagWriterCreate*
- *BTagWriterCreateDynamic*
- *BTagWriterCreateSavePoint*
- *BTagWriterCreateSavePointDynamic*
- *BTagWriterDestroy*
- *BTagWriterDestroySavePoint*
- *BTagWriterEndTag*
- *BTagWriterFinish*
- *BTagWriterInit*

- *BTagWriterInit2*
- *BTagWriterRestoreSavePoint*
- *BTagWriterStartTag*
- *BTagWriterSwitchBuffer*
- *Buffer*
- *BufferPool*
- *BufferPoolFactoryArgs*
- *BufferPoolFactoryBase*
- *BufferToString*
- *BuildAndEnqueueV3Request*
- *BUS_INFO*
- *BUS_STATE*
- *BUS_TYPE*
- *BusScanConfHeader*
- *BusSpecific*
- *BUSSTATE*
- *BYTE_AS_BIT*
- *BYTE_TO_BCD*
- *BYTE_TO_GRAY*
- *BYTE_TO_HEXinASCII*
- *ByteBuffer*
- *ByteOrder*
- *C_TS_Type*
- *CAADiagDeviceDefault*
- *CAADiagTreeBase*
- *CAAReconfigureBase*
- *CalcCCIT16*
- *CalcHesseRepresentation*
- *CalcRootLin*
- *CalcRootParable*
- *CalculateCenter*
- *CalculatePropertyBufferSize*
- *CallbackNetVar*
- *CallbackTaskCodeNC*
- *CallFunctionByIndex*
- *CallGlueDeserializeParameters*
- *CallGlueFunctionParameterSet*
- *CallGlueSerializeReturnValues*
- *CANbus*
- *CANbus_Diag*
- *CANDiagnosis*
- *CANOPEN_DIAGNOSIS_INFO*
- *CANOPEN_KERNEL_ERROR*
- *CANOPEN_KERNEL_STATE*
- *CANOPEN_STATE*
- *CANopenDevice*
- *CANopenDevice_Diag*
- *CANopenDeviceSIL2*
- *CANopenDeviceUnsafe*
- *CANOpenDiagnosisInfo*
- *CANopenEvent*
- *CANopenManager*
- *CANopenManager_Diag*
- *CANopenManagerSIL2*
- *CANopenManagerUnsafe*

- *CANopenSafetyBase*
- *CanReconfigure*
- *CANRemoteDevice*
- *CANRemoteDevice_Diag*
- *CANRemoteDeviceSafe*
- *CANRemoteDeviceUnsafe*
- *CANRemoteModule_Diag*
- *CartesianToPolar*
- *CaseSensitiveNamedTreeNode*
- *CB_CALLBACK*
- *CCB*
- *CD522DoubleWordCounter*
- *CD522Encoder32Bit*
- *CD522FreqOut*
- *CD522FreqScan*
- *CD522FreqScan_PLUS*
- *CD522In*
- *CD522OneWordCounter*
- *CD522Out*
- *CD522PulseOut*
- *CD522PwmOut*
- *CD522ReadInput*
- *CD522SsiCnt*
- *CD522SsiCnt_PLUS*
- *CD522TwoWordCounters*
- *CD522WriteOutput*
- *CDClose*
- *CDIoctl*
- *CDLseek*
- *CDMmap*
- *CDMunmap*
- *CDOpen*
- *CDRead*
- *CDSV3Request*
- *CDWrite*
- *Ceil*
- *CeilF*
- *CERT_INFO*
- *CertCreate*
- *CertificateStoreOwner*
- *CertRemove*
- *ChainBuffer*
- *ChannelDiagnosisData*
- *ChannelErrorType*
- *ChannelProperties*
- *ChannelProperties_Type*
- *CharBufferPtr*
- *CharBufferString*
- *CHARCURVE*
- *CharCurve_DINT*
- *CharCurve_LREAL*
- *CharToUpper*
- *CHCAddressComponent*
- *CHCAddressType*
- *CHCPeerAddress*

- *CHCProtocolDataUnit*
- *Check*
- *CheckConfigSRDO*
- *CheckExpSubmodule*
- *CheckInverseData*
- *CheckReceivedSRDO*
- *CheckSymbolValidity*
- *CheckThumbString*
- *CIF_MemCpy*
- *CIF_MemSet*
- *CIF_StrLen*
- *CIFEXTMESSAGEHEADERtyp*
- *CIFFMSANYMESSAGEtyp*
- *CIFMESSAGEHEADERtyp*
- *CIFMESSAGERAWtyp*
- *CIFX_APPLICATION_CHANNEL_INFO*
- *CIFX_BOARD*
- *CIFX_BOARD_INFORMATION*
- *CIFX_CHANNEL*
- *CIFX_CHANNEL_INFO_BLOCK*
- *CIFX_CHANNEL_INFORMATION*
- *CIFX_COM_DIAGNOSTICS*
- *CIFX_COMMON_STATUS_BLOCK*
- *CIFX_COMMON_STATUS_BLOCK_MASTER*
- *CIFX_COMMUNOICATION_CHANNEL_INFO*
- *CIFX_DEV_INFO*
- *CIFX_DIRECTORY_ENTRY*
- *CIFX_ERROR_FIELD*
- *CIFX_GetBusActivationBeforeReset*
- *CIFX_GETSLAVECONNECTINFO_REQ*
- *CIFX_GETSLAVEHANDLE_CONF*
- *CIFX_GETSLAVEHANDLE_REQ*
- *CIFX_HANDSHAKE_CHANNEL_INFO*
- *CIFX_INDICATION_PARAM*
- *CIFX_MASTER_DIAG*
- *CIFX_MAX_PACKET*
- *CIFX_MEMORY_INFORMATION*
- *CIFX_PACKET*
- *CIFX_ResetConfigApplication*
- *CIFX_SYSTEM_CHANNEL_INFO*
- *CIFX_SYSTEM_INFO_BLOCK*
- *CIFX_xChannelBusState*
- *CIFX_xChannelClose*
- *CIFX_xChannelCommonStatusBlock*
- *CIFX_xChannelConfigLock*
- *CIFX_xChannelControlBlock*
- *CIFX_xChannelDownload*
- *CIFX_xChannelExtendedStatusBlock*
- *CIFX_xChannelFindFirstFile*
- *CIFX_xChannelFindNextFile*
- *CIFX_xChannelGetMBXState*
- *CIFX_xChannelGetPacket*
- *CIFX_xChannelGetPacketTimeout*
- *CIFX_xChannelGetSendPacket*
- *CIFX_xChannelHostState*

- *CIFX_xChannelInfo*
- *CIFX_xChannelIOInfo*
- *CIFX_xChannelIORead*
- *CIFX_xChannelIOReadSendData*
- *CIFX_xChannelIOWrite*
- *CIFX_xChannelOpen*
- *CIFX_xChannelOpen2*
- *CIFX_xChannelPLCActivateRead*
- *CIFX_xChannelPLCActivateWrite*
- *CIFX_xChannelPLCIsReadReady*
- *CIFX_xChannelPLCIsWriteReady*
- *CIFX_xChannelPLCMemoryPtr*
- *CIFX_xChannelPutPacket*
- *CIFX_xChannelReset*
- *CIFX_xChannelSetPacketTimeout*
- *CIFX_xChannelUpload*
- *CIFX_xChannelUserBlock*
- *CIFX_xChannelWatchdog*
- *CIFX_xDriverClose*
- *CIFX_xDriverEnumBoards*
- *CIFX_xDriverEnumChannels*
- *CIFX_xDriverGetErrorDescription*
- *CIFX_xDriverGetInformation*
- *CIFX_xDriverMemoryPointer*
- *CIFX_xDriverOpen*
- *CIFX_xMemCpy*
- *CIFX_xSysdeviceClose*
- *CIFX_xSysdeviceDownload*
- *CIFX_xSysdeviceFindFirstFile*
- *CIFX_xSysdeviceFindNextFile*
- *CIFX_xSysdeviceGetMBXState*
- *CIFX_xSysdeviceGetPacket*
- *CIFX_xSysdeviceInfo*
- *CIFX_xSysdeviceOpen*
- *CIFX_xSysdevicePutPacket*
- *CIFX_xSysdeviceReset*
- *CIFX_xSysdeviceUpload*
- *CIFXProfinetController*
- *CIFXProfinetControllerDiag*
- *CiModCi52x*
- *CiModCiClusterDiag*
- *CiModClusterDiag*
- *CiModClusterStatus*
- *CiModCmdQueueInput*
- *CiModDataAI523*
- *CiModDataAI531*
- *CiModDataAI561*
- *CiModDataAI562*
- *CiModDataAI563*
- *CiModDataAO523*
- *CiModDataAO561*
- *CiModDataAX521*
- *CiModDataAX522*
- *CiModDataAX561*
- *CiModDataCI521*

- *CiModDataCI522*
- *CiModDataDA501*
- *CiModDataDA502*
- *CiModDataDC522*
- *CiModDataDC523*
- *CiModDataDC532*
- *CiModDataDC561*
- *CiModDataDC562*
- *CiModDataDI524*
- *CiModDataDI561*
- *CiModDataDI562*
- *CiModDataDI571*
- *CiModDataDI572*
- *CiModDataDO524*
- *CiModDataDO526*
- *CiModDataDO561*
- *CiModDataDO562*
- *CiModDataDO571*
- *CiModDataDO572*
- *CiModDataDO573*
- *CiModDataDX522*
- *CiModDataDX531*
- *CiModDataDX561*
- *CiModDataDX571*
- *CiModDiag*
- *CiModDiagModInfo*
- *CiModDiagTableType*
- *CiModInput*
- *CiModParaAI523*
- *CiModParaAI531*
- *CiModParaAI561*
- *CiModParaAI562*
- *CiModParaAI563*
- *CiModParaAO523*
- *CiModParaAO561*
- *CiModParaAX521*
- *CiModParaAX522*
- *CiModParaAX561*
- *CiModParaCI521*
- *CiModParaCI522*
- *CiModParaDA501*
- *CiModParaDA502*
- *CiModParaDC522*
- *CiModParaDC523*
- *CiModParaDC532*
- *CiModParaDC561*
- *CiModParaDC562*
- *CiModParaDI524*
- *CiModParaDI561*
- *CiModParaDI562*
- *CiModParaDI571*
- *CiModParaDI572*
- *CiModParaDO524*
- *CiModParaDO526*
- *CiModParaDO561*

- *CiModParaDO562*
- *CiModParaDO571*
- *CiModParaDO572*
- *CiModParaDO573*
- *CiModParaDX522*
- *CiModParaDX531*
- *CiModParaDX561*
- *CiModParaDX571*
- *CIP_Attribute*
- *CIPClass*
- *CIPCommonService*
- *CIPHER_LIST*
- *CLASS_INFO*
- *ClassCreate*
- *ClassDelete*
- *ClassFree*
- *CLClient*
- *CLClientOptions*
- *CLClientState*
- *Client*
- *CLIENT_ACCEPT*
- *CLIENT_REPLY*
- *ClientCreatableObjects*
- *ClientRequest*
- *ClientRequestMaskWriteRegister*
- *ClientRequestRead*
- *ClientRequestReadBits*
- *ClientRequestReadCoils*
- *ClientRequestReadDiscreteInputs*
- *ClientRequestReadHoldingRegisters*
- *ClientRequestReadInputRegisters*
- *ClientRequestReadRegisters*
- *ClientRequestReadWriteMultipleRegisters*
- *ClientRequestWriteMultiple*
- *ClientRequestWriteMultipleCoils*
- *ClientRequestWriteMultipleRegisters*
- *ClientRequestWriteSingle*
- *ClientRequestWriteSingleCoil*
- *ClientRequestWriteSingleRegister*
- *ClientSerial*
- *ClientSide*
- *ClientTCP*
- *CLOCK*
- *CLOCK_DT*
- *CloneMessage*
- *Close*
- *CLRequestState*
- *CLServer*
- *CLServerOptions*
- *CM579EtherCATDeviceInfoType*
- *CM582ProfibusDeviceInfoType*
- *CM589ProfinetDeviceInfoType*
- *CM592CommErrorInfo*
- *CM592CommStatus*
- *CM592Control*

- *CM592DPV1Masc1Read*
- *CM592DPV1Masc1Write*
- *CM592ExtDiagData*
- *CM592ProfibusDeviceInfoType*
- *CM592ReadInput*
- *CM592ReadOutput*
- *CM592SlaveDiagnosis*
- *CM592SlaveStates*
- *CM592StateBits*
- *CM592StationStatus_1*
- *CM592StationStatus_2*
- *CM592StationStatus_3*
- *CM592SystemDiagnosis*
- *Cm598Base*
- *Cm598CanInfo*
- *Cm598CanInfoType*
- *Cm598CanMessageType*
- *Cm598CanMsgRec*
- *Cm598CanMsgRecEvt*
- *Cm598CanMsgSend*
- *Cm598CanopenComErrorType*
- *Cm598CanopenNmt*
- *Cm598CanopenSdoRead*
- *Cm598CanopenSdoWrite*
- *Cm598CanopenState*
- *Cm598CanopenStateBitsType*
- *Cm598CanopenStateType*
- *CM598DeviceInfoType*
- *Cm598NmtCmd*
- *CMAddComponent*
- *CMAddComponent2*
- *CMExitComponent*
- *CMGetComponentByName*
- *CMGetCoreVersion*
- *CMInitComponent*
- *CmpIoDrvC*
- *CmpIoDrvWrapper*
- *CmpLogAsyncFB*
- *CmpTlsAccept*
- *CmpTlsBufferDataReceived*
- *CmpTlsBufferDataSent*
- *CmpTlsBufferDataToSendAvailable*
- *CmpTlsBufferOpen*
- *CmpTlsClose*
- *CmpTlsConnect*
- *CmpTlsCreateContext*
- *CmpTlsCreateContext2*
- *CmpTlsFreeContext*
- *CmpTlsMethod*
- *CmpTlsRead*
- *CmpTlsShutdown*
- *CmpTlsWrite*
- *CMRemoveComponent*
- *CMShutDown*
- *CMUtlcwstrcpy*

- *CMUtlSafeStrCpy*
- *CMUtlStrICmp*
- *CMUtlUtf8ToW*
- *CMUtlwstrcpy*
- *CMUtlWToUtf8*
- *CNCT*
- *COBID*
- *CodeMClose*
- *CodeMDecrypt*
- *CodeMEncrypt*
- *CodeMGetContentByFirmcode*
- *CodeMGetContentByFirmcode2*
- *CodeMGetExpirationTime*
- *CodeMGetFeatureMapByFirmcode*
- *CodeMGetFirst*
- *CodeMGetInfo*
- *CodeMGetName*
- *CodeMGetNext*
- *CodeMGetQuantity*
- *CodeMGetUnitCounter*
- *CodeMOpen*
- *CodeWriter*
- *COLLECTION_ERROR*
- *COM_CFG_FORMAT_ENUM*
- *COM_MOD_FCT22_TYPE*
- *COM_MOD_FCT23_TYPE*
- *COM_PORT_ID*
- *CombineDateTime*
- *ComGetCaaSerialComConfig*
- *ComGetIdByName*
- *CommAbbxUsrMsgGet*
- *CommAbbxUsrMsgRec*
- *CommAbbxUsrMsgSend*
- *CommandHandler*
- *CommandManager*
- *CommStatus*
- *CommunicationErrorCIFX*
- *Compare*
- *CompareString*
- *CompareWString*
- *CompatibilitySafeGetPrepareExitCommProcessingLastCall*
- *CompatibilitySafeSetPrepareExitCommProcessingFurtherCallNecessary*
- *ComponentBase*
- *ComponentManager*
- *ComponentPseudo*
- *ComponentRenamed*
- *ComponentSimple*
- *CONCAT*
- *CONFIG_SRDO*
- *ConfigError*
- *ConfigGetConnector*
- *ConfigGetFirstChild*
- *ConfigGetFirstConnector*
- *ConfigGetNextChild*
- *ConfigGetNextConnector*

- *ConfigGetParameter*
- *ConfigGetParameterLength*
- *ConfigGetParameterValueByte*
- *ConfigGetParameterValueDword*
- *ConfigGetParameterValuePointer*
- *ConfigGetParameterValueWord*
- *ConfigureByString*
- *Connect*
- *Connect2*
- *ConnectionHandler*
- *ConnectionSetup*
- *Connector*
- *ConnectorFlagController*
- *ConnectorState*
- *ContentFeatureFlags*
- *ControllerConfigUtil*
- *ControllerState*
- *ConvertNSecToTick*
- *ConvertSystimedateToUTC*
- *ConvertSystimedateUsingLDate*
- *ConvertSysTimeValueToLWord*
- *ConvertTickToNSec*
- *ConvertTickToUSec*
- *ConvertTimestampToLDateAndTime*
- *ConvertUSecToTick*
- *ConvertUTF16toUTF8*
- *ConvertUTF8toUTF16*
- *ConvThumbToBytes*
- *ConvThumbToString*
- *Copy*
- *CopyBufferData*
- *COUNT*
- *COUNT_TO_UDINT*
- *COUNT_TO_UINT*
- *COUNT_TO_ULINT*
- *CPU_PROD_READ_ASYNC*
- *CpuCoreBindingDesc*
- *CpuCoreBits*
- *CRC16_CCITT*
- *CRC16_generic*
- *CRC16_Modbus*
- *CRC16_standard*
- *CRC16Finish*
- *CRC16Init*
- *CRC16Update*
- *CRC32*
- *CRC32Finish*
- *CRC32Init*
- *CRC32Update*
- *CRC32Update2*
- *CREATE_ID*
- *CreateBuffer*
- *CreateIdAreaReceiver*
- *CreateInstance*
- *CreateMaskReceiver*

- *CreateMessage*
- *CreateSingleIdReceiver*
- *CreateTextFromString*
- *CreateTextFromWString*
- *CreateXMLParser2*
- *CredentialsHandling*
- *CrossProduct*
- *CrossProductNormed*
- *CryptoAsymmetricDecrypt*
- *CryptoAsymmetricEncrypt*
- *CryptoDeletePrivateKey*
- *CryptoDeriveKey*
- *CryptoExportAsymmetricKey*
- *CryptoGenerateAsymmetricKeyPair*
- *CryptoGenerateHash*
- *CryptoGenerateRandomNumber*
- *CryptoGetAlgorithmById*
- *CryptoGetAsymmetricKeyLength*
- *CryptoGetFirstAlgorithm*
- *CryptoGetNextAlgorithm*
- *CryptoHMACSign*
- *CryptoHMACVerify*
- *CryptoImportAsymmetricKey*
- *CryptoKeyExit*
- *CryptoKeyInit*
- *CryptoLoadPrivateKey*
- *CryptoRtsByteStringExit*
- *CryptoRtsByteStringInit*
- *CryptoRtsByteStringInit2*
- *CryptoSignatureGenerate*
- *CryptoSignatureVerify*
- *CryptoStorePrivateKey*
- *CryptoSymmetricDecrypt*
- *CryptoSymmetricEncrypt*
- *CSMD_SVC_ERROR_CODES*
- *CTD*
- *CTU*
- *CTUD*
- *CustomRequestQueue*
- *CustomRequestResponse*
- *CustomRequestState*
- *CWCHAR*
- *DATA*
- *DATA_TYPE*
- *DataCopyToBlob*
- *DataItem*
- *DataItemAndPtrVectors*
- *DataItemBase*
- *DataItemItfVector*
- *DataItemList*
- *DataItemListPublic*
- *DataItemListPublicPersistant*
- *DataItemLocation*
- *DataItemPtrVector*
- *DataItemVector*

- *DataRepresentation*
- *Datasource*
- *DataSourceError*
- *DataSourceMonitoringState*
- *Datasources*
- *DatasourcesAction*
- *DatasourcesActionRecord*
- *DatasourceShutdownInfo*
- *DatasourcesMgr*
- *DataSourcesQualityChecker*
- *DataSourceState*
- *DateConcat*
- *DateSplit*
- *DateTime*
- *DATETIME_TO_RTS_SYSTIMEDATE*
- *DateTimeFromWeek*
- *DateTimeProvider*
- *DateTimeToString*
- *DateTimeToTimestamp*
- *DAY*
- *DayOfWeek*
- *DAYS*
- *DCC_SvcAppHook*
- *DCP_DeviceData*
- *DCP_DeviceRole*
- *DCP_Error*
- *DCP_FilterData*
- *DCP_FilterMode*
- *DCP_FilterOptions*
- *DCP_Get*
- *DCP_GetOptions*
- *DCP_Identify*
- *DCP_Reset*
- *DCP_ResetMode*
- *DCP_Service*
- *DCP_Set*
- *DCP_SetData*
- *DCP_SetOptions*
- *DeallocStackAllocatedContentBuffer*
- *DebugItfAddrToItfPtr*
- *Decode*
- *DECODE_IOL_STATUS*
- *DecodeClass*
- *DecodeEmcyCOBID*
- *DecodeEvent*
- *DecodeHeartbeatConsumerSettings*
- *DecodeLastRune*
- *DecodePDOCOBID*
- *DecodePDOMappingEntry*
- *DecodeRune*
- *DecodeSyncCOBID*
- *DefaultAlarmFilterCriteria*
- *DefaultIParameterDB*
- *DefaultIParData*
- *Deg2Rad*

- *DELETE*
- *Delete*
- *DeleteBuffer*
- *DeleteInstance*
- *DeleteReceiver*
- *DERIVATIVE*
- *Derivative*
- *DeserializeHexReal*
- *DEVICE*
- *DEVICE_INFO*
- *DEVICE_STATE*
- *DEVICE_TRANSITION_STATE*
- *DEVICE_TYPE*
- *DeviceAR*
- *DeviceAR_State*
- *DeviceConfigUtil*
- *DeviceDateTime*
- *DeviceIdentification*
- *DeviceInfo*
- *DeviceIterator*
- *DeviceState*
- *DeviceStatusT*
- *DEVINFO*
- *Diag*
- *DIAG_HISTORY_TXT_TYPE*
- *DIAG_TXT_TYPE*
- *DIAG_VAL_TYPE*
- *DiagHistory*
- *DiagHistoryValToTxt*
- *DiagMessageFactory*
- *DIAGNOSIS_INFO*
- *DiagnosisDataBuffer*
- *DiagnosisDataReader*
- *DiagnosisDirection*
- *DiagnosisInformationUSI*
- *DiagnosisRecordIndex*
- *DiagnosisSeverity*
- *DiagnosisSource*
- *DiagValToTxt*
- *DiagVerifyTextListCallback*
- *DINT_TO_SIGNED*
- *DintElement*
- *DintElementFactory*
- *DintSetBitBased*
- *DintSetFull*
- *DintToDintMap*
- *DintVector*
- *DirClose*
- *DirCopy*
- *DirCreate*
- *DirectAssigner*
- *DirectIOBits16*
- *DirectIOBits8*
- *direction*
- *Directory*

- *DriverDiag*
- *DriverGetSize*
- *DriverOpenH*
- *DriverOpenP*
- *DRV_PDRIVE_PRM_REQ_ERROR*
- *DRV_PDRIVE_PRM_TYPE*
- *DrvControlACS*
- *DrvControlCANCiA402*
- *DrvControlDCS*
- *DrvControlModbusACS*
- *DrvControlModbusDCS*
- *DrvControlModbusEng*
- *DrvDataType*
- *DrvDataTypeInternal*
- *DrvModbusRead*
- *DrvModbusReadWrite23*
- *DrvModbusRtu*
- *DrvModbusRtuBroadcast*
- *DrvModbusTcp*
- *DrvModbusWrite*
- *DrvModFct23Type*
- *DrvModMastType*
- *DrvModPara32Bit*
- *DrvPdPrmDpv1DataType*
- *DrvPnRead*
- *DrvPnWrite*
- *DrvScaling*
- *DS_DISK_STATUS*
- *DS_EOL_INFO*
- *DS_LIFETIME_USED*
- *DT_TO_INT64*
- *DT_TO_ISO8601*
- *DT_TO_REAL8*
- *DT_to_Timestamp*
- *DT_to_Timestamp2*
- *DTC*
- *DTCBufferWriter*
- *DTCLogger*
- *DTConcat*
- *DTCProvider*
- *DTR_CONTROL*
- *DTSplit*
- *DTToOpcDate*
- *DTU_GETDATEANDTIME_PARAMS*
- *DTU_GETTIMEZONEINFORMATION_PARAMS*
- *DTU_SETDATEANDTIME_PARAMS*
- *DTU_SETTIMEZONEINFORMATION_PARAMS*
- *Dummy*
- *DummyJob*
- *DURATION*
- *DURATION_TO_LTIME*
- *DURATION_TO_TIME*
- *DUT_GPIOPin*
- *DWORD_AS_BIT*
- *DWORD_SEQ_LET*

- *DWORD_SEQ_LT*
- *DWORD_TO_BCD*
- *DWORD_TO_GRAY*
- *DWORD_TO_HANDLE*
- *DWORD_TO_IDENT*
- *DWORD_TO_PVOID*
- *DwordVector*
- *DynamicTextChangeLanguage*
- *DynamicTextGetCurrentLanguage*
- *DynamicTextGetDefaultText*
- *DynamicTextGetDefaultTextW*
- *DynamicTextGetText*
- *DynamicTextGetTextW*
- *DynamicTextIterateIndices*
- *DynamicTextLoadDefaultTexts*
- *DynamicTextRegisterFile*
- *DynamicTextRegisterPath*
- *DynamicTextReloadTexts*
- *DynamicTextUnRegisterFile*
- *DynamicTraceLoader*
- *DynamicTraceLoaderRemote*
- *EAlarmStorageReaderErrors*
- *EAlarmTableParts*
- *EAlarmType*
- *ECAT_402ParameterHoming_APP*
- *ECAT_CiA_Object_App*
- *ECAT_CiA402_Control_App*
- *ECAT_CiA402_TouchProbe_App*
- *ECAT_HomingOnTouchProbe_APP*
- *ECAT_Read_Byte_App*
- *ECAT_Read_Coe_List_App*
- *ECAT_Read_DInt_App*
- *ECAT_Read_Int_App*
- *ECAT_Write_Byte_App*
- *ECAT_Write_Coe_List_App*
- *ECAT_Write_DInt_App*
- *ECAT_Write_Int_App*
- *EcatBusDiag*
- *EcatBusGetDCMaxDeviation*
- *EcatBusSetState*
- *EcatCoeRead*
- *EcatCoeWrite*
- *EcatDeviceIdentification*
- *EcatDeviceInfoData*
- *EcatDeviceTypeIdentification*
- *EcatGetExtSyncInfo*
- *EcatMasterGetCPULoad*
- *EcatMasterGetFrameLossCount*
- *EcatMasterGetMemInfo*
- *EcatMasterGetThresholdCount*
- *EcatMasterGetTimingInfo*
- *EcatRegisterRead*
- *EcatRegisterWrite*
- *EcatScanTopology*
- *EcatScanTopologyStop*

- *EcatSlvDiag*
- *EcatSlvGetDCInfo*
- *EcatSlvGetMDPModules*
- *EcatSlvGetState*
- *EcatSlvReadESCVersion*
- *EcatSlvReadLostLinkCnt*
- *EcatSlvReadRxErrorCnt*
- *EcatSlvSetState*
- *EcatSoeRead*
- *EcatSoeWrite*
- *EcatStartCom*
- *EcatState*
- *EcatStopCom*
- *EcatSync*
- *EcatVendor*
- *EcatVendorIDList*
- *EcatVendorName2Device*
- *ECM_IF_DC_CONTROL_STATUS_E*
- *ECM_IF_GET_SLAVE_DC_INFO_FLAGS_E*
- *EColorSetting*
- *ECUSTATE*
- *EDBActiveIndex*
- *EDBType*
- *eDeviceState*
- *EdgeTriggeredBehaviourModelBase*
- *EdgeTriggeredTimingControlledBehaviourModelBase*
- *EEthernetState*
- *eFastCounter*
- *EFillingStyle*
- *EFilterCriteriaActivity*
- *EFilterLatchContent*
- *EFilterTimeRangeType*
- *EImageStyle*
- *EIP_CloseClass3Connection*
- *EIP_OpenClass3Connection*
- *EIP_SendClass3ConnectedMessage*
- *EIP_SendUnconnectedMessage*
- *ElaborateLatchFilterCriteria*
- *ElaborateTimeRangeFilterCriteria*
- *Element*
- *EMCY_DATA*
- *EMCY_ERROR*
- *eModulName*
- *EnableSyncService*
- *Encode*
- *EncodeEmcyCOBID*
- *EncodeHeartbeatConsumerSettings*
- *EncodePDOCOBID*
- *EncodePDOMappingEntry*
- *EncodeRune*
- *EncodeSpec*
- *EncodeSyncCOBID*
- *ENDIANESS*
- *EndpointDescriptionToString*
- *EndpointReceiver*

- *EnqueuedRequest*
- *ENUM61850_BASIC_TYPES*
- *ENUM61850_CLOCK_SYNC_MODE*
- *ENUM61850_DataPoint_Type*
- *ENUM61850_SIM_MODE*
- *EnumAttributes*
- *EnumCommand*
- *EnumErrors*
- *EnumUnitTest*
- *EnumValues*
- *EnXYChartClientActity*
- *EnXYChartDataProviderAxisVar*
- *EnXYChartDataProviderCurveVar*
- *EnXYChartDataProviderVar*
- *EnXYChartUpdateType*
- *EOF*
- *eParaState*
- *ERectSetting*
- *ERROR*
- *Error*
- *ERROR_ID*
- *ERROR_INFO*
- *ErrorCode*
- *ErrorCode1_RW*
- *ErrorCodesOB*
- *ErrorInjection*
- *ErrorPLCHToString*
- *ErrorToString*
- *EShadowStyle*
- *ESpecial_FP_Value*
- *ETC_ADS_IoLinkRead*
- *ETC_ADS_IoLinkWrite*
- *ETC_CO_Emergency*
- *ETC_CO_ERROR*
- *ETC_CO_MODE*
- *ETC_CO_SdoInfoGeEntryDescription*
- *ETC_CO_SdoInfoGetObjectDescription*
- *ETC_CO_SdoInfoGetODList*
- *ETC_CO_SdoRead*
- *ETC_CO_SdoRead_Access*
- *ETC_CO_SdoRead_Channel*
- *ETC_CO_SdoRead4*
- *ETC_CO_SdoReadDWord*
- *ETC_CO_SdoWrite*
- *ETC_CO_SdoWrite_Access*
- *ETC_CO_SdoWrite4*
- *ETC_CO_SdoWriteDWord*
- *ETC_FoE_Download*
- *ETC_FoE_Upload*
- *ETC_LASTERROR*
- *ETC_MASTER_STATE*
- *ETC_SDO_INFO_LIST_TYPE*
- *ETC_SDO_INFO_OBJECT_CODE*
- *ETC_SLAVE_STATE*
- *ETC_SoE_Cmd*

- *ETC_SOE_ERROR*
- *ETC_SoE_IDNRead*
- *ETC_SoE_IDNRead4*
- *ETC_SoE_IDNWrite*
- *ETC_SoE_IDNWrite4*
- *ETC_VoE_SendReceive*
- *ETCDeviceIdentMode*
- *ETCERRORCODES*
- *ETCMasterStack*
- *ETCSlave*
- *ETCSlave_Diag*
- *ETCSlaveStack*
- *ETH_MOD_FCT22_TYPE*
- *ETH_MOD_FCT23_TYPE*
- *EthDNSResolve*
- *EtherCATDevice*
- *EthercatMaster_GetVersion*
- *EthernetState*
- *EthIcmpPing*
- *EthOwnIP*
- *EthOwnIPInfo*
- *EthSetOwnIP*
- *EthSetRtoMin*
- *ETHx_ICMP_PING*
- *ETHx_MOD_CONFIG*
- *ETHx_MOD_INFO*
- *ETHx_MOD_MAST*
- *ETHx_OWN_IP*
- *ETHx_OWN_IP_INFO*
- *ETraceGradientType*
- *ETrendStorageGraphType*
- *ETrendStoragePenStyle*
- *ETrendStorageReaderErrors*
- *ETrendStorageReaderStep*
- *ETrig*
- *ETrigA*
- *ETrigATI*
- *ETrigATITo*
- *ETrigATo*
- *ETrigTI*
- *ETrigTIA*
- *ETrigTITo*
- *ETrigTo*
- *ETrigToA*
- *ETrigToTI*
- *ETrigToTIA*
- *EVENT*
- *EVENT_CLASS*
- *EVENT_SOURCE*
- *EventClose2*
- *EventCreate*
- *EventCreate2*
- *EventCreateEventID*
- *EventDelete2*
- *EventElementData*

- *EventGetClass*
- *EventGetEvent*
- *EventIdToString*
- *EventListener*
- *EventOpen*
- *EventParam*
- *EventParam2*
- *EventPost*
- *EventPost2*
- *EventPostByEvent*
- *EventPostByEvent2*
- *EventQueueAndElement*
- *EventRegisterCallback*
- *EventRegisterCallback2*
- *EventRegisterCallbackFunction*
- *EventRegisterCallbackFunction2*
- *EventRegisteredCallbacks*
- *EventUnregisterCallback*
- *EventUnregisterCallbackFunction*
- *EventUnregisterCallbackFunction2*
- *EVT_BACNET_ACKALARM*
- *EVT_BACNET_ADDELEMENT*
- *EVT_BACNET_ADDRESSCHANGECALLBACK*
- *EVT_BACNET_BACKUPRESTOREPROGRESSCALLBACK*
- *EVT_BACNET_CHANGEOFVALUEEVENTS*
- *EVT_BACNET_CLIENTEVENTCALLBACK*
- *EVT_BACNET_CLIENTSTATUSCALLBACK*
- *EVT_BACNET_CLIENTUNSUBSCRIBECOMPLETECALLBACK*
- *EVT_BACNET_CLIENTVALUECALLBACK*
- *EVT_BACNET_CONFCOVNOTIFICATION*
- *EVT_BACNET_CONFEVENTNOTIFICATION*
- *EVT_BACNET_CONFPRIVATEXFER*
- *EVT_BACNET_CONFTEXTMESSAGE*
- *EVT_BACNET_CREATEOBJECT*
- *EVT_BACNET_DCC*
- *EVT_BACNET_DELETEOBJECT*
- *EVT_BACNET_GETALARMSUMMARY*
- *EVT_BACNET_GETENROLLMENTSUMMARY*
- *EVT_BACNET_GETEVENTINFO*
- *EVT_BACNET_IACTIONERRCALLBACK*
- *EVT_BACNET_IAM*
- *EVT_BACNET_IHAVE*
- *EVT_BACNET_INTRINSICEVENTS*
- *EVT_BACNET_LIFESAFETYOPERATION*
- *EVT_BACNET_NETWORKEVENTS*
- *EVT_BACNET_OBJECTIDCHANGECALLBACK*
- *EVT_BACNET_OSTIMEPROVIDERCALLBACK*
- *EVT_BACNET_READFILE*
- *EVT_BACNET_READPROPERTY*
- *EVT_BACNET_READPROPERTY_TO_STRING*
- *EVT_BACNET_READPROPERTYCALLBACK*
- *EVT_BACNET_READPROPERTYMULT*
- *EVT_BACNET_READPROPERTYRELEASECALLBACK*
- *EVT_BACNET_READRANGE*
- *EVT_BACNET_REINITDEV*

- *EVT_BACNET_REMOVEELEMENT*
- *EVT_BACNET_STACKACTION*
- *EVT_BACNET_SUBSCRIBECOV*
- *EVT_BACNET_SUBSCRIBECOVPROPERTY*
- *EVT_BACNET_TIMESYNC*
- *EVT_BACNET_UNCONFCOVNOTIFICATION*
- *EVT_BACNET_UNCONFEVENTNOTIFICATION*
- *EVT_BACNET_UNCONFPRIVATEXFER*
- *EVT_BACNET_UNCONFTEXTMESSAGE*
- *EVT_BACNET_UTCTIMESYNC*
- *EVT_BACNET_WHOHAS*
- *EVT_BACNET_WHOIS*
- *EVT_BACNET_WRITEFILE*
- *EVT_BACNET_WRITEGROUP*
- *EVT_BACNET_WRITEPROPERTY*
- *EVT_BACNET_WRITEPROPERTY_TO_STRING*
- *EVT_BACNET_WRITEPROPERTYCALLBACK*
- *EVT_BACNET_WRITEPROPERTYCALLBACK2*
- *EVT_BACNET_WRITEPROPERTYMULT*
- *EVTPARAM_BeforeCheckFirmware*
- *EVTPARAM_CIFX_GetFirmware*
- *EVTPARAM_CIFX_LoadFirmware*
- *EVTPARAM_CIFX_xChannelClose*
- *EVTPARAM_CIFX_xChannelOpen*
- *EVTPARAM_CmpApp*
- *EVTPARAM_CmpAppAllBootAppsLoaded*
- *EVTPARAM_CmpAppComm*
- *EVTPARAM_CmpAppCommCycle*
- *EVTPARAM_CmpAppConfig*
- *EVTPARAM_CmpAppDeny*
- *EVTPARAM_CmpAppDenyDelete*
- *EVTPARAM_CmpAppDenyLoadBootproject*
- *EVTPARAM_CmpAppDenyStart*
- *EVTPARAM_CmpAppDenyStop*
- *EVTPARAM_CmpAppException*
- *EVTPARAM_CmpAppExit*
- *EVTPARAM_CmpAppOEMServiceTag*
- *EVTPARAM_CmpAppOperatingStateChanged*
- *EVTPARAM_CmpAppPrepareLoadBootproject*
- *EVTPARAM_CmpAppRegisterBootproject*
- *EVTPARAM_CmpAppReset*
- *EVTPARAM_CmpAppResetAllApplications*
- *EVTPARAM_CmpAppRetainBackupState*
- *EVTPARAM_CmpAppSourceDownload*
- *EVTPARAM_CmpAppStateChanged*
- *EVTPARAM_CmpAppStop*
- *EVTPARAM_CmpChS_ChannelClosed*
- *EVTPARAM_CmpChS_ChannelOpened*
- *EVTPARAM_CmpIecTask*
- *EVTPARAM_CmpIecTask2*
- *EVTPARAM_CmpIoMgr*
- *EVTPARAM_CmpLogAdd*
- *EVTPARAM_CmpMgr_DisableOperation*
- *EVTPARAM_CmpMgr_LicenseRequest*
- *EVTPARAM_CmpMgr_PrepareExitCommProcessing*

- *EVTPARAM_CmpMgr_Shutdown*
- *EVTPARAM_CmpOPCUAServerSessionsChanged*
- *EVTPARAM_CmpSrv*
- *EVTPARAM_CmpSupervisor_StateChanged*
- *EVTPARAM_CmpTraceMgr_Packet*
- *EVTPARAM_CmpTraceMgr_Record*
- *EVTPARAM_CmpXMLData*
- *EVTPARAM_CmpXMLEnd*
- *EVTPARAM_CmpXMLStart*
- *EVTPARAM_DownloadProgress*
- *EVTPARAM_PacketConfirmation*
- *EVTPARAM_PacketIndication*
- *EVTPARAM_PacketUnhandled*
- *EVTPARAM_PlcShellCommand*
- *EVTPARAM_UploadProgress*
- *ExampleDataModel*
- *ExceptionCodes*
- *export*
- *ExpressionResult*
- *ExpSubmodule*
- *EXTRACT*
- *F_TRIG*
- *FactoryBase*
- *FailureReadRequest*
- *FaultStatus*
- *FbChangeVisu*
- *FbCloseDialog*
- *FBFileTransfer*
- *fbIEC61850_Subs_ASN1_CheckData*
- *fbIEC61850_Subs_ASN1_Decoder*
- *fbIEC61850_Subs_ASN1_Decoder_CheckDataNum*
- *fbIEC61850_Subs_ASN1_Decoding_Data*
- *fbIEC61850_Subscriber*
- *FbIterateClients*
- *FbOpenDialog*
- *FbOpenDialogExtended*
- *FctIncreaseElemRectForLine*
- *FctPointIntersectsRectangle*
- *FD_CLR*
- *FILE_DIR_ENTRY*
- *FILENAME*
- *FileNameString*
- *FillNodeInfoInt*
- *FIND*
- *Find2*
- *FindBlock*
- *FindByte*
- *FIRMWAREINFO*
- *FlatClass*
- *FlatCreateH*
- *FlatCreateP*
- *FlatDelete*
- *FlatDisable*
- *FlatEnable*
- *FlatGetSize*

- *FlatRead*
- *FlatTest*
- *FlatUpdate*
- *FLOAT*
- *FLOAT_TO_LREAL*
- *FLOAT_TO_REAL*
- *Floor*
- *FloorF*
- *Flush*
- *FMI*
- *fmod*
- *FORMAT_MODE*
- *FormatDateTime*
- *FormatTimestamp*
- *FormatTimestamp2*
- *FormatTypedValue*
- *FrameManager*
- *FrameRegistrationData*
- *FreeMessage*
- *FreeStackAllocatedMemory*
- *FreeXMLParser*
- *FREQ_MEASURE*
- *FromBACnetBitString*
- *FromBACnetBoolean*
- *FromBACnetDate*
- *FromBACnetDateRange*
- *FromBACnetDateTime*
- *FromBACnetDevObjPropReference*
- *FromBACnetSetpointReference*
- *FromBACnetString*
- *FromBACnetTime*
- *FromBACnetTimeStamp*
- *FSLState*
- *FunctionCodes*
- *funIEC61850_GetReportHeaderLen*
- *funIEC61850_MMSTYPE_TO_STRING*
- *funIEC61850_Subs_Bits_SwapRight*
- *funIEC61850_Subs_InitDatapoint*
- *funIEC61850_SubsCheckDataNum*
- *GEN*
- *GEN_MODE*
- *Generic_Service*
- *Get_Attribute_Single*
- *Get_Attributes_All*
- *GET_CANOPEN_KERNEL_STATE*
- *GET_LOCAL_NODE_ID*
- *GET_STATE*
- *GetAttribute*
- *GetBACnetDataTypeSize*
- *GetBACnetPropertyDataType*
- *GetBaudrate*
- *GETBIT*
- *GetBitStringFromContents*
- *GetBitValue*
- *GetBooleanProperty*

- *GetBoolFromContents*
- *GetBufferSize*
- *GetBusAlarm*
- *GetBusError*
- *GetBusload*
- *GetBusScan*
- *GetBusState*
- *GetCallback*
- *GetCallbackTypeOfEventId*
- *GetCBTypeOfEventId*
- *GetCertHandle*
- *GetCertRenewTime*
- *GetChar*
- *GetCiAState*
- *GetClass*
- *GetClassInfo*
- *GetCompany*
- *GetConfigType*
- *GetConnectionInfo*
- *GetConnectionState*
- *GetControllerNode*
- *GetCurrentUtcOffset*
- *GetDateAndTime*
- *GetDateFromContents*
- *GetDateRangeFromContents*
- *GetDateTime*
- *GetDateTimeFromContents*
- *GetDayOfWeek*
- *GetDeviceError*
- *GetDeviceInfo*
- *GetDeviceNode*
- *GetDevObjPropReferenceFromContents*
- *GetDiagnosis*
- *GetElapsedTimeInNSec*
- *GetElapsedTimeInUSec*
- *GetEventIdOfCallbackType*
- *GetEventIdOfCBType*
- *GetHandleOfCallback*
- *GetHostname*
- *GetID*
- *GetIDeviceInstByIoAddr*
- *GetInfo*
- *GETIO_PART*
- *GetIPAddress*
- *GetLatchVarColumnID*
- *GetLibVersion*
- *GetLibVersionNumber*
- *GetLINTValue*
- *GetLINTValue2*
- *GetLINTValue3*
- *GetLocalDateTime*
- *GetLocalTime*
- *GetLostCounter*
- *GetLrealFromContents*
- *GetLRealSpecialVal*

- *GetMessageDataPointer*
- *GetMessageId*
- *GetMessageLength*
- *GetMsgCount*
- *GetNetId*
- *GetNextNode*
- *GetNodeDepth*
- *GetNumberActiveCallbacks*
- *GetNumberProperty*
- *GetObjectIDFromContents*
- *GetParent*
- *GetPlcIdent*
- *GetPos*
- *GetProperty*
- *GetRealFromContents*
- *GetRealSpecialVal*
- *GetReceiveCounter*
- *GetReceiveErrorCounter*
- *GetReceivePoolSize*
- *GetReceiveQueueLength*
- *GetRedundancyState*
- *GetRoot*
- *GetSetpointReferenceFromContents*
- *GetSignedFromContents*
- *GetSize*
- *GetSpecificDeviceError*
- *GetState*
- *GetSubmoduleDiagnosis*
- *GetSupplierVersion*
- *GetSyncInformation*
- *GetSystemTimeZone*
- *GetText*
- *GetTextListInfo*
- *GetTextProperty*
- *GetTextProperty2*
- *GetTextW*
- *GetTick*
- *GetTime*
- *GetTimeFromContents*
- *GetTimeStamp*
- *GetTimeStampsDifference*
- *GetTimeZoneInformation*
- *GetTitle*
- *GetTransmitCounter*
- *GetTransmitErrorCounter*
- *GetTransmitPoolSize*
- *GetTransmitQueueLength*
- *GetUnitTestStatus*
- *GetUnsignedFromContents*
- *GetVersion*
- *GetVersionProperty*
- *GetWStringFromContents*
- *GlobalTextList*
- *GPIOSysfs*
- *GPIOSysfsDiag*

- *GRAY_TO_BYTE*
- *GRAY_TO_DWORD*
- *GRAY_TO_WORD*
- *GuidHelper*
- *HaModAIO*
- *HaModCallbackStop*
- *HaModControl*
- *HaModCtd*
- *HaModCtu*
- *HaModCtud*
- *HaModDataSync*
- *HaModDerivative*
- *HaModDiag*
- *HaModDIO*
- *HaModEthFrame*
- *HaModEthFrameHeader*
- *HaModIntegral*
- *HaModPid*
- *HaModPidFixCycle*
- *HaModRampInt*
- *HaModRampReal*
- *HaModStatus*
- *HaModStatusLifecom2*
- *HaModStatusPlc*
- *HaModTof*
- *HaModTon*
- *HaModVisuData*
- *HANDLE*
- *HANDLE_TO_DWORD*
- *HANDLE_TO_LWORD*
- *HANDLE_TO_WORD*
- *HandleChannelError*
- *HandleReply*
- *HandleStore*
- *HasAlarmStorageRecordLimit*
- *HashCodeFromString*
- *HashCodeFromWString*
- *HashTable*
- *HashTableFactory*
- *HEADER_TAG*
- *HeapInspectionInfo*
- *HEXinASCII_TO_BYTE*
- *HexStrToLReal*
- *HexStrToReal*
- *HighByte*
- *HighWord*
- *HIL_LiveList*
- *HilscherCardMgr*
- *HistoricalActiveAlarmRowID*
- *history*
- *HOSTNAME*
- *HOUR*
- *HYSTERESIS*
- *Hysteresis_DINT*
- *Hysteresis_LREAL*

- *IAbortable*
- *IAC500Diag*
- *IAC500DiagGet*
- *IACAlarmExtender*
- *IACAlarmExtender2*
- *IACAlarmExtender3*
- *IActionController*
- *IActionController2*
- *IActionProvider*
- *IAddressResolver*
- *IAlarm*
- *IAlarm2*
- *IAlarm3*
- *IAlarm4*
- *IAlarm5*
- *IAlarmClass*
- *IAlarmConfiguration7*
- *IAlarmGroup*
- *IAlarmGroup3*
- *IAlarmHandler*
- *IAlarmHandler2*
- *IAlarmHandler3*
- *IAlarmHandler4*
- *IAlarmHandler5*
- *IAlarmHandlerRemoteMonitor*
- *IAlarmManagerClient*
- *IAlarmManagerClient2*
- *IAlarmNotifiable*
- *IAlarmRemote*
- *IAlarmStateChangedEventListener*
- *IAlarmStateChangedListener*
- *IAlarmStateChangedListener2*
- *IAlarmStorageListener*
- *IAlarmStorageReaderConsumer*
- *IAlarmStorageReaderConsumer2*
- *IApplicationRectangleProvider*
- *IARPCallback*
- *IARPEthernetClient*
- *IArrayNotifiable*
- *IAsyncActionProvider*
- *IAsyncProperty*
- *IBackgroundTask*
- *IBACnetClient*
- *IBACnetEventConsumer*
- *IBACnetObjectBase*
- *IBACnetPersistence*
- *IBACnetPropertyConfiguration*
- *IBACnetServer*
- *IBACnetServerPlugin*
- *IBACnetServerPluginCallback*
- *IBACnetServerPluginHook*
- *IBACnetStaticObjectBase*
- *IBase*
- *IBehaviourModel*
- *IBoolElement*

- *IBranchTreeNode*
- *IBuffer*
- *IBufferPool*
- *IBufferPoolFactoryArgs*
- *IBus*
- *ICallOnDialogBlocks*
- *ICallOnVisuBlocks*
- *ICANopenEventHandler*
- *ICanOpenStack*
- *ICascadedDisposalProvider*
- *ICDSV3RequestBuilder*
- *ICDSV3RequestCallback*
- *ICertificateVerifier*
- *ICleanupActionProvider*
- *IClient*
- *IClientObjectInfo*
- *IClippingLayer*
- *ICmpEventCallback*
- *ICmpIoDrv*
- *ICmpIoDrvBusControl*
- *ICmpIoDrvBusControl2*
- *ICmpIoDrvCIPServices*
- *ICmpIoDrvLiveList*
- *ICmpIoDrvParameter*
- *ICmpIoDrvParameter2*
- *ICmpIoDrvPbSlaveActivation*
- *ICmpIoDrvProfibus*
- *ICmpIoDrvProfibusConfig*
- *ICmpIoDrvProfiNet*
- *ICollection*
- *ICompactTextListInfo2*
- *ICompleteSurroundingRectInfo*
- *IConfigurationProvider*
- *IConfigurationProvider2*
- *IConnection*
- *IContainerPaintSelf*
- *IContainsValue*
- *ICursor*
- *ICursor2*
- *ICursor3*
- *ICursorAsync*
- *ICustomAlarmToOpcUaMapping*
- *ICustomEventHandler*
- *ICyclicActionProvider*
- *ID*
- *ID_TO_ADDR*
- *IData*
- *IDataItemCompound*
- *IDataItemListInternal*
- *IDatasourcesActionRecordInternal*
- *IDatasourcesResourceEntryAllocator*
- *IDateTimeLanguageTextTarget*
- *IDateTimeProvider*
- *IDENT*
- *IDENT_TO_DWORD*

- *IDENT_TO_WORD*
- *IDevice*
- *IDevice2*
- *IDeviceCM579EtherCAT*
- *IDeviceCM582Profibus*
- *IDeviceCM589Profinet*
- *IDeviceCM592Profibus*
- *IDeviceCM598Can*
- *IDeviceSM560*
- *IDialogCloseListener*
- *IDialogCloseListenerWithTag*
- *IDialogManager2*
- *IDialogManager3*
- *IDialogManager4*
- *IDialogManager5*
- *IDialogManager6*
- *IDialogManager7*
- *IDialogManager8*
- *IDintElement*
- *IDintSet*
- *IDisposable*
- *IDoubleLinkedList*
- *IDrawSequentially*
- *IEC_BACNET_ABORT_REASON*
- *IEC_BACNET_ACCESS*
- *IEC_BACNET_ACCESS_AUTHENTICATION_FACTOR_DISABLE*
- *IEC_BACNET_ACCESS_CREDENTIAL_DISABLE*
- *IEC_BACNET_ACCESS_CREDENTIAL_DISABLE_REASON*
- *IEC_BACNET_ACCESS_EVENT*
- *IEC_BACNET_ACCESS_PASSBACK_MODE*
- *IEC_BACNET_ACCESS_RULE*
- *IEC_BACNET_ACCESS_RULE_LOCATION_SPECIFIER*
- *IEC_BACNET_ACCESS_RULE_RANGE_SPECIFIER*
- *IEC_BACNET_ACCESS_USER_TYPE*
- *IEC_BACNET_ACCESS_ZONE_OCCUPANCY_STATE*
- *IEC_BACNET_ACCUMULATOR_RECORD*
- *IEC_BACNET_ACCUMULATOR_STATUS*
- *IEC_BACNET_ACK_ALARM_INFO*
- *IEC_BACNET_ACK_FILTER*
- *IEC_BACNET_ACTION*
- *IEC_BACNET_ACTION_COMMAND*
- *IEC_BACNET_ACTION_LIST*
- *IEC_BACNET_ADDRESS*
- *IEC_BACNET_ADDRESS_BINDING*
- *IEC_BACNET_ADDRESS_TO_STRING*
- *IEC_BACNET_ALARM_INFO*
- *IEC_BACNET_ALARM_SUMMARY*
- *IEC_BACNET_APDU_PROPERTIES*
- *IEC_BACNET_ARRAY_INDEX*
- *IEC_BACNET_ASSIGNED_ACCESS_RIGHTS*
- *IEC_BACNET_AUTHENTICATION_FACTOR*
- *IEC_BACNET_AUTHENTICATION_FACTOR_FORMAT*
- *IEC_BACNET_AUTHENTICATION_FACTOR_TYPE*
- *IEC_BACNET_AUTHENTICATION_POLICY*
- *IEC_BACNET_AUTHENTICATION_POLICY_DATAINPUT*

- *IEC_BACNET_AUTHENTICATION_STATUS*
- *IEC_BACNET_AUTHORIZATION_MODE*
- *IEC_BACNET_BACKUP_STATE*
- *IEC_BACNET_BACKUPRESTORE_INFO*
- *IEC_BACNET_BINARY_PV*
- *IEC_BACNET_BIT_STRING*
- *IEC_BACNET_BOOLEAN*
- *IEC_BACNET_BUFFER*
- *IEC_BACNET_BVLL_BDT_ENTRY*
- *IEC_BACNET_BVLL_DELETE_FDT*
- *IEC_BACNET_BVLL_DISTRIBUTE_NPDU*
- *IEC_BACNET_BVLL_FDT_ENTRY*
- *IEC_BACNET_BVLL_FORWARDED_NPDU*
- *IEC_BACNET_BVLL_READ_BDT*
- *IEC_BACNET_BVLL_READ_FDT*
- *IEC_BACNET_BVLL_RESULT_CODE*
- *IEC_BACNET_BVLL_TYPE*
- *IEC_BACNET_BVLL_WRITE_BDT*
- *IEC_BACNET_BYTE*
- *IEC_BACNET_CALENDAR_ENTRY*
- *IEC_BACNET_CALENDAR_ENTRY_TYPE*
- *IEC_BACNET_CALLBACK_STATUS*
- *IEC_BACNET_CALLBACK_TYPE*
- *IEC_BACNET_CB_STATUS*
- *IEC_BACNET_CB_TYPE*
- *IEC_BACNET_CHANGE_LIST_INFO*
- *IEC_BACNET_CHANNEL_VALUE*
- *IEC_BACNET_CLI_INIT*
- *IEC_BACNET_CLIENT_COV*
- *IEC_BACNET_CLIENT_DEVICE_COMM_STATE*
- *IEC_BACNET_CLIENT_SUBSCRIBE_MODE*
- *IEC_BACNET_CONF_SERV_REQUEST*
- *IEC_BACNET_CONTROL_RANGECHK*
- *IEC_BACNET_CONTROL_REDUNDANT*
- *IEC_BACNET_CONTROL_STATS*
- *IEC_BACNET_COV_NOTIF_INFO*
- *IEC_BACNET_COV_SUBSCRIPTION*
- *IEC_BACNET_CREATE_OBJECT_INFO*
- *IEC_BACNET_CREATE_OBJECT_TYPE*
- *IEC_BACNET_CREDENTIAL_AUTHENTICATION_FACTOR*
- *IEC_BACNET_DAILY_SCHEDULE*
- *IEC_BACNET_DATA_TYPE*
- *IEC_BACNET_DATABASE_INFO*
- *IEC_BACNET_DATE*
- *IEC_BACNET_DATE_RANGE*
- *IEC_BACNET_DATE_TIME*
- *IEC_BACNET_DATE_TIME_TO_STRING*
- *IEC_BACNET_DATE_TO_STRING*
- *IEC_BACNET_DAY_OF_WEEK*
- *IEC_BACNET_DAY_OF_WEEK_BITS*
- *IEC_BACNET_DCC_INFO*
- *IEC_BACNET_DCC_VALUE*
- *IEC_BACNET_DDX_DDV_SIZE*
- *IEC_BACNET_DESTINATION*
- *IEC_BACNET_DEV_OBJ_PROP_REFERENCE*

- *IEC_BACNET_DEV_OBJ_PROP_VALUE*
- *IEC_BACNET_DEV_OBJ_REFERENCE*
- *IEC_BACNET_DEVICE_STATUS*
- *IEC_BACNET_DOOR_ALARM_STATE*
- *IEC_BACNET_DOOR_SECURED_STATUS*
- *IEC_BACNET_DOOR_STATUS*
- *IEC_BACNET_DOOR_VALUE*
- *IEC_BACNET_DOUBLE*
- *IEC_BACNET_DUMP_REPORT_FLAGS*
- *IEC_BACNET_DUMP_STATE*
- *IEC_BACNET_DWORD*
- *IEC_BACNET_ELEMENT_COUNT*
- *IEC_BACNET_EN_CONDITIONAL*
- *IEC_BACNET_EN_MANDATORY*
- *IEC_BACNET_EN_MANDATORY_TO_STRING*
- *IEC_BACNET_ENGINEERING_UNITS*
- *IEC_BACNET_ENROLLMENT_FILTER*
- *IEC_BACNET_ENROLLMENT_INFO*
- *IEC_BACNET_ENROLLMENT_SUMMARY*
- *IEC_BACNET_ENUM*
- *IEC_BACNET_EP_ACCESS_EVENT_PARAM*
- *IEC_BACNET_EP_BUF_READY_PARAM*
- *IEC_BACNET_EP_CHG_OF_BITS_PARAM*
- *IEC_BACNET_EP_CHG_OF_CHARSTRING_PARAM*
- *IEC_BACNET_EP_CHG_OF_STAT_FLG_PARAM*
- *IEC_BACNET_EP_CHG_OF_STATES_PARAM*
- *IEC_BACNET_EP_CMD_FAIL_PARAM*
- *IEC_BACNET_EP_COLS_PARAM*
- *IEC_BACNET_EP_COV_CRITERIA_TYPE*
- *IEC_BACNET_EP_COV_PARAM*
- *IEC_BACNET_EP_DBL_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_EP_E_PARAMETER*
- *IEC_BACNET_EP_EXT_PARAM*
- *IEC_BACNET_EP_FLOAT_LIMIT_PARAM*
- *IEC_BACNET_EP_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_EP_SIG_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_EP_UNS_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_EP_URANGE_PARAM*
- *IEC_BACNET_EPFP_E_PARAMETER*
- *IEC_BACNET_ERROR*
- *IEC_BACNET_ERROR_CLASS*
- *IEC_BACNET_ERROR_CODE*
- *IEC_BACNET_ERROR_TO_STRING*
- *IEC_BACNET_ERROR_TYPE*
- *IEC_BACNET_EVENT_INFO*
- *IEC_BACNET_EVENT_INFO_INFO*
- *IEC_BACNET_EVENT_LOG_RECORD*
- *IEC_BACNET_EVENT_LOG_RECORD_TYPE*
- *IEC_BACNET_EVENT_NOTIF_INFO*
- *IEC_BACNET_EVENT_NOTIFICATION_SUBSCRIPTION*
- *IEC_BACNET_EVENT_PARAMETER*
- *IEC_BACNET_EVENT_STATE*
- *IEC_BACNET_EVENT_SUMMARY*
- *IEC_BACNET_EVENT_TRANSITION_BITS*
- *IEC_BACNET_EVENT_TYPE*

- *IEC_BACNET_FAILURE_TYPE*
- *IEC_BACNET_FAULT_PARAM_TYPE*
- *IEC_BACNET_FAULT_PARAMETER*
- *IEC_BACNET_FILE_ACCESS_METHOD*
- *IEC_BACNET_FILE_ACCESS_TYPE*
- *IEC_BACNET_FP_CHARSTRING_PARAM*
- *IEC_BACNET_FP_COLS_PARAM*
- *IEC_BACNET_FP_E_PARAMETER*
- *IEC_BACNET_FP_EXT_PARAM*
- *IEC_BACNET_FP_STAT_FLG_PARAM*
- *IEC_BACNET_FP_STATES_PARAM*
- *IEC_BACNET_FRAME_PARAM*
- *IEC_BACNET_FRAME_PART*
- *IEC_BACNET_FRAME_PART_TYPE*
- *IEC_BACNET_GROUP_CHANNEL_VALUE*
- *IEC_BACNET_HANDLE*
- *IEC_BACNET_I_AM_INFO*
- *IEC_BACNET_I_HAVE_INFO*
- *IEC_BACNET_INST_NUMBER*
- *IEC_BACNET_IP_BBMD_ENTRY*
- *IEC_BACNET_KEY_ID_ALGORITHM*
- *IEC_BACNET_KEY_ID_NUMBER*
- *IEC_BACNET_KEY_IDENTIFIER*
- *IEC_BACNET_KEY_REVISION*
- *IEC_BACNET_LIFE_SAFETY_INFO*
- *IEC_BACNET_LIFE_SAFETY_MODE*
- *IEC_BACNET_LIFE_SAFETY_OPERATION*
- *IEC_BACNET_LIFE_SAFETY_STATE*
- *IEC_BACNET_LIGHTING_COMMAND*
- *IEC_BACNET_LIGHTING_IN_PROGRESS*
- *IEC_BACNET_LIGHTING_OPERATION*
- *IEC_BACNET_LIGHTING_TRANSITION*
- *IEC_BACNET_LIMIT_ENABLE*
- *IEC_BACNET_LOCK_STATUS*
- *IEC_BACNET_LOG_RECORD*
- *IEC_BACNET_LOG_RECORD_M_DATA*
- *IEC_BACNET_LOG_RECORD_M_DATA_TYPE*
- *IEC_BACNET_LOG_RECORD_M_ENTRY*
- *IEC_BACNET_LOG_RECORD_MULTIPLE*
- *IEC_BACNET_LOG_RECORD_MULTIPLE_TYPE*
- *IEC_BACNET_LOG_RECORD_TYPE*
- *IEC_BACNET_LOG_STATUS_BITS*
- *IEC_BACNET_LOGGING_TYPE*
- *IEC_BACNET_MAC_ETH*
- *IEC_BACNET_MAC_IP*
- *IEC_BACNET_MAC_IP_TO_STRING*
- *IEC_BACNET_MAC_LON*
- *IEC_BACNET_MAC_MSTP*
- *IEC_BACNET_MAC_PTP*
- *IEC_BACNET_MAINTENANCE*
- *IEC_BACNET_MESSAGE_CLASS*
- *IEC_BACNET_MESSAGE_CLASS_TYPE*
- *IEC_BACNET_MESSAGE_PRIORITY*
- *IEC_BACNET_MONTH*
- *IEC_BACNET_NETWORK_MANAGEMENT_MESSAGE*

- *IEC_BACNET_NETWORK_MANAGEMENT_MSG_TYPE*
- *IEC_BACNET_NETWORK_SECURITY_POLICY*
- *IEC_BACNET_NMM_BVLL*
- *IEC_BACNET_NMM_CONFIRM_TO_NETWORK*
- *IEC_BACNET_NMM_ESTABLISH_TO_NETWORK*
- *IEC_BACNET_NMM_EVENT*
- *IEC_BACNET_NMM_EVENT_REASON*
- *IEC_BACNET_NMM_IAM_ROUTER_TO_NETWORK*
- *IEC_BACNET_NMM_ICOULDBE_ROUTER_TO_NETWORK*
- *IEC_BACNET_NMM_INDICATE_TO_NETWORK*
- *IEC_BACNET_NMM_INIT_ROUTINGTABLE*
- *IEC_BACNET_NMM_MESSAGE_ID*
- *IEC_BACNET_NMM_NETWORK*
- *IEC_BACNET_NMM_NETWORK_NUMBER_IS*
- *IEC_BACNET_NMM_REJECT_REASON*
- *IEC_BACNET_NMM_REJECT_TO_NETWORK*
- *IEC_BACNET_NMM_ROUTER_AVAIL_TO_NETWORK*
- *IEC_BACNET_NMM_ROUTER_BUSY_TO_NETWORK*
- *IEC_BACNET_NMM_ROUTINGTABLE_ACK*
- *IEC_BACNET_NMM_ROUTINGTABLE_ENTRY*
- *IEC_BACNET_NMM_TIMESTAMP*
- *IEC_BACNET_NMM_TYPE*
- *IEC_BACNET_NODE_TYPE*
- *IEC_BACNET_NOTIFICATION_PARAMETERS*
- *IEC_BACNET_NOTIFY_TYPE*
- *IEC_BACNET_NP_ACCESS_EVENT_PARAM*
- *IEC_BACNET_NP_BUF_READY_PARAM*
- *IEC_BACNET_NP_BUF_READY_PARAM2*
- *IEC_BACNET_NP_CHG_OF_BITS_PARAM*
- *IEC_BACNET_NP_CHG_OF_CHARSTRING_PARAM*
- *IEC_BACNET_NP_CHG_OF_RELIABTY_PARAM*
- *IEC_BACNET_NP_CHG_OF_STAT_FLG_PARAM*
- *IEC_BACNET_NP_CHG_OF_STATE_PARAM*
- *IEC_BACNET_NP_CMD_FAIL_PARAM*
- *IEC_BACNET_NP_COLS_PARAM*
- *IEC_BACNET_NP_COMPLEX_PARAM*
- *IEC_BACNET_NP_COV_PARAM*
- *IEC_BACNET_NP_COV_TYPE*
- *IEC_BACNET_NP_DBL_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_NP_E_PARAMETER*
- *IEC_BACNET_NP_EXT_PARAM*
- *IEC_BACNET_NP_FLOAT_LIMIT_PARAM*
- *IEC_BACNET_NP_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_NP_SIG_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_NP_UNS_OUT_OF_RANGE_PARAM*
- *IEC_BACNET_NP_URANGE_PARAM*
- *IEC_BACNET_NPDU_REJECT_REASON*
- *IEC_BACNET_NPDU_TYPE*
- *IEC_BACNET_OBJ_PROP_REFERENCE*
- *IEC_BACNET_OBJECT_ID*
- *IEC_BACNET_OBJECT_ID_TO_STRING*
- *IEC_BACNET_OBJECT_SPECIFIER*
- *IEC_BACNET_OBJECT_TYPE*
- *IEC_BACNET_OBJECT_TYPES_BITS*
- *IEC_BACNET_OCTET_STRING*

- *IEC_BACNET_OPTIONAL_STRING*
- *IEC_BACNET_OS_TIME_PROVIDER*
- *IEC_BACNET_OS_TIME_PROVIDER_TIME*
- *IEC_BACNET_PDU_TYPE*
- *IEC_BACNET_PERIOD_TYPE*
- *IEC_BACNET_POLARITY*
- *IEC_BACNET_PORT_PERMISSION*
- *IEC_BACNET_PRESCALE*
- *IEC_BACNET_PRIORITY_ARRAY_ITEM*
- *IEC_BACNET_PRIORITY_LEVEL*
- *IEC_BACNET_PRIVATE_TRANSFER_INFO*
- *IEC_BACNET_PROCESS_ID_SELECTION*
- *IEC_BACNET_PROGRAM_ERROR*
- *IEC_BACNET_PROGRAM_REQUEST*
- *IEC_BACNET_PROGRAM_STATE*
- *IEC_BACNET_PROP_STATES_TYPE*
- *IEC_BACNET_PROPERTY_ACCESS_RESULT*
- *IEC_BACNET_PROPERTY_CONTENTS*
- *IEC_BACNET_PROPERTY_DESCRIPTION*
- *IEC_BACNET_PROPERTY_DESCRIPTION_LIST*
- *IEC_BACNET_PROPERTY_ID*
- *IEC_BACNET_PROPERTY_INSTANCE*
- *IEC_BACNET_PROPERTY_REFERENCE*
- *IEC_BACNET_PROPERTY_STATES*
- *IEC_BACNET_PROPERTY_VALUE*
- *IEC_BACNET_RANGE_FLAGS*
- *IEC_BACNET_RANGE_TYPE*
- *IEC_BACNET_RAW_ASN1_VALUE*
- *IEC_BACNET_READ_ACCESS_RESULT*
- *IEC_BACNET_READ_ACCESS_SPEC*
- *IEC_BACNET_READ_FILE_INFO*
- *IEC_BACNET_READ_FILE_RANGE*
- *IEC_BACNET_READ_FILE_RESULT*
- *IEC_BACNET_READ_INFO*
- *IEC_BACNET_READ_INFO_TO_STRING*
- *IEC_BACNET_READ_LIST*
- *IEC_BACNET_READ_MUL_INFO*
- *IEC_BACNET_READ_RANGE_INFO*
- *IEC_BACNET_READ_RANGE_RANGE*
- *IEC_BACNET_READ_RANGE_RESULT*
- *IEC_BACNET_READ_RAW_RESULT_LIST*
- *IEC_BACNET_READ_RESULT_ITEM*
- *IEC_BACNET_READ_RESULT_LIST*
- *IEC_BACNET_REAL*
- *IEC_BACNET_RECIPIENT*
- *IEC_BACNET_RECIPIENT_PROCESS*
- *IEC_BACNET_RECIPIENT_TYPE*
- *IEC_BACNET_REINIT_DEV_INFO*
- *IEC_BACNET_REINIT_TYPE*
- *IEC_BACNET_REJECT_REASON*
- *IEC_BACNET_RELATION_TYPE*
- *IEC_BACNET_RELIABILITY*
- *IEC_BACNET_REMOTE_DEVICE_CAPS*
- *IEC_BACNET_RESTART_REASON*
- *IEC_BACNET_SCALE*

- *IEC_BACNET_SECURITY_KEY_SET*
- *IEC_BACNET_SECURITY_LEVEL*
- *IEC_BACNET_SECURITY_POLICY*
- *IEC_BACNET_SECURITY_RESPONSE_CODE*
- *IEC_BACNET_SEGMENTATION*
- *IEC_BACNET_SELECTION_LOGIC*
- *IEC_BACNET_SERVICES_BITS*
- *IEC_BACNET_SESSION_KEY*
- *IEC_BACNET_SETPOINT_REFERENCE*
- *IEC_BACNET_SHED_LEVEL*
- *IEC_BACNET_SHED_LEVEL_TYPE*
- *IEC_BACNET_SHED_STATE*
- *IEC_BACNET_SIGNED*
- *IEC_BACNET_SILENCED_STATE*
- *IEC_BACNET_SPECIAL_EVENT*
- *IEC_BACNET_SRVR_INIT*
- *IEC_BACNET_STACK_CONTROL*
- *IEC_BACNET_STACK_CONTROL_TYPE*
- *IEC_BACNET_STACK_DATALINK*
- *IEC_BACNET_STACK_DATALINK_CONFIG*
- *IEC_BACNET_STACK_DATALINK_TYPE*
- *IEC_BACNET_STACK_ETHERNET_DATALINK*
- *IEC_BACNET_STACK_IERROR*
- *IEC_BACNET_STACK_IERROR_TYPE*
- *IEC_BACNET_STACK_IP_DATALINK*
- *IEC_BACNET_STACK_LONTALK_DATALINK*
- *IEC_BACNET_STACK_MSTP_DATALINK*
- *IEC_BACNET_STACK_PTP_DATALINK*
- *IEC_BACNET_STACK_VIRTUAL_DATALINK*
- *IEC_BACNET_STATE_FILTER*
- *IEC_BACNET_STATUS*
- *IEC_BACNET_STATUS_FLAG_BITS*
- *IEC_BACNET_STRING*
- *IEC_BACNET_STRING_TABLE_ENTRY*
- *IEC_BACNET_STRING_TABLE_INFO*
- *IEC_BACNET_STRING_TYPE*
- *IEC_BACNET_SUBSCRIBE_COV_INFO*
- *IEC_BACNET_SUBSCRIBE_COVP_INFO*
- *IEC_BACNET_TAG*
- *IEC_BACNET_TEMPLATE_DEVICE*
- *IEC_BACNET_TEMPLATE_OBJECT*
- *IEC_BACNET_TEXT_MESSAGE_INFO*
- *IEC_BACNET_TIME*
- *IEC_BACNET_TIME_STAMP*
- *IEC_BACNET_TIME_STAMP_TYPE*
- *IEC_BACNET_TIME_SYNC_INFO*
- *IEC_BACNET_TIME_TO_STRING*
- *IEC_BACNET_TIME_VALUE*
- *IEC_BACNET_UNCONF_SERV_REQUEST*
- *IEC_BACNET_UNSIGNED*
- *IEC_BACNET_VT_CLASS*
- *IEC_BACNET_VT_SESSION*
- *IEC_BACNET_WEEK_AND_DAY*
- *IEC_BACNET_WEEK_OF_MONTH*
- *IEC_BACNET_WHO_HAS_IND_OBJ_SPEC_TYPE*

- *IEC_BACNET_WHO_HAS_INFO*
- *IEC_BACNET_WHO_HAS_PARAM*
- *IEC_BACNET_WHO_HAS_TYPE*
- *IEC_BACNET_WHO_IS_INFO*
- *IEC_BACNET_WORD*
- *IEC_BACNET_WRITE_FILE_DATA*
- *IEC_BACNET_WRITE_FILE_INFO*
- *IEC_BACNET_WRITE_FILE_RESULT*
- *IEC_BACNET_WRITE_GROUP_INFO*
- *IEC_BACNET_WRITE_INFO*
- *IEC_BACNET_WRITE_INFO_TO_STRING*
- *IEC_BACNET_WRITE_ITEM*
- *IEC_BACNET_WRITE_LIST*
- *IEC_BACNET_WRITE_MUL_INFO*
- *IEC_BACNET_WRITE_STATUS*
- *IEC_CYCLE_STRUCT*
- *IEC_STATE*
- *IEC60870_5_104_Connection*
- *IEC60870_BACKGROUND_SCAN*
- *IEC60870_DISABLE*
- *IEC60870_DoubleCommand*
- *IEC60870_DoublePointInformation*
- *IEC60870_GET_ADDRESS*
- *IEC60870_IntegratedTotal*
- *IEC60870_MeasuredValue*
- *IEC60870_REC_C_DC*
- *IEC60870_REC_C_SC*
- *IEC60870_REC_C_SE*
- *IEC60870_REC_C_TS_NA_1*
- *IEC60870_REC_M_DP*
- *IEC60870_REC_M_IT*
- *IEC60870_REC_M_ME*
- *IEC60870_REC_M_ME_1*
- *IEC60870_REC_M_SP*
- *IEC60870_REC_P_ME*
- *IEC60870_SEND_C_CI_NA_1*
- *IEC60870_SEND_C_CI_NA_1_2*
- *IEC60870_SEND_C_CS_NA_1*
- *IEC60870_SEND_C_CS_NA_1_2*
- *IEC60870_SEND_C_DC*
- *IEC60870_SEND_C_IC_NA_1*
- *IEC60870_SEND_C_IC_NA_1_2*
- *IEC60870_SEND_C_RD_NA_1*
- *IEC60870_SEND_C_RP_NA_1*
- *IEC60870_SEND_C_RP_NA_1_2*
- *IEC60870_SEND_C_SC*
- *IEC60870_SEND_C_SE*
- *IEC60870_SEND_C_TS_NA_1_ACT*
- *IEC60870_SEND_C_TS_NA_1_ACTCON*
- *IEC60870_SEND_DISABLE*
- *IEC60870_SEND_M_DP*
- *IEC60870_SEND_M_DP_ET*
- *IEC60870_SEND_M_EI_NA_1*
- *IEC60870_SEND_M_IT*
- *IEC60870_SEND_M_IT_1*

- *IEC60870_SEND_M_IT_1_ET*
- *IEC60870_SEND_M_IT_16*
- *IEC60870_SEND_M_IT_16_ET*
- *IEC60870_SEND_M_ME*
- *IEC60870_SEND_M_ME_1*
- *IEC60870_SEND_M_ME_1_ET*
- *IEC60870_SEND_M_ME_16*
- *IEC60870_SEND_M_ME_16_ET*
- *IEC60870_SEND_M_SP*
- *IEC60870_SEND_M_SP_1_ET*
- *IEC60870_SEND_M_SP_16*
- *IEC60870_SEND_M_SP_16_ET*
- *IEC60870_SEND_P_ME*
- *IEC60870_SetPoint*
- *IEC60870_SingleCommand*
- *IEC60870_SinglePointInformation*
- *IEC60870_STATE*
- *IEC60870_TIME*
- *IEC60870Commands*
- *IEC60870Disable*
- *IEC60870DisableSend*
- *IEC60870GetConfigAddress*
- *IEC60870GetConfigValues*
- *IEC60870GetConnectionStatistics*
- *IEC60870GetPinData*
- *IEC60870GetStatesOfPinParam*
- *IEC60870GetStatesOfPins*
- *IEC60870GetTestInformation*
- *IEC60870SendCommand*
- *IEC60870SendPinData*
- *IEC60870SetParameterValues*
- *IEC60870SetPinData*
- *IEC60870StartScan*
- *IEC61850_ArrayBits_SwapLeft*
- *IEC61850_ASN1_Decoder*
- *IEC61850_ASN1_DECODING*
- *IEC61850_ASN1_Decoding_Data*
- *IEC61850_ASN1_EncodingBlock*
- *IEC61850_ASN1_EncodingSize*
- *IEC61850_ASN1_EncodingSpecific*
- *IEC61850_ASN1_EncodingStruct*
- *IEC61850_ASN1_GetNextTag*
- *IEC61850_ASN1_NewDecoder*
- *IEC61850_ByteBits_SwapLeft*
- *IEC61850_ByteBits_SwapRight*
- *IEC61850_CDC_ACD*
- *IEC61850_CDC_ACT*
- *IEC61850_CDC_ALM*
- *IEC61850_CDC_APC*
- *IEC61850_CDC_ASG*
- *IEC61850_CDC_ASS*
- *IEC61850_CDC_BCR*
- *IEC61850_CDC_BRCB*
- *IEC61850_CDC_BSC*
- *IEC61850_CDC_CMD*

- *IEC61850_CDC_CMV*
- *IEC61850_CDC_CSD*
- *IEC61850_CDC_CTE*
- *IEC61850_CDC_CURVE*
- *IEC61850_CDC_DEL*
- *IEC61850_CDC_DPC*
- *IEC61850_CDC_DPL*
- *IEC61850_CDC_DPS*
- *IEC61850_CDC_GoCB*
- *IEC61850_CDC_HDEL*
- *IEC61850_CDC_HMV*
- *IEC61850_CDC_HWYE*
- *IEC61850_CDC_INC*
- *IEC61850_CDC_ING*
- *IEC61850_CDC_INS*
- *IEC61850_CDC_ISC*
- *IEC61850_CDC_LPL*
- *IEC61850_CDC_MV*
- *IEC61850_CDC_ORG*
- *IEC61850_CDC_SAV*
- *IEC61850_CDC_SEC*
- *IEC61850_CDC_SEQ*
- *IEC61850_CDC_SPC*
- *IEC61850_CDC_SPG*
- *IEC61850_CDC_SPS*
- *IEC61850_CDC_SPV*
- *IEC61850_CDC_STV*
- *IEC61850_CDC_TMS*
- *IEC61850_CDC_URCB*
- *IEC61850_CDC_WDPL*
- *IEC61850_CDC_WYE*
- *IEC61850_Check_HexString*
- *IEC61850_CheckBufferIx*
- *IEC61850_CheckByteOrder*
- *IEC61850_CheckClients*
- *IEC61850_CheckDataPoint*
- *IEC61850_CheckDoubleDP*
- *IEC61850_CheckEntryID*
- *IEC61850_CheckEnumRange*
- *IEC61850_CheckTrgOp*
- *IEC61850_CLIENT_ACCEPT*
- *IEC61850_ClientConnectionFB*
- *IEC61850_CONCAT3*
- *IEC61850_CONCAT4*
- *IEC61850_CONCAT5*
- *IEC61850_CONCAT6*
- *IEC61850_CpyAndSwap*
- *IEC61850_CreateBasicNames*
- *IEC61850_DatasetFB*
- *IEC61850_DateTime*
- *IEC61850_DecodeNull*
- *IEC61850_DeleteDataSet*
- *IEC61850_DWORD_TO_HEXSTRING*
- *IEC61850_Encoding_Array_Count*
- *IEC61850_Encoding_Array_Struct*

- *IEC61850_Encoding_Component*
- *IEC61850_Encoding_Component_Struct*
- *IEC61850_Encoding_ComponentSingle*
- *IEC61850_Encoding_DirectoryNames*
- *IEC61850_Encoding_ListOfData*
- *IEC61850_Encoding_ListOfData_Struct*
- *IEC61850_Encoding_ListOfVariable*
- *IEC61850_Encoding_Value*
- *IEC61850_ENUM_ASN1_TAGS*
- *IEC61850_ENUM_ATTR_NAMES*
- *IEC61850_ENUM_DA_ALM_STATE*
- *IEC61850_ENUM_DA_ANGID*
- *IEC61850_ENUM_DA_ANGIDCMV*
- *IEC61850_ENUM_DA_ANGLEREFERENCEKIND*
- *IEC61850_ENUM_DA_ASS_STVAL*
- *IEC61850_ENUM_DA_BEH*
- *IEC61850_ENUM_DA_CBOPCAP*
- *IEC61850_ENUM_DA_CMDQUAL*
- *IEC61850_ENUM_DA_CONTROLOUTPUTKIND*
- *IEC61850_ENUM_DA_CTE_HISRS*
- *IEC61850_ENUM_DA_CTE_RSPER*
- *IEC61850_ENUM_DA_CTLMODELKIND*
- *IEC61850_ENUM_DA_CTLMODELS*
- *IEC61850_ENUM_DA_CURVECHARKIND*
- *IEC61850_ENUM_DA_DAWEEKDAYKIND*
- *IEC61850_ENUM_DA_DBPOS*
- *IEC61850_ENUM_DA_DIR*
- *IEC61850_ENUM_DA_DIRMOD*
- *IEC61850_ENUM_DA_ENUMERATED*
- *IEC61850_ENUM_DA_FAILMOD*
- *IEC61850_ENUM_DA_FANCTL*
- *IEC61850_ENUM_DA_FAULTDIRECTIONKIND*
- *IEC61850_ENUM_DA_GNST*
- *IEC61850_ENUM_DA_HEALTH*
- *IEC61850_ENUM_DA_HVID*
- *IEC61850_ENUM_DA_HVREFERENCEKIND*
- *IEC61850_ENUM_DA_LEVMOD*
- *IEC61850_ENUM_DA_LIVDEAMOD*
- *IEC61850_ENUM_DA_MOD*
- *IEC61850_ENUM_DA_MONTHKIND*
- *IEC61850_ENUM_DA_MULTIPLIER*
- *IEC61850_ENUM_DA_MULTIPLIERKIND*
- *IEC61850_ENUM_DA_OCCURRENCEKIND*
- *IEC61850_ENUM_DA_OPMOD*
- *IEC61850_ENUM_DA_ORCAT*
- *IEC61850_ENUM_DA_ORIGINATORCATEGORYKIND*
- *IEC61850_ENUM_DA_PERIODKIND*
- *IEC61850_ENUM_DA_PHASEANGLEREFERENCEKIND*
- *IEC61850_ENUM_DA_PHASEFAULTDIRECTIONKIND*
- *IEC61850_ENUM_DA_PHASEREFERENCEKIND*
- *IEC61850_ENUM_DA_PHSID*
- *IEC61850_ENUM_DA_POLQTY*
- *IEC61850_ENUM_DA_POWCAP*
- *IEC61850_ENUM_DA_RANGE*
- *IEC61850_ENUM_DA_RANGEKIND*

- *IEC61850_ENUM_DA_RETRMOD*
- *IEC61850_ENUM_DA_RSTMOD*
- *IEC61850_ENUM_DA_RVAMOD*
- *IEC61850_ENUM_DA_SBOCLASSES*
- *IEC61850_ENUM_DA_SBOCLASSKIND*
- *IEC61850_ENUM_DA_SCHTYP*
- *IEC61850_ENUM_DA_SEQT*
- *IEC61850_ENUM_DA_SEQUENCEKIND*
- *IEC61850_ENUM_DA_SETCHARACT*
- *IEC61850_ENUM_DA_SEV*
- *IEC61850_ENUM_DA_SEVERITYKIND*
- *IEC61850_ENUM_DA_SHOPCAP*
- *IEC61850_ENUM_DA_SIUNIT*
- *IEC61850_ENUM_DA_SIUNITKIND*
- *IEC61850_ENUM_DA_SPV_CHAPERRS*
- *IEC61850_ENUM_DA_SPV_SPACS*
- *IEC61850_ENUM_DA_SWOPCAP*
- *IEC61850_ENUM_DA_SWTYP*
- *IEC61850_ENUM_DA_TCMD*
- *IEC61850_ENUM_DA_TMS_HISRS*
- *IEC61850_ENUM_DA_TMS_RSPER*
- *IEC61850_ENUM_DA_TRGMOD*
- *IEC61850_ENUM_DA_TRMOD*
- *IEC61850_ENUM_DA_TYPRSCRV*
- *IEC61850_ENUM_DA_UNBLKMOD*
- *IEC61850_ENUM_DA_WEIMOD*
- *IEC61850_ENUM_ELEMENTTYP*
- *IEC61850_ENUM_FC*
- *IEC61850_ENUM_MMS_CONFIRMED_REQ_PDU*
- *IEC61850_ENUM_MMS_CONFIRMED_RESP_PDU*
- *IEC61850_ENUM_MMS_DataType*
- *IEC61850_ENUM_MMS_OBJECTCLASS*
- *IEC61850_ENUM_MMS_PDU*
- *IEC61850_ENUM_QUALITY*
- *IEC61850_ENUM_SERVICES*
- *IEC61850_ENUM_TRGOPT*
- *IEC61850_EthernetAdapter*
- *IEC61850_GetDatapoint*
- *IEC61850_GetDataPointLen*
- *IEC61850_GetDatapointRef*
- *IEC61850_GetDefinition*
- *IEC61850_GetDirectory*
- *IEC61850_GetDirectory_All*
- *IEC61850_GetFC*
- *IEC61850_GetReportLen*
- *IEC61850_GetURCBDataLen*
- *IEC61850_GetValue*
- *IEC61850_GetValues_All*
- *IEC61850_Goose_ASN1_Decoder*
- *IEC61850_GOOSE_MReq*
- *IEC61850_GooseDecodeData*
- *IEC61850_HEXSTRING_TO_DWORD*
- *IEC61850_HistDataBuffer_In*
- *IEC61850_HistDataBufferFB*
- *IEC61850_Init_BReportBlock*

- *IEC61850_Init_DataPoints*
- *IEC61850_Init_GoCB*
- *IEC61850_Init_UBReportBlock*
- *IEC61850_InitDSLastValPtr*
- *IEC61850_INT_TO_STRING*
- *IEC61850_MMS_Data_InterpreterFB*
- *IEC61850_MMS_ErrorPDU*
- *IEC61850_MMS_InterpreterFB*
- *IEC61850_MMSGetBlockLen*
- *IEC61850_ReadDWord*
- *IEC61850_ReadISOHeader*
- *IEC61850_ReadString*
- *IEC61850_ReadWord*
- *IEC61850_SetDatasetVal*
- *IEC61850_SetDSError*
- *IEC61850_SetISOEntry*
- *IEC61850_SetISOLen*
- *IEC61850_SetReportValue*
- *IEC61850_SetStructIndex*
- *IEC61850_SetTrgOpt*
- *IEC61850_SetValue*
- *IEC61850_SimpleClock*
- *IEC61850_STR_TO_BYTE*
- *IEC61850_String_Split*
- *IEC61850_SWAP_2_BYTE*
- *IEC61850_SWAP_3_BYTE*
- *IEC61850_SWAP_4_BYTE*
- *IEC61850_SysMemCpy*
- *IEC61850_TimeStampR*
- *IEC61850_Version*
- *IEC61850_WordBits_SwapLeft*
- *IEC61850_WordBits_SwapRight*
- *IEC61850ServerFB*
- *IecTaskCreate*
- *IecTaskCreate2*
- *IecTaskDelete2*
- *IecTaskDisableScheduling*
- *IecTaskDisableWatchdog*
- *IecTaskDisableWatchdog2*
- *IecTaskEnableScheduling*
- *IecTaskEnableWatchdog*
- *IecTaskEnableWatchdog2*
- *IecTaskGetCurrent*
- *IecTaskGetDesc*
- *IecTaskGetFirst*
- *IecTaskGetInfo3*
- *IecTaskGetNext*
- *IecTaskGetProfiling*
- *IecTaskReload*
- *IecTaskResetStatistics*
- *IecVarAccBrowseCallback*
- *IecVarAccBrowseDirection*
- *IecVarAccBrowseDown2*
- *IecVarAccBrowseGetNext2*
- *IecVarAccBrowseRecursive*

- *IecVarAccBrowseUp2*
- *IecVarAccess*
- *IecVarAccessUaInformationModelMetaData*
- *IecVarAccExitVarInfo*
- *IecVarAccGetFirstInterface*
- *IecVarAccGetFirstInterface2*
- *IecVarAccGetNextInterface*
- *IecVarAccGetNextInterface2*
- *IecVarAccGetNode4*
- *IecVarAccGetNodeFullPath4*
- *IecVarAccGetNodeName4*
- *IecVarAccGetSymbolSetMask*
- *IecVarAccInitVarInfo*
- *IecVarAccInitVarInfo2*
- *IecVarAccInvalidateNode*
- *IecVarAccNodeInfoAddBrowseInfo*
- *IecVarAccNodeInfoAddReference*
- *IecVarAccNodeInfoGetBrowseInfo*
- *IecVarAccNodeInfoGetReference*
- *IecVarAccNodeInfoRemoveBrowseInfo*
- *IecVarAccNodeInfoRemoveReference*
- *IecVarAccRegisterInstance*
- *IecVarAccRegisterInstance2*
- *IecVarAccRegisterInstance3*
- *IecVarAccRegisterInstanceBase*
- *IecVarAccRegisterInstanceBase2*
- *IecVarAccSetSymbolconfigCrc*
- *IecVarAccSymbolSetDescription*
- *IecVarAccUnregisterInstance*
- *IecVarAccUpdateSymbolSets*
- *IEdgeTriggered*
- *IElement*
- *IEthernet*
- *IETrig*
- *IETrigA*
- *IETrigATI*
- *IETrigATITo*
- *IETrigATo*
- *IETrigTI*
- *IETrigTITo*
- *IETrigTo*
- *IExitActionProvider*
- *IExpandSubNodeAdapterSingleRelease*
- *IExternalUserDatabaseProvider*
- *IExternalUserDatabaseProvider2*
- *IFactory*
- *IFBCommand*
- *IFrame*
- *IFrameElement2*
- *IFrameElement3*
- *IFrameManager*
- *IFrameManager2*
- *IFrameManagerBase*
- *IGeneralCommand*
- *IGestureEventHandler*

- *IGestureEventHandler2*
- *IGestureEventHandler3*
- *IGridProvider*
- *IHasContinuousBehaviour*
- *iIEC61850_LogicalDevice*
- *IIecVarAccess*
- *IIecVarAccess10*
- *IIecVarAccess11*
- *IIecVarAccess12*
- *IIecVarAccess13*
- *IIecVarAccess14*
- *IIecVarAccess15*
- *IIecVarAccess2*
- *IIecVarAccess3*
- *IIecVarAccess4*
- *IIecVarAccess5*
- *IIecVarAccess6*
- *IIecVarAccess7*
- *IIecVarAccess8*
- *IIecVarAccess9*
- *IIecVarAccessOpcUaMetaData*
- *IInputOnElementEventHandler*
- *IInputRectangle*
- *IInputRectangleMgr*
- *IInputRectangleProvider*
- *IInstance*
- *IIntElement*
- *IIoDrvEIPAcylicServices*
- *IIPAddress*
- *IIPAddressSet*
- *IIParameterData*
- *IIPv4Address*
- *IIterator*
- *IKeyEventHandler*
- *ILayeredVisualElement*
- *ILayerManager*
- *ILCon*
- *ILConC*
- *ILConTI*
- *ILConTIC*
- *ILConTITo*
- *ILConTo*
- *ILeafTreeNode*
- *ILevelControlled*
- *ILinkedListIterator*
- *ILintElement*
- *IList*
- *IList2*
- *IListIterator*
- *ILocalAssigner*
- *ILocalizedDateTimeNames*
- *ILogger*
- *ILRealToStringFormatter*
- *IMap*
- *IMap2*

- *IMemberIndex*
- *IMemoryManager*
- *IModuleAlarming*
- *IMouseEventHandler*
- *IMultitouchElement*
- *INADDR*
- *INetworkInterface*
- *INFO*
- *InfoValues*
- *InitializeBACnetBitString*
- *InitializeBACnetBoolean*
- *InitializeBACnetDate*
- *InitializeBACnetDateRange*
- *InitializeBACnetDateTime*
- *InitializeBACnetDateTimeUnspecified*
- *InitializeBACnetDevObjPropReference*
- *InitializeBACnetSetpointReference*
- *InitializeBACnetString*
- *InitializeBACnetTime*
- *InitializeBACnetTimeStamp*
- *InitializeEmptyPropertyInstance*
- *InitializePropertyInstance*
- *INode*
- *INode_TO_IBus*
- *INode_TO_IDevice*
- *INode_TO_IDevice2*
- *INode_TO_IStack*
- *INodeId*
- *INodeName*
- *InputDataSave*
- *INSERT*
- *Inspect_Heap*
- *InstanceBase*
- *InstanceData*
- *InstancePathBuildingBranchNode*
- *InstancePathBuildingNode*
- *InstancePathNodeFinder*
- *INT_TO_BCD*
- *INT_TO_SIGNED*
- *INT64*
- *INT64_TO_DT*
- *INT64_TO_ISO8601*
- *INT64_TO_LOCALTIME*
- *INT64_TO_LTIME*
- *INT64_TO_REAL8*
- *INT64_TO_TIME*
- *INT64_TO_UTC*
- *INTEGRAL*
- *Integral*
- *IntElement*
- *IntElementFactory*
- *InterfaceEthernetStatistic*
- *InterfaceVersion*
- *InternalConnectionState*
- *InternalState*

- *INullElement*
- *InverseMemCopy*
- *IO_SYSTEM_TYPE*
- *IObjectDictionary*
- *IOBus_Download*
- *IOBus_GetBusInfo*
- *IOBus_GetBusStatistcis*
- *IOBus_GetDownloadState*
- *IOBus_GetHotplugOK*
- *IOBus_GetIODriverVersion*
- *IOBus_GetModState*
- *IOBus_GetModuleInfo*
- *IOBus_GetModuleLinkStatistics*
- *IOBus_GetModuleStatistics*
- *IOBus_GetModuleVersion*
- *IOBus_GetPlugged*
- *IOBus_GetProductionData*
- *IOBus_GetRun*
- *IOBUS_INFO*
- *IOBUS_LINKSTATISTICS*
- *IOBUS_MOD_STATE*
- *IOBUS_MODUL_STATE*
- *IOBUS_MODULINFO*
- *IOBUS_PARA_STATE*
- *IOBUS_PRODDATA*
- *IOBUS_STATISTICS*
- *IOBus_SwitchLinkStatistics*
- *IOBUS_TU_STATE*
- *IOBUS_VERSIONINFO*
- *IoConfigChannelMap*
- *IoConfigConnector*
- *IoConfigConnectorMap*
- *IoConfigParameter*
- *IoConfigTaskMap*
- *IoCopyIn*
- *IoCopyOut*
- *IODCallback*
- *IODObject*
- *IoDrvAL1030*
- *IoDrvAL1x3x*
- *IoDrvAnalogBase*
- *IoDrvBase*
- *IoDrvCIFX*
- *IoDrvCIFXEthernetIP*
- *IoDrvCIFXEthernetIP_Diag*
- *IoDrvCIFXProfibus*
- *IoDrvCIFXProfibusDevice*
- *IoDrvCIFXProfibusDeviceDiag*
- *IoDrvCIFXProfibusDiag*
- *IoDrvCIFXProfinetDevice*
- *IoDrvCIFXProfinetDeviceDiag*
- *IoDrvCM579EtherCAT*
- *IoDrvCM579EtherCATDiag*
- *IoDrvCM579Profinet*
- *IoDrvCM579ProfinetDiag*

- *IoDrvCM582Profibus*
- *IoDrvCM582ProfibusDiag*
- *IoDrvCM589Profinet*
- *IoDrvCM589ProfinetDiag*
- *IoDrvCM592Profibus*
- *IoDrvCM592ProfibusDiag*
- *IoDrvCM598*
- *IoDrvCM598Diag*
- *IoDrvCpuModuleDiag*
- *IoDrvDigitalOptionBoardBase*
- *IoDrvEL6224*
- *IoDrvEL6631*
- *IoDrvEL6631_0010*
- *IoDrvEL6631_0010_Diag*
- *IoDrvEL6631Diag*
- *IoDrvEL6731*
- *IoDrvEL6731_0010*
- *IoDrvEL6731_0010_Diag*
- *IoDrvEL6731Diag*
- *IoDrvEtherCAT*
- *IoDrvEthercat_Diag*
- *IoDrvEthernet*
- *IoDrvEthernetAC500*
- *IoDrvEthernetAC500Diag*
- *IoDrvEthernetDiag*
- *IoDrvEtherNetIP*
- *IoDrvEtherNetIP_diag*
- *IoDrvEtherNetIPAdapter*
- *IoDrvEtherNetIPAdapter_Diag*
- *IoDrvGpioSysfs*
- *IoDrvGpioSysfsDiag*
- *IoDrvHilscher*
- *IoDrvHilscherProfibus*
- *IoDrvHilscherProfibusWrapper*
- *IoDrvInfo*
- *IoDrvIoBusModuleDiag*
- *IoDrvJ1939Diag*
- *IoDrvKNX*
- *IoDrvKNXDiag*
- *IoDrvModbusComPort*
- *IoDrvModbusComPort_Diag*
- *IoDrvModbusSerialSlave*
- *IoDrvModbusTCP*
- *IoDrvModbusTCP_Diag*
- *IoDrvModbusTCPSlave*
- *IoDrvOnboardIO*
- *IoDrvOnboardIODiag*
- *IoDrvS500ModuleDiag*
- *IoDrvSafetySp*
- *IoDrvSercos3*
- *IoDrvSercos3_Diag*
- *IoDrvSM560*
- *IoDrvSM560Diag*
- *IoDrvTA5101*
- *IoDrvTA5101Diag*

- *IoDrvTA5105*
- *IoDrvTA5105Diag*
- *IoDrvTA5110*
- *IoDrvTA5110Diag*
- *IoDrvTA5120*
- *IoDrvTA5120Diag*
- *IoDrvTA5122*
- *IoDrvTA5122Diag*
- *IoDrvTA5123*
- *IoDrvTA5123Diag*
- *IoDrvTA5126*
- *IoDrvTA5126Diag*
- *IODSubObject*
- *IOL_AdditionalCode*
- *IOL_AdjustableSwitchingSensor*
- *IOL_AdSS_Function*
- *IOL_AdSS_Status*
- *IOL_AdSS_TeachFunction*
- *IOL_AdSS_TeachMode*
- *IOL_CALL*
- *IOL_DataStorage*
- *IOL_DiagEntry*
- *IOL_Error*
- *IOL_ErrorCode*
- *IOL_Event*
- *IOL_EventCode*
- *IOL_EventCode_Device*
- *IOL_EventCode_Port*
- *IOL_EventQualifier*
- *IOL_EventQualifier_Instance*
- *IOL_EventQualifier_Mode*
- *IOL_EventQualifier_Source*
- *IOL_EventQualifier_Type*
- *IOL_FieldbusStatus*
- *IOL_GetEvent_ChannelDiagnosis*
- *IOL_GetEvent_UDINT*
- *IOL_IdentificationAndDiagnosis*
- *IOL_IdentificationAndDiagnosis_Function*
- *IOL_IdentificationObjects*
- *IOL_Index*
- *IOL_IOLM_Info*
- *IOL_IOLM_InfoRecord*
- *IOL_IQ_Behavior*
- *IOL_MasterIdent*
- *IOL_MasterIdent_Features_1*
- *IOL_MasterType*
- *IOL_MeasurementDataChannel*
- *IOL_PN_PortControl*
- *IOL_PortConfigList*
- *IOL_PortConfiguration*
- *IOL_PortConfigurationRecord*
- *IOL_PortError*
- *IOL_PortMode*
- *IOL_PortQualityInfo*
- *IOL_PortStatus*

- *IOL_PortStatusInfo*
- *IOL_PortStatusList*
- *IOL_PortStatusRecord*
- *IOL_PortType*
- *IOL_PQI*
- *IOL_ProfileIdentifier*
- *IOL_TransmissionRate*
- *IOL_ValidationBackup*
- *IoLinkService*
- *IOLINKSERVICEHEADER*
- *IOLinkServices*
- *IoMgrConfigGetConnector*
- *IoMgrConfigGetConnectorByDriver*
- *IoMgrConfigGetConnectorList*
- *IoMgrConfigGetDriver*
- *IoMgrConfigGetFirstChild*
- *IoMgrConfigGetFirstConnector*
- *IoMgrConfigGetFirstParameter*
- *IoMgrConfigGetNextChild*
- *IoMgrConfigGetNextConnector*
- *IoMgrConfigGetNextParameter*
- *IoMgrConfigGetParameter*
- *IoMgrConfigGetParameterValueByte*
- *IoMgrConfigGetParameterValueDword*
- *IoMgrConfigGetParameterValuePointer*
- *IoMgrConfigGetParameterValueWord*
- *IoMgrConfigResetDiagnosis*
- *IoMgrConfigSetDiagnosis*
- *IoMgrCopyInputBE*
- *IoMgrCopyInputLE*
- *IoMgrCopyOutputBE*
- *IoMgrCopyOutputLE*
- *IoMgrGetBusCycleType*
- *IoMgrGetConfigApplication*
- *IoMgrGetDriverProperties*
- *IoMgrGetFirstDriverInstance*
- *IoMgrGetModuleDiagnosis*
- *IoMgrGetNextDriverInstance*
- *IoMgrIdentify*
- *IoMgrLockEnter*
- *IoMgrLockLeave*
- *IoMgrReadInputs*
- *IoMgrReadParameter*
- *IoMgrReconfigure*
- *IoMgrRegisterConfigApplication*
- *IoMgrRegisterInstance2*
- *IoMgrScanModules*
- *IoMgrSetDriverProperties*
- *IoMgrSetDriverProperty*
- *IoMgrStartBusCycle*
- *IoMgrStartBusCycle2*
- *IoMgrUnregisterConfigApplication*
- *IoMgrUnregisterInstance*
- *IoMgrUpdateConfiguration*
- *IoMgrUpdateConfiguration2*

- *IoMgrUpdateMapping*
- *IoMgrUpdateMapping2*
- *IoMgrWatchdogTrigger*
- *IoMgrWriteOutputs*
- *IoMgrWriteParameter*
- *IOMODULEDESC*
- *IOnlineChangeSafeLinkedListElement*
- *IOPCUAClientConnectionCallback*
- *IOPCUAClientDataAccessCallback*
- *IOPCUAClientDiscoveryCallback*
- *IOPCUAClientMethodCallback*
- *IOPCUAClientMonitoredItemCallback*
- *IOPCUAClientSubscriptionCallback*
- *IOPCUAClientViewCallback*
- *IOpcUaDataTypeMetaData*
- *IOpcUaInstanceMetaData*
- *IOptionalMultitouchElement*
- *IOxStatus*
- *IP_ADDR*
- *IP_ADR_DWORD_TO_STRING*
- *IP_ADR_STRING_TO_DWORD*
- *IPAADialog*
- *IPacket*
- *IPacketPool*
- *IPacketQueue*
- *IPADDRESS*
- *IPAddressSet*
- *IPaintAfterAll*
- *IPaintAfterAll2*
- *IPaintAfterAllRectangleProvider*
- *IPaintAfterAllSelection*
- *IPaintSelectionInElement*
- *IPARRAY_TO_INADDR*
- *IPARRAY_TO_IPSTRING*
- *IPARRAY_TO_UDINT*
- *iParServer*
- *iParServerError*
- *IPBSlaveDiag*
- *IPeer*
- *IPersistantRecipeListSupportsAdd*
- *IPParameterValue*
- *IProvidesBitOffset*
- *IProvidesDifferentRemoteName*
- *IProvidesRootInfo*
- *IProvidesTabOrder*
- *IProxyMonitor*
- *IPseudoNode*
- *IPSTRING_TO_UDINT*
- *IPStringAndIntElement*
- *IPStringElement*
- *IPv4Address*
- *IQueryInterfaceElement*
- *IQueue*
- *IQueue2*
- *IQueueableNode*

- *IRdtProt*
- *IRdtProtClient*
- *IRdtProtServer*
- *IReadableSharedArea*
- *IRecipeCheckOnStart*
- *IRecipeDefinition2*
- *IReconfigureProvider*
- *IRectangleListManager*
- *IRectangleListManager2*
- *IRectangleListManager3*
- *IRectangleListManager4*
- *IRectangleProvider*
- *IRequest*
- *IRequestNoSyncReleaseDuringShutdown*
- *IRequestResult*
- *IRequiresInitMeasureString*
- *IResetActionProvider*
- *IResolveCallbackHandler*
- *IRow*
- *IRow2*
- *IRow3*
- *IRowAsync*
- *IRowBase*
- *IRowIdIterator*
- *IRowPlanchet*
- *IRowPlanchetAsync*
- *IRPCCLClient*
- *IRPCCLClientCallback*
- *IRPCProvider*
- *IRPCProviderCallback*
- *IRtsServiceHandler*
- *IRtsServiceHandler2*
- *IS_MULTICAST_GROUP*
- *Is29BitIdMessage*
- *IsAcceptedLeafNode*
- *IsAddressInArea*
- *IsaInterrupt*
- *ISampleActionProvider*
- *ISavepoint*
- *ISavepointAsync*
- *IsBACnetBACnetDateTimeUnspecified*
- *IsBACnetDateTimeUnspecified*
- *IsBACnetObjectAMEVCreatable*
- *IsBACnetPropertyAMEVASBWritable*
- *IsBroadcast*
- *IScrollValueProvider*
- *ISDOHandler*
- *ISearchCallbacks*
- *ISegment*
- *ISegmentPool*
- *ISelectableInside*
- *ISelectionManager*
- *IServer*
- *IServerCommand*
- *IServiceReader*

- *IServiceWriter*
- *IsFullRune*
- *IsHandleValid*
- *ISharedArea*
- *ISharedAreaObserver*
- *ISharedAreaRef*
- *ISharedAreaUtilities*
- *ISharedPointer*
- *ISharedQueue*
- *ISimpleList*
- *ISimpleTree*
- *IsInvalidMemoryAddress*
- *IsLeapYear*
- *IsLegalUTF8*
- *IsLibReleased*
- *IsLRealNaN*
- *IsLRealNegInfinity*
- *IsLRealNumber*
- *IsLRealPosInfinity*
- *ISO8073_FB*
- *ISO8327_FB*
- *ISO8327_ReadHeader*
- *ISO8601*
- *ISO8601_TO_DT*
- *ISO8601_TO_LTIME*
- *ISO8601_TO_TIME*
- *ISO8650_FB*
- *ISO8823_FB*
- *ISOLayer_FB*
- *ISortedList*
- *ISortedList2*
- *IsP2P*
- *ISpecialEventHandler*
- *IsRealNaN*
- *IsRealNegInfinity*
- *IsRealNumber*
- *IsRealPosInfinity*
- *IsRTRMessage*
- *IsRuneStart*
- *IsSendingActive*
- *IStack*
- *IStack2*
- *IStartActionProvider*
- *IStorage*
- *IStorage2*
- *IStorageAsync*
- *IsTransmitMessage*
- *IStream*
- *IStringElement*
- *ISupportsRealDrawing*
- *IsValid*
- *IsValidRune*
- *ISysInt*
- *ITable*
- *ITable2*

- *ITable3*
- *ITable4*
- *ITableAsync*
- *ITargetVisuLight*
- *ITaskFinishedCallback*
- *ITCPProcessor*
- *ITextListInfo*
- *ITextListWrapper*
- *ITimeElement*
- *ITimeLimited*
- *ITimeOutConstraint*
- *ITimingControlled*
- *ITimingController*
- *ITLSContext*
- *ITransaction*
- *ITransactionAsync*
- *ITransformationImplProvider*
- *ITree*
- *ITreeNode*
- *ITreeWalker*
- *ITrendRootPageManager2*
- *ITrendStorageAccessReadOperator*
- *ITrendStorageAccessReadOperator2*
- *ITrendStorageReaderConsumer*
- *ITrendStorageWriterListener*
- *ITSNContext*
- *ITypedElement*
- *ITypeDesc*
- *ITypeDesc2*
- *ITypeDesc3*
- *ITypeDesc4*
- *ITypeDescExecutable*
- *ITypeDescSubrange*
- *ITypeDescWithAttributes*
- *ITypeDescWithBaseType*
- *ITypeDescWithReferenceType*
- *ITypedList*
- *ITypedTree*
- *IUdintElement*
- *IUDPProcessor*
- *IUintElement*
- *IUlintElement*
- *IUseDataContextSubNodes*
- *IUserMgmtEventHandler*
- *IValueChangedListener*
- *IVariableInformation*
- *IVariableInformation2*
- *IVariableInformation3*
- *IVariableInformation4*
- *IVariableInformation5*
- *IVerifyCertCallback*
- *IVisualElementLayer*
- *IVisualElementOfflineScaling*
- *IVisualElementProvidesSubElements*
- *IVisualElementWithoutBlobInit*

- *IVisualisationAccessRights*
- *IVisualizationClient*
- *IVisualizationClientFilter*
- *IVisualizationClientIteration*
- *IVisuManager*
- *IVisuManager2*
- *IVisuManagerBase*
- *IVisuStreamFileNameInfo*
- *IVisuStreamHandler*
- *IVisuStreamReader*
- *IVisuStreamSetFileName*
- *IVisuStreamWriter*
- *IVisuUserEventManager*
- *IVisuUserManagement*
- *IVisuUserManagement2*
- *IVisuUserMgmtCyclicCall*
- *IWORKER*
- *IWriteableSharedArea*
- *IWStringElement*
- *IXYChartDataProvider*
- *IXYChartDataProvider2*
- *IXYChartDataProvider3*
- *IXYChartDataProviderAxis*
- *IXYChartDataProviderCurve*
- *IXYChartFont*
- *IXYChartGenericVariable*
- *IXYChartGenericVariable2*
- *IXYChartStringApproxMeasurer*
- *IXYChartVisuStructLevelLine*
- *J1939ECUBase*
- *J1939LocalECU*
- *J1939LocalECUDiag*
- *J1939RemoteECU*
- *J1939RemoteECUDiag*
- *Jitter_Distribution*
- *JOB_STATE*
- *JobAbort*
- *JobClass*
- *JobClose*
- *JobExecute*
- *JobGetId*
- *JobGetParams*
- *JobGetState*
- *JobOpen*
- *JobOpenEx*
- *JobReset*
- *JobSetState*
- *JoinDateTime*
- *JSON_ARR_REF*
- *JSON_OBJ_REF*
- *JsonAddArray*
- *JsonAddBool*
- *JsonAddInt*
- *JsonAddObject*
- *JsonAddReal*

- *JsonAddString*
- *JsonArrayAddArray*
- *JsonArrayAddBool*
- *JsonArrayAddInt*
- *JsonArrayAddObject*
- *JsonArrayAddReal*
- *JsonArrayAddString*
- *JsonArrayGetArray*
- *JsonArrayGetBool*
- *JsonArrayGetInt*
- *JsonArrayGetObject*
- *JsonArrayGetReal*
- *JsonArrayGetString*
- *JsonArrayRemoveEntry*
- *JsonCreateArray*
- *JsonCreateObject*
- *JsonFreeArray*
- *JsonFreeObject*
- *JsonGetArray*
- *JsonGetBool*
- *JsonGetInt*
- *JsonGetObject*
- *JsonGetReal*
- *JsonGetString*
- *JsonParseArrayFromString*
- *JsonParseObjectFromString*
- *JsonRemoveEntry*
- *JsonSerializeArray*
- *JsonSerializeObject*
- *KeyValuePair*
- *LAMP_FLASH*
- *LAMP_INFO*
- *LAMP_STATUS*
- *LatchVariable*
- *LCon*
- *LConC*
- *LConTI*
- *LConTIC*
- *LConTITo*
- *LConTo*
- *LCTD*
- *LCTU*
- *LCTUD*
- *LeafTreeNode*
- *LeafTreeNodeOpcUA*
- *LeafTreeNodeTypeMember*
- *LeafTreeNodeTypeMemberOpcUA*
- *LED_ID*
- *LEFT*
- *LegacyRTSVisuStructEvent2*
- *LEN*
- *LicenseFunctions*
- *LIMITALARM*
- *LimitAlarm_DINT*
- *LimitAlarm_LREAL*

- *LimitDeviceObjectPropertyReferencesToCertainTypes*
- *LIN_TRAFO*
- *LINE_3D*
- *LinearMemoryManager*
- *LinearTrafo*
- *LinkedList*
- *LinkedListElementBase*
- *LinkedListFactory*
- *LinkedListIterator*
- *LinkState_Link*
- *LINT_TO_SIGNED*
- *LintElement*
- *LintElementFactory*
- *List*
- *ListBase*
- *Listener*
- *ListFactory*
- *ListIterator*
- *ListNewClient*
- *ListNewFrame*
- *ListNewLogin*
- *ListNewPage*
- *ListOfDevices*
- *ListRemoveClient*
- *ListValueChanged*
- *ListVisuClient*
- *ListVisuClientDwnSL*
- *LMMBlock*
- *LocalDateTime*
- *LOG_ENTRY*
- *LogAdd*
- *LogAdd2*
- *LogClose*
- *LogComponent*
- *LogCreate*
- *LogDelete*
- *LogGeneric_Input*
- *LogGeneric_Output*
- *LOGGER_MODE*
- *LoggingHelper*
- *LoggingInit*
- *LoggingOptions*
- *LogHandling*
- *Logiec60870_Input*
- *Logiec60870_Output*
- *LogManager*
- *LogMessage*
- *LogObjectBaseFileHandleTableEntry*
- *LogObjectsBase*
- *LogOpen*
- *LogOptions*
- *LogToDevice*
- *LostMessages*
- *LowByte*
- *LowWord*

- *LREAL_TO_FLOAT*
- *LRealToHexStr*
- *LRealToStr*
- *LTIME_TO_DURATION*
- *LTIME_TO_INT64*
- *LTIME_TO_ISO8601*
- *LTIME_TO_REAL8*
- *LTOF*
- *LTON*
- *LTP*
- *LTrig*
- *LWORD_TO_HANDLE*
- *LWORD_TO_PVOID*
- *MAC_ADDRESS_COMPARE*
- *MAKE_EVENTID*
- *MakeNormed3D*
- *MapErrorCode*
- *MapErrorCodeFailedAsConnLost*
- *MapIECResult*
- *MapNetBaseServiceError*
- *MapOpcUaStatus*
- *MappingDesc_ArrayArbitrary*
- *MappingDesc_ArraySubRange*
- *MAUType*
- *MB_AccessTypes*
- *MB_ErrorCodes*
- *MB_MasterParameter*
- *MB_Medium*
- *MB_Parity*
- *MB_PortParameter*
- *MB_SlaveParameter*
- *MB_Transmission*
- *MB_TriggerType*
- *MBFunctionCode*
- *MD5*
- *MD5_FF*
- *MD5_GG*
- *MD5_HH*
- *MD5_II*
- *MD5_Transform*
- *MeasureFrequence*
- *MeasuringPoint*
- *MemBuffer*
- *MemCmp*
- *MemCopy*
- *MemCopySwap*
- *MemCpy*
- *MemFill*
- *MemForceSwap*
- *MemMove*
- *MemoryBarrier*
- *MemoryManager*
- *MemSet*
- *MESSAGE*
- *MessageBox_Struct*

- *METRICS*
- *MID*
- *Mid2*
- *MILLISECOND*
- *MINUTE*
- *MMAP_PROT*
- *MMAPS_FLAGS*
- *ModbusChannel*
- *ModbusCommand*
- *ModbusRequest*
- *ModbusRequest2*
- *ModbusSerialDeviceDiag*
- *ModbusSerialSlaveBase*
- *ModbusServer*
- *ModbusSlaveComPort*
- *ModbusSlaveComPort_Diag*
- *ModbusTCPComSettings*
- *ModbusTCPComState*
- *ModbusTCPDeviceDiag*
- *ModbusTCPSlave*
- *ModbusTCPSlave_Diag*
- *ModbusTCPSlaveBase*
- *ModbusTCPSlaveUnit*
- *ModbusTCPSlaveUnit_Diag*
- *MODE*
- *ModRtuGenDevDataType*
- *ModRtuGenDevDataTypeInternal*
- *ModRtuMast*
- *ModRtuMastTypeInternal*
- *ModRtuRead*
- *ModRtuReadWrite23*
- *ModRtuToken*
- *ModRtuTokenType*
- *ModRtuWrite*
- *ModTcpConfig*
- *ModTcpInfo*
- *ModTcpMast*
- *ModTcpMast2*
- *ModTcpServOnOff*
- *Module*
- *Module_Diag*
- *ModuleAlarmInfo*
- *ModuleCall*
- *ModuleEvent*
- *MODULESTATE*
- *ModuleState*
- *MonitorDBStatus*
- *MonitoredItem*
- *MonitoredItemState*
- *MonitoredReadRequest*
- *MonitoredReadRequestState*
- *MonitorFilterByDateTime*
- *MonitorFilterByLatch*
- *Monitoring2ByteCode*
- *Monitoring2ByteCodeUnion*

- *MonitoringService*
- *MonitoringServiceHelper*
- *MonitorPopulateFilterCriteria*
- *MONTH*
- *MoveAbsolute*
- *MoveAbsoluteData*
- *MoveRelative*
- *MQTT_CONNECTION*
- *MQTT_MESSAGE*
- *MQTT_QOS*
- *MqttConnectWithCertBuffer*
- *MqttConnectWithCertFile*
- *MqttConnectWithCertStore*
- *MqttDisconnect*
- *MqttGetReceivedPacket*
- *MqttPing*
- *MqttPublish*
- *MqttSubscribe*
- *MqttUnsubscribe*
- *MsgAddRef*
- *MsgClass*
- *MsgClone*
- *MsgGetData*
- *MsgReceive*
- *MsgRelease*
- *MsgReleaseEx*
- *MsgSend*
- *MSK_ECM_IF_EXT_SYNC_INFO_FLAGS*
- *NamedTreeNode*
- *NamespaceIdFixer*
- *NamespaceNodeFlags*
- *NamespaceTable*
- *NCAPDUFaultStatus*
- *NestingPathEntry*
- *NestingPathInformation*
- *NET_INFO*
- *NetClientCloseChannel*
- *NetClientOpenChannel*
- *NetClientOpenChannelResult*
- *NetClientSend*
- *NetDiagnosis*
- *NetVarDataItem_Udp*
- *NetVarManager_Udp_FB*
- *NetVarOD_Service_Udp*
- *NetVarPDO_Rx_Udp*
- *NetVarPDO_Tx_Udp*
- *NetVarTelegramm_Udp*
- *NetVarTlgHeader_Udp*
- *NetVarUDPDiagStruct*
- *NetVarUDPError*
- *NETX_DEV_DIAG*
- *NETX_SYSTEM_CHANNEL*
- *NETX_UDINT_TO_STRINGHEX*
- *NetxEcatInit*
- *NetxEcatIsCompatible*

- *NetxEcatJobBusScanStart*
- *NetxEcatJobBusScanStop*
- *NetxEcatJobGetDevStatistics*
- *NetxEcatJobGetExtSyncInfo*
- *NetxEcatJobGetInfo*
- *NetxEcatJobGetMasterCPULoad*
- *NetxEcatJobGetMasterDcInfo*
- *NetxEcatJobGetMasterFrameLossCount*
- *NetxEcatJobGetMasterMemInfo*
- *NetxEcatJobGetMasterThresholdCount*
- *NetxEcatJobGetMasterTimingInfo*
- *NetxEcatJobGetSlaveDcInfo*
- *NetxEcatJobGetSlaveDiag*
- *NetxEcatJobGetSlaveMDPModules*
- *NetxEcatJobPrmSanityCheck*
- *NetxEcatJobPrmSanityCheckSlave*
- *NetxEcatJobReadRegister*
- *NetxEcatJobReadSlaveLostLinkCnt*
- *NetxEcatJobReadSlaveRxErrorCnt*
- *NetxEcatJobReadSlaveVersion*
- *NetxEcatJobSdoRead*
- *NetxEcatJobSdoWrite*
- *NetxEcatJobSetSlaveState*
- *NetxEcatJobSetState*
- *NetxEcatJobSoeRead*
- *NetxEcatJobSoeWrite*
- *NetxEcatJobStart*
- *NetxEcatJobStop*
- *NetxEcatJobWriteRegister*
- *NMT*
- *NMT_ERROR_BEHAVIOUR*
- *NodeFlags*
- *NODEID*
- *NodeId*
- *NodeIdArray*
- *NodeInformation*
- *NodeMapper*
- *NOP*
- *Norm3D*
- *NSC_AddrComponent*
- *NSC_CompleteNodeInfo*
- *NSC_NodeAddress*
- *NSC_NodeInfoExt*
- *NSC_NodeInfoInt*
- *NSClientClose*
- *NSClientGeneralResolveCallback*
- *NSClientOpen*
- *NSClientResolveAll*
- *NSClientSearchNodeFlags*
- *NSClientSysMemAllocator*
- *NSClientTaskBase*
- *NSClientTaskResolveAllNodes*
- *NSClientTaskSearchForSpecificNode*
- *NSClientUtil_DumpAddress*
- *NSClientUtil_DumpAddressHelp*

- *NSClientUtil_DumpCallback*
- *NSClientUtil_DumpNodeInfo*
- *NSClientUtil_DumpStartSearchNodeParams*
- *NSClientUtil_Log1*
- *NSClientUtil_Log2*
- *NSClientUtil_Log3*
- *NSClientWrapper*
- *NtpSourceInfoData*
- *NtpSourceMode*
- *NtpSourceState*
- *NullElement*
- *OBIO_PTO_Motion_Parameter*
- *OBIO_PTOMotionKernel*
- *OBIO_PWM_Motion_Parameter*
- *OBIO_PWMMotionKernel*
- *OBIOBasicPoint2Point*
- *OBIOEncoderCounter*
- *OBIOForwardCounter*
- *OBIOFreqOut*
- *OBIOInterruptInfo*
- *OBIOInterruptPara*
- *OBIOLimitSwitch*
- *OBIOMotionPTO*
- *OBIOMotionPwm*
- *OBIOPulseTrainOutput*
- *OBIOPwm*
- *OBIOSineSquarePoint2Point*
- *ObjectIterator*
- *ObjectPersistence*
- *OffsetPoints*
- *OLM_OnlineLicenseManager*
- *OPCAClientCredentials_UserPassword*
- *OpcDateTimeToDT*
- *OpcUa_ActivateSessionRequest*
- *OpcUa_ActivateSessionResponse*
- *OpcUa_AddNodesItem*
- *OpcUa_AddNodesRequest*
- *OpcUa_AddNodesResponse*
- *OpcUa_AddNodesResult*
- *OpcUa_AddReferencesItem*
- *OpcUa_AddReferencesRequest*
- *OpcUa_AddReferencesResponse*
- *OpcUa_AggregateConfiguration*
- *OpcUa_AggregateFilter*
- *OpcUa_AggregateFilterResult*
- *OpcUa_Annotation*
- *OpcUa_AnonymousIdentityToken*
- *OpcUa_ApplicationDescription*
- *OpcUa_ApplicationType*
- *OpcUa_Argument*
- *OpcUa_ArrayType*
- *OpcUa_AttributeOperand*
- *OpcUa_Attributes*
- *OpcUa_AxisInformation*
- *OpcUa_AxisScaleEnumeration*

- *OpcUa_Boolean*
- *OpcUa_BrowseDescription*
- *OpcUa_BrowseDirection*
- *OpcUa_BrowseNextRequest*
- *OpcUa_BrowseNextResponse*
- *OpcUa_BrowsePath*
- *OpcUa_BrowsePathResult*
- *OpcUa_BrowsePathTarget*
- *OpcUa_BrowseRequest*
- *OpcUa_BrowseResponse*
- *OpcUa_BrowseResult*
- *OpcUa_BrowseResultMask*
- *OpcUa_BuildInfo*
- *OpcUa_BuiltInType*
- *OpcUa_Byte*
- *OpcUa_ByteString*
- *OpcUa_CallMethodRequest*
- *OpcUa_CallMethodResult*
- *OpcUa_CallRequest*
- *OpcUa_CallResponse*
- *OpcUa_CancelRequest*
- *OpcUa_CancelResponse*
- *OpcUa_ChannelSecurityToken*
- *OpcUa_CharA*
- *OpcUa_CloseSecureChannelRequest*
- *OpcUa_CloseSecureChannelResponse*
- *OpcUa_CloseSessionRequest*
- *OpcUa_CloseSessionResponse*
- *OpcUa_ComplexNumberType*
- *OpcUa_ContentFilter*
- *OpcUa_ContentFilterElement*
- *OpcUa_ContentFilterElementResult*
- *OpcUa_ContentFilterResult*
- *OpcUa_CreateMonitoredItemsRequest*
- *OpcUa_CreateMonitoredItemsResponse*
- *OpcUa_CreateSessionRequest*
- *OpcUa_CreateSessionResponse*
- *OpcUa_CreateSubscriptionRequest*
- *OpcUa_CreateSubscriptionResponse*
- *OpcUa_DataChangeFilter*
- *OpcUa_DataChangeNotification*
- *OpcUa_DataChangeTrigger*
- *OpcUa_DataTypeAttributes*
- *OpcUa_DataValue*
- *OpcUa_DateTime*
- *OpcUa_Decoder*
- *OpcUa_DeleteAtTimeDetails*
- *OpcUa_DeleteEventDetails*
- *OpcUa_DeleteMonitoredItemsRequest*
- *OpcUa_DeleteMonitoredItemsResponse*
- *OpcUa_DeleteNodesItem*
- *OpcUa_DeleteNodesRequest*
- *OpcUa_DeleteNodesResponse*
- *OpcUa_DeleteRawModifiedDetails*
- *OpcUa_DeleteReferencesItem*

- *OpcUa_DeleteReferencesRequest*
- *OpcUa_DeleteReferencesResponse*
- *OpcUa_DeleteSubscriptionsRequest*
- *OpcUa_DeleteSubscriptionsResponse*
- *OpcUa_DiagnosticInfo*
- *OpcUa_Double*
- *OpcUa_DoubleComplexNumberType*
- *OpcUa_ElementOperand*
- *OpcUa_EncodeableObjectBody*
- *OpcUa_EncodeableType*
- *OpcUa_Encoder*
- *OpcUa_EndpointConfiguration*
- *OpcUa_EndpointDescription*
- *OpcUa_EndpointUrlListDataType*
- *OpcUa_EnumDefinition*
- *OpcUa_EnumField*
- *OpcUa_EnumValueType*
- *OpcUa_EUInformation*
- *OpcUa_EventFieldList*
- *OpcUa_EventFilter*
- *OpcUa_EventFilterResult*
- *OpcUa_EventNotificationList*
- *OpcUa_ExpandedNodeId*
- *OpcUa_ExtensionObject*
- *OpcUa_ExtensionObject_Body*
- *OpcUa_ExtensionObjectEncoding*
- *OpcUa_FilterOperator*
- *OpcUa_FindServersOnNetworkRequest*
- *OpcUa_FindServersOnNetworkResponse*
- *OpcUa_FindServersRequest*
- *OpcUa_FindServersResponse*
- *OpcUa_Float*
- *OpcUa_GenericAttributes*
- *OpcUa_GenericAttributeValue*
- *OpcUa_GetEndpointsRequest*
- *OpcUa_GetEndpointsResponse*
- *OpcUa_Guid*
- *OPcUa_Handle*
- *OpcUa_HistoryData*
- *OpcUa_HistoryEvent*
- *OpcUa_HistoryEventFieldList*
- *OpcUa_HistoryModifiedData*
- *OpcUa_HistoryReadRequest*
- *OpcUa_HistoryReadResponse*
- *OpcUa_HistoryReadResult*
- *OpcUa_HistoryReadValueId*
- *OpcUa_HistoryUpdateDetails*
- *OpcUa_HistoryUpdateRequest*
- *OpcUa_HistoryUpdateResponse*
- *OpcUa_HistoryUpdateResult*
- *OpcUa_HistoryUpdateType*
- *OpcUa_IdentifierType*
- *OpcUa_Int*
- *OpcUa_Int16*
- *OpcUa_Int32*

- *OpcUa_Int64*
- *OpcUa_IssuedIdentityToken*
- *OpcUa_LiteralOperand*
- *OpcUa_LocalizedText*
- *OpcUa_MdnsDiscoveryConfiguration*
- *OpcUa_MessageSecurityMode*
- *OpcUa_MethodAttributes*
- *OpcUa_ModelChangeStructureDataType*
- *OpcUa_ModificationInfo*
- *OpcUa_ModifyMonitoredItemsRequest*
- *OpcUa_ModifyMonitoredItemsResponse*
- *OpcUa_ModifySubscriptionRequest*
- *OpcUa_ModifySubscriptionResponse*
- *OpcUa_MonitoredItemCreateRequest*
- *OpcUa_MonitoredItemCreateResult*
- *OpcUa_MonitoredItemModifyRequest*
- *OpcUa_MonitoredItemModifyResult*
- *OpcUa_MonitoredItemNotification*
- *OpcUa_MonitoringMode*
- *OpcUa_MonitoringParameters*
- *OpcUa_NetworkGroupDataType*
- *OpcUa_NodeAttributes*
- *OpcUa_NodeClass*
- *OpcUa_NodeId*
- *OpcUa_NodeId_anon*
- *OpcUa_NodeIds*
- *OpcUa_NodeReference*
- *OpcUa_NodeTypeDescription*
- *OpcUa_NotificationMessage*
- *OpcUa_ObjectAttributes*
- *OpcUa_ObjectTypeAttributes*
- *OpcUa_OpenSecureChannelRequest*
- *OpcUa_OpenSecureChannelResponse*
- *OpcUa_OptionSet*
- *OpcUa_ParsingResult*
- *OpcUa_PerformUpdateType*
- *OpcUa_ProgramDiagnostic2DataType*
- *OpcUa_ProgramDiagnosticDataType*
- *OpcUa_PublishRequest*
- *OpcUa_PublishResponse*
- *OpcUa_QualifiedName*
- *OpcUa_QueryDataDescription*
- *OpcUa_QueryDataSet*
- *OpcUa_QueryFirstRequest*
- *OpcUa_QueryFirstResponse*
- *OpcUa_QueryNextRequest*
- *OpcUa_QueryNextResponse*
- *OpcUa_Range*
- *OpcUa_ReadAtTimeDetails*
- *OpcUa_ReadEventDetails*
- *OpcUa_ReadProcessedDetails*
- *OpcUa_ReadRawModifiedDetails*
- *OpcUa_ReadRequest*
- *OpcUa_ReadResponse*
- *OpcUa_ReadValueId*

- *OpcUa_RedundantServerDataType*
- *OpcUa_ReferenceDescription*
- *OpcUa_ReferenceTypeAttributes*
- *OpcUa_RegisteredServer*
- *OpcUa_RegisterNodesRequest*
- *OpcUa_RegisterNodesResponse*
- *OpcUa_RegisterServer2Request*
- *OpcUa_RegisterServer2Response*
- *OpcUa_RegisterServerRequest*
- *OpcUa_RegisterServerResponse*
- *OpcUa_RelativePath*
- *OpcUa_RelativePathElement*
- *OpcUa_RepublishRequest*
- *OpcUa_RepublishResponse*
- *OpcUa_RequestHeader*
- *OpcUa_ResponseHeader*
- *OpcUa_RolePermissionType*
- *OpcUa_SamplingIntervalDiagnosticsDataType*
- *OpcUa_SByte*
- *OpcUa_SecurityTokenRequestType*
- *OpcUa_SemanticChangeStructureDataType*
- *OpcUa_ServerDiagnosticsSummaryDataType*
- *OpcUa_ServerOnNetwork*
- *OpcUa_ServerState*
- *OpcUa_ServerStatusDataType*
- *OpcUa_ServiceCounterDataType*
- *OpcUa_ServiceFault*
- *OpcUa_SessionDiagnosticsDataType*
- *OpcUa_SessionlessInvokeRequestType*
- *OpcUa_SessionlessInvokeResponseType*
- *OpcUa_SessionSecurityDiagnosticsDataType*
- *OpcUa_SetMonitoringModeRequest*
- *OpcUa_SetMonitoringModeResponse*
- *OpcUa_SetPublishingModeRequest*
- *OpcUa_SetPublishingModeResponse*
- *OpcUa_SetTriggeringRequest*
- *OpcUa_SetTriggeringResponse*
- *OpcUa_SignatureData*
- *OpcUa_SignedSoftwareCertificate*
- *OpcUa_SimpleAttributeOperand*
- *OpcUa_StatusChangeNotification*
- *OpcUa_StatusCode*
- *OpcUa_StatusResult*
- *OpcUa_String*
- *OpcUa_StructureDefinition*
- *OpcUa_StructureField*
- *OpcUa_StructureType*
- *OpcUa_SubscriptionAcknowledgement*
- *OpcUa_SubscriptionDiagnosticsDataType*
- *OpcUa_TimestampsToReturn*
- *OpcUa_TimeZoneDataType*
- *OpcUa_TransferResult*
- *OpcUa_TransferSubscriptionsRequest*
- *OpcUa_TransferSubscriptionsResponse*
- *OpcUa_TranslateBrowsePathsToNodeIdsRequest*

- *OpcUa_TranslateBrowsePathsToNodeIdsResponse*
- *OpcUa_UInt*
- *OpcUa_Uint16*
- *OpcUa_UInt32*
- *OpcUa_UInt64*
- *OpcUa_UnregisterNodesRequest*
- *OpcUa_UnregisterNodesResponse*
- *OpcUa_UpdateDataDetails*
- *OpcUa_UpdateEventDetails*
- *OpcUa_UpdateStructureDataDetails*
- *OpcUa_UserIdentityToken*
- *OpcUa_UserNameIdentityToken*
- *OpcUa_UserTokenPolicy*
- *OpcUa_UserTokenType*
- *OpcUa_VariableAttributes*
- *OpcUa_VariableTypeAttributes*
- *OpcUa_Variant*
- *OpcUa_VariantArrayType*
- *OpcUa_VariantArrayUnion*
- *OpcUa_VariantArrayValue*
- *OpcUa_VariantMatrixValue*
- *OpcUa_VariantUnion*
- *OpcUa_ViewAttributes*
- *OpcUa_ViewDescription*
- *OpcUa_WriteRequest*
- *OpcUa_WriteResponse*
- *OpcUa_WriteValue*
- *OpcUa_X509IdentityToken*
- *OpcUa_XVType*
- *OpcUaApplicationDescriptionClear*
- *OpcUaApplicationDescriptionInitialize*
- *OpcUaBrowsePathClear*
- *OpcUaBrowsePathInitialize*
- *OpcUaBrowsePathResultClear*
- *OpcUaBrowsePathResultInitialize*
- *OpcUaBrowseResultClear*
- *OpcUaBrowseResultInitialize*
- *OpcUaByteStringClear*
- *OpcUaByteStringCompare*
- *OpcUaByteStringConcatenate*
- *OpcUaByteStringCopyTo*
- *OpcUaByteStringInitialize*
- *OPCUAClient_Browse*
- *OPCUAClient_BrowseNext*
- *OPCUAClient_Call*
- *OPCUAClient_Connect*
- *OPCUAClient_Create*
- *OPCUAClient_CreateMonitoredItems*
- *OPCUAClient_CreateSubscription*
- *OPCUAClient_Delete*
- *OPCUAClient_DeleteMonitoredItems*
- *OPCUAClient_DeleteSubscription*
- *OPCUAClient_Disconnect*
- *OPCUAClient_FindServers*
- *OPCUAClient_FindServersOnNetwork*

- *OPCUAClient_GetConfig*
- *OPCUAClient_GetEndpoints*
- *OPCUAClient_ModifyMonitoredItems*
- *OPCUAClient_ModifySubscription*
- *OPCUAClient_Read*
- *OPCUAClient_RegisterNodes*
- *OPCUAClient_SetDataChangeFilterStatic*
- *OPCUAClient_SetEventFilterStatic*
- *OPCUAClient_SetMonitoringMode*
- *OPCUAClient_SetPublishingMode*
- *OPCUAClient_TranslateBrowsePathsToNodeIds*
- *OPCUAClient_UnregisterNodes*
- *OPCUAClient_Write*
- *OPCUAClientConnectionConfiguration*
- *OPCUAClientConnectionState*
- *OPCUAClientCredentials*
- *OPCUAClientMonitoredItemConfiguration*
- *OPCUAClientMonitoredItemState*
- *OPCUAClientSubscriptionState*
- *OPCUAClientUserToken*
- *OpcUaDataTypeDescription*
- *OpcUaDataValueClear*
- *OpcUaDataValueCompare*
- *OpcUaDataValueCopyTo*
- *OpcUaDataValueInitialize*
- *OpcUaDateTimeUtcNow*
- *OpcUaElementDescription*
- *OpcUaEndpointDescriptionClear*
- *OpcUaEndpointDescriptionInitialize*
- *OpcUaEventFieldListClear*
- *OpcUaEventFieldListInitialize*
- *OpcUaEventNotificationListClear*
- *OpcUaEventNotificationListInitialize*
- *OpcUaExpandedNodeIdClear*
- *OpcUaExpandedNodeIdCompare*
- *OpcUaExpandedNodeIdCopyTo*
- *OpcUaExpandedNodeIdInitialize*
- *OpcUaExpandedNodeIdIsNull*
- *OpcUaExtensionObjectClear*
- *OpcUaExtensionObjectCompare*
- *OpcUaExtensionObjectCopyTo*
- *OpcUaExtensionObjectCreate*
- *OpcUaExtensionObjectDelete*
- *OpcUaExtensionObjectInitialize*
- *OpcUaLocalizedTextClear*
- *OpcUaLocalizedTextCompare*
- *OpcUaLocalizedTextCopyTo*
- *OpcUaLocalizedTextInitialize*
- *OpcUaMetaDataType*
- *OpcUaMethodDescription*
- *OpcUaMethodMetaData*
- *OpcUaNodeIdClear*
- *OpcUaNodeIdCompare*
- *OpcUaNodeIdCopyTo*
- *OpcUaNodeIdInitialize*

- *OpcUaNodeIdIsNull*
- *OpcUaNodeMetaData*
- *OpcUaObjectDescription*
- *OpcUaObjectTypeDescription*
- *OpcUaOwnDataTypeMetaData*
- *OpcUaQualifiedNameClear*
- *OpcUaQualifiedNameCompare*
- *OpcUaQualifiedNameCopyTo*
- *OpcUaQualifiedNameInitialize*
- *OpcUaReadValueIdClear*
- *OpcUaReadValueIdInitialize*
- *OpcUaReferenceDescriptionClear*
- *OpcUaReferenceDescriptionInitialize*
- *OpcUaReferenceTypeDescription*
- *OpcUaServer_MessageSecurityMode*
- *OpcUaServer_Session_Information*
- *OpcUaServer_SessionEvents*
- *OpcUaServerGetFirstSession*
- *OpcUaServerGetNextSession*
- *OpcUaServerGetSessionInfo*
- *OpcUaServerNodeDescription*
- *OpcUaServerOnNetworkClear*
- *OpcUaServerOnNetworkInitialize*
- *OpcUaServerReferenceDescription*
- *OpcUaSimpleAttributeOperandClear*
- *OpcUaSimpleAttributeOperandInitialize*
- *OpcUaStatusChangeNotificationClear*
- *OpcUaStatusChangeNotificationInitialize*
- *OpcUaStringAttachCopy*
- *OpcUaStringAttachReadOnly*
- *OpcUaStringAttachToString*
- *OpcUaStringClear*
- *OpcUaStringGetRawString*
- *OpcUaStringInitialize*
- *OpcUaStringIsEmpty*
- *OpcUaStringIsNull*
- *OpcUaStringStrLen*
- *OpcUaStringStrnCat*
- *OpcUaStringStrnCmp*
- *OpcUaStringStrnCpy*
- *OpcUaStringStrSize*
- *OpcUaTypeMetaData*
- *OpcUaTypeMetaDataUnion*
- *OpcUaVariableDescription*
- *OpcUaVariableTypeDescription*
- *OpcUaVariantClear*
- *OpcUaVariantCompare*
- *OpcUaVariantCopyTo*
- *OpcUaVariantInitialize*
- *OpcUaViewDescription*
- *OpcUaWellKnownDataTypeMetaData*
- *Open*
- *OPERATION_FWK_ACCESS_ADDRESS*
- *OPERATION_FWK_ACCESS_CONFIG*
- *OPERATION_FWK_ACCESS_PARAMETER*

- *OPERATION_FWK_CALLINFO*
- *OPERATION_FWK_GET_DEV_STATUS_PARAMETER*
- *OPERATION_FWK_SEND_COMMAND*
- *OPERATION_FWK_SEND_PARAMETER*
- *OPERATION_FWK_SET_PARAMETER*
- *OPERATION_FWK_START_SCAN*
- *OPERATION_FWK_STATUS_PARAMETER*
- *OPERATION_FWK_TEST_ADDRESS*
- *OperationsQueue*
- *OS*
- *OurVarInfo*
- *PACK*
- *PackArrayOfBoolToArrayOfByte*
- *PackBitsToByte*
- *PackBitsToDword*
- *PackBitsToWord*
- *PackBytesToDword*
- *PackBytesToWord*
- *PacketPool*
- *PacketPoolFactoryArgs*
- *PacketPoolFactoryBase*
- *PacketReader*
- *PacketWriter*
- *PackWordsToDword*
- *PaintCmdAndEventListener*
- *PaintRectangle*
- *Pair_DintDint*
- *Pair_PStringDint*
- *Pair_PStringXWORD*
- *Pair_StringDint*
- *Pair_StringString*
- *PARAM_ID*
- *PARAMETER*
- *ParameterServiceResult*
- *PARITY*
- *ParseCANID*
- *ParsePGN*
- *ParseXML2*
- *PB_CNCT*
- *PB_SLAVE_CIFX_DIAG*
- *PB_SlaveActivation*
- *PB_SlaveConfigurationData*
- *PBS_CONFIG_STATES*
- *PBScanData*
- *PBSlave*
- *PBSlaveDiag*
- *PCB*
- *PCI_INFO*
- *PciInterrupt*
- *PD*
- *PenStyle*
- *PERIOD*
- *PERIODE*
- *PERIODE_INFO*
- *Persistence*

- *PersistenceWriteProperty*
- *PERSISTENT_DATA_BUFFER*
- *PERSISTENT_INDEX_HEADER*
- *PERSISTENT_PDATA_ENTRY*
- *PERSISTENT_PDATA_HEADER*
- *PFSYS_TASK_EXCEPTIONHANDLER*
- *PFSYS_TASK_FUNCTION*
- *PFTIMERCALLBACK*
- *PFTIMEREXCEPTIONHANDLER*
- *PG_TYPE*
- *PID*
- *PID_FIXCYCLE*
- *PINGROUP*
- *PLANE_H*
- *PLC_IDENT*
- *PlcConnectionInitFlags*
- *PlcCryptType*
- *PlcOperationControl*
- *PlcShellAppend*
- *PlcShellRegister*
- *PlcShellSetEof*
- *PlcShellSkip*
- *PlcShellUnregister*
- *PM_VERSION*
- *PmBatt*
- *PmDiskLifetimeUsed*
- *PmDiskStatus*
- *PmDispSetText*
- *PmEcoResetFRAM*
- *PmErrLedSet*
- *PmGetDeviceState*
- *PmGetPlcId*
- *PmLedSet*
- *PmNtpInfo*
- *PmPlcReboot*
- *PmProdReadAsync*
- *PmRealtimeClock*
- *PmRealtimeClockDT*
- *PmSntpInfo*
- *PmSramCleared*
- *PmSramExport*
- *PmSramImport*
- *PmSysTime*
- *PmVersion*
- *PN_ADDR*
- *PN_AINFO*
- *PN_DEVICE_ID*
- *PN_PortConfiguration*
- *PN_PortConfigurationRecord*
- *PNIO_COMM_ERNO_TYPE*
- *PNIO_MST_STATE_TYPE*
- *PnioCntrlGetCntrlState*
- *PnioCntrlGetDevIM0Data*
- *PnioCntrlGetDevState*
- *PnioCntrlRead*

- *PnioCntrlStartCom*
- *PnioCntrlStopCom*
- *PnioCntrlWrite*
- *PnioImSwRevType*
- *PnioImVersionType*
- *PNM_AP_CFG_OEMPRM_DEVICE_IDENTITY_T*
- *PNS_CONFIG_STATES*
- *PNS_DIAG*
- *PNS_DIAG_LinkState*
- *PNS_IF_APDU_STATUS_CHANGED_IND_DATA_T*
- *PNS_IF_APDU_STATUS_CHANGED_IND_T*
- *PNS_IF_AR_ABORT_IND_IND_T*
- *PNS_IF_AR_CHECK_IND_DATA_T*
- *PNS_IF_AR_CHECK_IND_T*
- *PNS_IF_AR_IN_DATA_IND_T*
- *PNS_IF_CHECK_IND*
- *PNS_IF_CHECK_IND_DATA_T*
- *PNS_IF_CMD*
- *PNS_IF_EVENT_IND_T*
- *PNS_IF_GET_ASSET_IND_DATA_T*
- *PNS_IF_GET_ASSET_IND_T*
- *PNS_IF_GET_IP_ADDR_CNF_DATA_T*
- *PNS_IF_GET_IP_ADDR_CNF_T*
- *PNS_IF_GET_STATION_NAME_CNF_DATA_T*
- *PNS_IF_GET_STATION_NAME_CNF_T*
- *PNS_IF_LOAD_REMANENT_DATA_REQ*
- *PNS_IF_READ_RECORD_IND_DATA_T*
- *PNS_IF_READ_RECORD_IND_T*
- *PNS_IF_READ_RECORD_RSP_DATA_T*
- *PNS_IF_READ_RECORD_RSP_T*
- *PNS_IF_RESET_FACTORY_SETTINGS_IND_T*
- *PNS_IF_START_LED_BLINKING_IND_T*
- *PNS_IF_STORE_REMANENT_DATA_IND_T*
- *PNS_IF_USER_ERROR_IND_DATA_T*
- *PNS_IF_USER_ERROR_IND_T*
- *PNSlave*
- *PNSlaveDiag*
- *POINT*
- *Point*
- *POINT2_DINT*
- *POINT2_LREAL*
- *PointArrayCalcSurroundingSimpleRect*
- *PolarToCartesian*
- *PolygonType*
- *PolynomialValue*
- *PoolClass*
- *PoolCreateH*
- *PoolCreateP*
- *PoolDelete*
- *PoolExtendH*
- *PoolGetBlock*
- *PoolGetSize*
- *PoolPutBlock*
- *PopTransformation*
- *Port*

- *PortStatus*
- *PostEvent*
- *PrimaryTables*
- *Printf*
- *PrintfW*
- *private_iomgr_memcpy*
- *PROC_CMD*
- *PROC_STATE*
- *ProfibusBaudrate*
- *ProfinetByteData*
- *ProfinetConfigType*
- *ProfinetController*
- *ProfinetControllerDiag*
- *ProfinetDevice*
- *ProfinetDeviceDiag*
- *ProfinetDeviceInstance*
- *ProfinetSubmodule*
- *PROJECT_INFO*
- *ProjectPointOnLine*
- *ProjectPointOnPlane*
- *PropAddrString*
- *PropertyAddrString*
- *PropertyAttributeExistenceString*
- *PropertyAttributePersistentString*
- *PropertyAttributesString*
- *PropertyAttributeWritableString*
- *PropertyConfiguration*
- *PropertyConfigurationMostlyAllPersistent*
- *PropertyConfigurationMostlyAllWritable*
- *PropertyConfigurationObjectPropertyPair*
- *PropertyContentToString*
- *PropertyIndexAddrString*
- *PropertyInfo*
- *PropertyInfoRemote*
- *PropertyLocation*
- *PROTOCOL_DATA_UNIT*
- *ProtocolDataUnit*
- *ProxyEnumState*
- *ProxyFbHistActiveAlarmsQueue*
- *ProxyFbHistAlarmsRowQueue*
- *ProxyStructError*
- *ProxyStructMonitor*
- *ProxyStructMonitorAlarmClassDesc*
- *ProxyStructMonitorAlarmDesc*
- *ProxyStructMonitorAlarmGroupDesc*
- *ProxyStructMonitorRequest*
- *PRVREC*
- *PRVREC_MODE*
- *PStrCat*
- *PStrCmp*
- *PStrICmp*
- *PStrIFind*
- *PStringElement*
- *PStringElementFactory*
- *PStringToDintMap*

- *PStringToXWORDMap*
- *PStringVector*
- *PStringVectorArrAccess*
- *PStrLen*
- *PStrLenUntil*
- *PStrNICmp*
- *PStrToUpper*
- *PT_SIZE*
- *PtrToString*
- *PtrVectorArrAccess*
- *PURPOSE*
- *PushTransformation*
- *PUTBIT*
- *PVOID*
- *PVOID_TO_DWORD*
- *PVOID_TO_LWORD*
- *PVOID_TO_WORD*
- *QOS_INFO*
- *Queue*
- *QueueFactory*
- *QuickSortAddrItemHelpers*
- *R_TRIG*
- *RaiseModuleEvent*
- *RALARM*
- *RALARM_MODE*
- *RAMP_INT*
- *RAMP_REAL*
- *RCSINFO*
- *RCVREC*
- *RCVREC_MODE*
- *RCX_SET_WATCHDOG_TIME_CNF_T*
- *RCX_SET_WATCHDOG_TIME_REQ_DATA_T*
- *RCX_SET_WATCHDOG_TIME_REQ_T*
- *RDIAG*
- *RDREC*
- *RDT_Base*
- *RDT_Client*
- *RDT_ERROR*
- *RDT_Server*
- *RdtInitStructClientTCP*
- *RdtInitStructServerTCP*
- *RdtProtStructCommPh*
- *RdtProtStructConnection*
- *RdtProtStructResPh*
- *Read*
- *ReadableRequestBase*
- *ReadArbitraryStringFromBuffer*
- *readBit*
- *ReadEEpromData*
- *ReadIdentification*
- *ReadItemInfo*
- *ReadItemVector*
- *ReadMemory*
- *ReadNbrSlaves*
- *ReadRequest*

- *ReadRequestState*
- *ReadWriteEEprom*
- *REAL_TO_FLOAT*
- *REAL8*
- *REAL8_TO_DT*
- *REAL8_TO_LTIME*
- *REAL8_TO_TIME*
- *RealToHexStr*
- *RealToStr*
- *ReceiveParameterGroup*
- *ReceiveWatchdog*
- *Recipe_FileParameters*
- *RecipeMan_FctTypeClassToDataType*
- *RecipeManCommands*
- *Reconfigure*
- *RectangleType*
- *RECV_EMCY*
- *RECV_EMCY_DEV*
- *REDUNDANCY_CONNECTION_INFO*
- *RedundancyState*
- *RedundancyStatus*
- *RedundancySynchronizeData*
- *RegContext*
- *Register*
- *RegisterCallback*
- *RegisterIdArea*
- *ReinitDevice_SvcAppHook*
- *RemoteAdapter*
- *RemoteAdapter_Diag*
- *RemoteAdapter_diag*
- *RemotePlcRequestIdentification*
- *RemoteProcedureCall*
- *RemoteVarInfo*
- *RemoteVarResolver*
- *Rename*
- *ReparseIOMemoryAccessExpression*
- *REPLACE*
- *ReplaceAlarmPlaceholderString*
- *Request*
- *RequestData*
- *RequestDataDiagnostics*
- *RequestDataMaskWriteRegister*
- *RequestDataRead*
- *RequestDataReadWriteMultipleRegisters*
- *RequestDataWriteMultiple*
- *RequestDataWriteSingle*
- *RequestFactory*
- *RequestStatus*
- *RequestUnion*
- *RequestVector*
- *Reset*
- *RESET_INIT*
- *RESET_OPTION*
- *RESET_RESET*
- *ResetBusAlarm*

- *ResetNodeInfo*
- *ResetNodeInfoInt*
- *ResolveHostname*
- *ReturnValues*
- *ReusableRequestInfo*
- *ReusableRequestState*
- *ReverseBitsInBYTE*
- *ReverseBitsInDWORD*
- *ReverseBitsInWORD*
- *ReverseBYTEsInDWORD*
- *ReverseBYTEsInWORD*
- *ReverseWORDsInDWORD*
- *RIGHT*
- *RLstAddPrio*
- *RLstCheckPrio*
- *RLstClass*
- *RLstCreateH*
- *RLstCreateP*
- *RLstDelete*
- *RLstGetHighestPrio*
- *RLstGetSize*
- *RLstRemovePrio*
- *RootDatasourceIndex*
- *RootPseudo*
- *RootRenamed*
- *RootRenamedDatasourceIndex*
- *RotatePoint*
- *RouterGetHostAddress*
- *RouterGetInstanceByName*
- *RouterGetName*
- *RouterGetParentAddress*
- *RPCDataRepresentation*
- *RPCNCARejectStatus*
- *RS*
- *RSM_HANDLE*
- *RSMClass*
- *RTC*
- *RTCLK_GETDATEANDTIME_PARAMS*
- *RTCLK_GETTIMEZONEINFORMATION_PARAMS*
- *RTCLK_PERIODE_INFO*
- *RTCLK_SETDATEANDTIME_PARAMS*
- *RTCLK_SETTIMEZONEINFORMATION_PARAMS*
- *RTCLK_SYSTEMTIME*
- *RTCLK_TIME_ZONE_INFO*
- *RTR_AddrComponent*
- *RTR_NodeAddress*
- *RTS_CMBOXENTRY*
- *RTS_CODEMETER_INFO*
- *RTS_CONTROL*
- *RTS_IEC_CWCHAR*
- *RTS_IEC_HANDLE*
- *RTS_IEC_RESULT*
- *RTS_IEC_SIZE*
- *RTS_SIL2_ADDRESSSTATE*
- *RTS_SIL2_CALLERCTX*

- *RTS_SIL2_EXCEPTION*
- *RTS_SIL2_OPMODE*
- *RTS_SOCKET_SO_VALUE_IP_MREQ*
- *RTS_SOCKET_SO_VALUE_LINGER*
- *RTS_SOCKET_SO_VALUE_TCP_KEEPALIVE*
- *RTS_SYSTIMEDATE*
- *RTS_SYSTIMEDATE_TO_STRING*
- *RtsAL1030Handler*
- *RtsBrowseInfo*
- *RtsByteString*
- *RtsCertEncoding*
- *RtsCertTrustLevel*
- *RtsCryptoID*
- *RtsCryptoKey*
- *RtsCryptoKeyStorage*
- *RtsCryptoKeyType*
- *RtsCryptoType*
- *RtsEL6224Handler*
- *RtsKdfParameter*
- *RtsOID*
- *RtsOIDClear*
- *RtsOIDCreate*
- *RtsOIDGetID*
- *RtsOIDGetName*
- *RtsOIDStore*
- *RtsScryptParameter*
- *RtsServicehandlerBase*
- *RtsServicehandlerBase2*
- *RtsX509AltName*
- *RtsX509AltNameStore*
- *RtsX509AltNameType*
- *RtsX509CertCheckFlags*
- *RtsX509CertFilter*
- *RtsX509CertFilterContent*
- *RtsX509CertFilterType*
- *RtsX509CertInfo*
- *RtsX509CertName*
- *RtsX509ExKeyUsage*
- *RtsX509NameEntry*
- *RtuAscii*
- *RudimentaryDeviceInfo*
- *RUNE*
- *RuneCount*
- *RuneLen*
- *RuntimeCredentialsHandler*
- *SAdapterFlags*
- *SAFE_SRDO_DATA*
- *SAFE_SRDO_RECEIVED*
- *SAFETY_EXCHANGE*
- *SAFETY_STATE*
- *SafetyMemCpy*
- *SALARM*
- *ScalePoint*
- *ScalProd3D*
- *ScalProd3DStand*

- *ScannerState*
- *SchedGetCurrentTask*
- *SchedGetNumOfTasks*
- *SchedGetProcessorLoad*
- *SchedGetTaskEventByHandle*
- *SchedGetTaskHandleByIndex*
- *SchedGetTaskHandleByName*
- *SchedGetTaskInterval*
- *SchedPostExternalEvent*
- *SchedRegisterExternalEvent*
- *SchedSetTaskInterval*
- *Schedule*
- *SchedUnregisterExternalEvent*
- *SchedWaitBusy*
- *SchedWaitSleep*
- *SDO_ABORT*
- *SDO_ERROR*
- *SDO_MODE*
- *SDO_READ*
- *SDO_READ_DATA*
- *SDO_READ4*
- *SDO_WRITE*
- *SDO_WRITE_DATA*
- *SDO_WRITE4*
- *SdoAbort*
- *SdoRead*
- *SDOServerClose*
- *SDOServerDoCycle*
- *SDOServerOpen*
- *SdoWrite*
- *SECOND*
- *SecurityModeToString*
- *SEEK_MODE*
- *Segment*
- *SegmentPool*
- *SegmentPoolFactoryArgs*
- *SegmentPoolFactoryBase*
- *Select*
- *SEMA*
- *SendEvent*
- *SeparateDateTime*
- *SERCOS_TOPOLOGY*
- *SERCOS3_ERROR*
- *Sercos3_IDNCmd*
- *Sercos3_IDNRead*
- *Sercos3_IDNRead4*
- *Sercos3_IDNWrite*
- *Sercos3_IDNWrite4*
- *Sercos3Master_GetVersion*
- *Sercos3Slave*
- *Sercos3Slave_Diag*
- *SerializeHexReal*
- *SerialSubFunctionCodes*
- *Server*
- *ServerCapabilities*

- *ServerCapabiltiesReader*
- *ServerClass*
- *ServerSerial*
- *ServerSide*
- *ServerStructCommand*
- *ServerTCP*
- *ServiceGroup*
- *SERVICEHANDLER_PARAMETER*
- *ServiceHeader*
- *ServiceReader*
- *ServiceRequest*
- *ServiceRequestBase*
- *ServiceRequestRaw*
- *ServiceResponse*
- *ServiceResult*
- *ServiceWriter*
- *ServiceWriterSavepoint*
- *Set_Attribute_Single*
- *Set_Attributes_All*
- *SETBIT*
- *SetBitValue*
- *SetCiAState*
- *SetCustomMapping*
- *SetDateAndTime*
- *SetError*
- *SETIO_PART*
- *SetLastError*
- *SetPaintRectangle*
- *SetParent*
- *SetPos*
- *SetPropertyAgain*
- *SetResult*
- *SetSimpleRectangle*
- *SettgBeginUpdate*
- *SettgEndUpdate*
- *SettgGetIntValue*
- *SettgGetStringValue*
- *SettgGetWStringValue*
- *SettgRemoveKey*
- *SettgSetIntValue*
- *SettgSetStringValue*
- *SettgSetWStringValue*
- *SetTimeZoneInformation*
- *SettingsHelper*
- *SettingValue*
- *Severity*
- *SFCActionControl*
- *SFCActionType*
- *SFCStepType*
- *sgn*
- *SharedArea*
- *SharedAreaFactoryArgs*
- *SharedAreaFactoryBase*
- *SharedAreaRefDisposer*
- *SharedPointer*

- *SharedPointerFactoryArgs*
- *SharedPointerFactoryBase*
- *SharedQueue*
- *SharedQueueFactoryArgs*
- *SharedQueueFactoryBase*
- *SIGNED*
- *SIGNED_TO_DINT*
- *SIGNED_TO_INT*
- *SIGNED_TO_LINT*
- *SIL2AddLog*
- *SIL2CheckCallerContext*
- *SIL2CopyCodeGuid*
- *SIL2CopyDataGuid*
- *SIL2ExecuteNonSafetyJob*
- *SIL2ExecuteNonSafetyJob_WRAP_FB_INIT*
- *SIL2ExecuteNonSafetyJob_WRAP_INITIALIZE*
- *SIL2OEMException*
- *SIL2OEMGetCallerContext*
- *SIL2OEMGetMemoryState*
- *SIL2OEMGetOperationMode*
- *SIL2OEMStackIsValid*
- *SimpleRectangle*
- *SIZE*
- *SIZE_TO_UDINT*
- *SIZE_TO_UINT*
- *SIZE_TO_ULINT*
- *SlaveDiag*
- *SlaveStateBitFieldType*
- *SLOT_ID*
- *Sm560Rec*
- *Sm560Send*
- *SNCM_ETC_Slave*
- *SNCM_ETC_VoE_SendReceive*
- *SntpSourceInfoData*
- *SntpSourceMode*
- *SntpSourceState*
- *SOCK_ADAPTER_INFORMATION*
- *SOCK_ADAPTER_INFORMATION2*
- *SOCK_HOSTENT*
- *SockAddr*
- *SOCKADDRESS*
- *SOCKET_FD_SET*
- *SOCKET_TIMEVAL*
- *SocketType*
- *softing_profi_end*
- *softing_profi_get_data*
- *softing_profi_get_dps_input_data*
- *softing_profi_get_dps_output_data*
- *softing_profi_get_last_errror*
- *softing_profi_get_serial_device_number*
- *softing_profi_get_versions*
- *softing_profi_init*
- *softing_profi_rcv_con_ind*
- *softing_profi_set_data*
- *softing_profi_set_dps_input_data*

- *softing_profi_snd_req_res*
- *SOFTING_T_DP_AAT_DATA*
- *SOFTING_T_DP_ACT_PARAM_IND*
- *SOFTING_T_DP_ACT_PARAM_REQ*
- *SOFTING_T_DP_ACT_PARAM_RES_CON*
- *SOFTING_T_DP_BUS_PARA_SET*
- *SOFTING_T_DP_CFG_DATA*
- *SOFTING_T_DP_DATA_TRANSFER_CON*
- *SOFTING_T_DP_DIAG_DATA*
- *SOFTING_T_DP_DOWNLOAD_IND*
- *SOFTING_T_DP_DOWNLOAD_REQ*
- *SOFTING_T_DP_DOWNLOAD_RES_CON*
- *SOFTING_T_DP_END_SEQ_IND*
- *SOFTING_T_DP_END_SEQ_REQ*
- *SOFTING_T_DP_END_SEQ_RES_CON*
- *SOFTING_T_DP_EXIT_MASTER_CON*
- *SOFTING_T_DP_GET_MASTER_DIAG_REQ*
- *SOFTING_T_DP_GET_MASTER_DIAG_RES_CON*
- *SOFTING_T_DP_GET_PRM_REQ*
- *SOFTING_T_DP_GET_SLAVE_DIAG_CON*
- *SOFTING_T_DP_GET_SLAVE_DIAG_IND*
- *SOFTING_T_DP_GET_SLAVE_PARAM_CON*
- *SOFTING_T_DP_GET_SLAVE_PARAM_REQ*
- *SOFTING_T_DP_INIT_MASTER_CON*
- *SOFTING_T_DP_INIT_MASTER_REQ*
- *SOFTING_T_DP_PRM_DATA*
- *SOFTING_T_DP_SET_BUSPARAMETER_CON*
- *SOFTING_T_DP_SET_BUSPARAMETER_REQ*
- *SOFTING_T_DP_SET_PRM_CON*
- *SOFTING_T_DP_SET_PRM_REQ*
- *SOFTING_T_DP_SLAVE_PARA_SET*
- *SOFTING_T_DP_SLAVE_PARAM_SLAVE_INFO*
- *SOFTING_T_DP_SLAVE_PARAM_SYS_INFO*
- *SOFTING_T_DP_SLAVE_USER_DATA*
- *SOFTING_T_DP_START_SEQ_IND*
- *SOFTING_T_DP_START_SEQ_REQ*
- *SOFTING_T_DP_START_SEQ_RES_CON*
- *SOFTING_T_DP_UPLOAD_REQ*
- *SOFTING_T_DP_UPLOAD_RES_CON*
- *SOFTING_T_FMB_CONFIG_CRL*
- *SOFTING_T_FMB_CONFIG_DP*
- *SOFTING_T_FMB_CONFIG_FDLIF*
- *SOFTING_T_FMB_CONFIG_SM7*
- *SOFTING_T_FMB_CONFIG_VFD*
- *SOFTING_T_FMB_FM2_EVENT_IND*
- *SOFTING_T_FMB_SET_CONFIGURATION_REQ*
- *SOFTING_T_PROFI_SERVICE_DESCR*
- *SortByAddrItemHelper*
- *SortedBranchNamedTreeNode*
- *SortedInstancePathBuildingBranchNode*
- *SortedList*
- *SortedListFactory*
- *SortedPStringVector*
- *SplitDateTime*
- *SplitString*

- *SplitTextListId*
- *SQLSTATEMENT*
- *SR*
- *SRAM_CLEARED*
- *SRAM_EXPORT*
- *SRAM_IMPORT*
- *SRDO_DATA*
- *SRDO_DIRECTION*
- *SRDO_LIST*
- *SRDO_STATE*
- *SRDOObject*
- *Stack*
- *StackFactory*
- *Start*
- *StatDynMemory*
- *STATE*
- *State*
- *StateFlags*
- *StateMachine*
- *StaticMemBuffer*
- *Statistics_DINT*
- *STATISTICS_INT*
- *Statistics_LREAL*
- *Statistics_LTIME*
- *STATISTICS_REAL*
- *STK_INFO*
- *STK_NODES*
- *STK_SPEC*
- *STK_STATE*
- *StkClose*
- *StkGetInfo*
- *StkOpen*
- *StkRegister*
- *StkUnregister*
- *STO_BLOB*
- *STO_METRICS*
- *STO_TEXT*
- *Stop*
- *STOPBIT*
- *Storage*
- *StrCaseCmpA*
- *StrCaseCmpEndA*
- *StrCaseCmpStartA*
- *StrCaseCmpW*
- *StrCaseFindA*
- *StrCaseFindW*
- *StrCmpA*
- *StrCmpEndA*
- *StrCmpStartA*
- *StrCmpW*
- *StrConcatA*
- *StrConcatW*
- *StrCpyA*
- *StrCpyW*
- *StrCpyWtoA*

- *StrDeleteA*
- *StrDeleteW*
- *Stream*
- *STREAM_STATE*
- *StrFindA*
- *StrFindW*
- *StringBuilder*
- *StringBuilderSysMemExtending*
- *StringElement*
- *StringElementFactory*
- *StringToDintMap*
- *StringToStringMap*
- *StringVector*
- *StrIsNullOrEmptyA*
- *StrIsNullOrEmptyW*
- *StrLenA*
- *StrLenW*
- *StrMidA*
- *StrMidW*
- *StrPadLeftA*
- *StrPadLeftW*
- *StrPadRightA*
- *StrPadRightW*
- *StrReplaceA*
- *StrReplaceW*
- *StrToLowerA*
- *StrToLReal*
- *StrToReal*
- *StrToUpperA*
- *StrTrimA*
- *StrTrimEndA*
- *StrTrimStartA*
- *STRUCT_BACNET_READ_FILE_RANGE_RECORD*
- *STRUCT_BACNET_READ_FILE_RANGE_STREAM*
- *STRUCT_BACNET_READ_RANGE_RANGE_POSITION*
- *STRUCT_BACNET_READ_RANGE_RANGE_SEQUENCE*
- *STRUCT_BACNET_READ_RANGE_RANGE_TIME*
- *STRUCT_BACNET_READ_RANGE_RANGE_TIMERANGE*
- *STRUCT_BACNET_WRITE_FILE_DATA_RECORD*
- *STRUCT_BACNET_WRITE_FILE_DATA_STREAM*
- *StructClientCommand*
- *StructClientCommandMonitor*
- *StructClientInitialize*
- *StructClientMonitor*
- *StructClientUseAsTCP*
- *StructCmdHandleCertificate*
- *StructCmdHandleClientAns*
- *StructCmdHandleClientAns2*
- *StructCmdHandleClientAnsSub*
- *StructCmdHandleClientAnsSub2*
- *StructCmdNewClient*
- *StructCmdNewFrame*
- *StructCmdNewLogin*
- *StructCmdNewPage*
- *StructCmdRemoveClient*

- *StructCmdValueChanged*
- *StructDataLogin*
- *StructFrame*
- *StructFrameDwnSL*
- *StructServerCommandMonitor*
- *StructServerInitialize*
- *StructServerMonitor*
- *StructServerUseAsTCP*
- *StructTicket*
- *StructValueChanged*
- *StructVisuClient*
- *StructVisuClientDwnSL*
- *StructVisuClientMonitor*
- *StuSprintf*
- *StuSprintfW*
- *StyleUtilFct_GetBoolFromStyle*
- *StyleUtilFct_GetBoolFromStyleEnumOrExplicitValue*
- *StyleUtilFct_GetSimpleTypeFromStyleEnumOrExplicitValue*
- *StyleUtilFct_GetUDIntFromStyle*
- *SubmoduleConfiguration*
- *SubmoduleDiagnosisEntry*
- *SubmoduleInfo*
- *SubmoduleIterator*
- *SubmoduleState_AddInfo*
- *SubmoduleState_ARInfo*
- *SubmoduleState_Detail*
- *SubmoduleState_IdentInfo*
- *SubmoduleStatus*
- *SubObjectIterator*
- *SubPoints*
- *SUBSLOT_ID*
- *SubVector*
- *SupervisorEntry*
- *SupervisorInstance*
- *SupervisorOperationAlive*
- *SupervisorOperationDead*
- *SupervisorOperationDisable*
- *SupervisorOperationEnable*
- *SupervisorOperationGetEntry*
- *SupervisorOperationGetFirst*
- *SupervisorOperationGetNext*
- *SupervisorOperationGetState2*
- *SupervisorOperationRegister*
- *SupervisorOperationSetTimeout*
- *SupervisorOperationUnregister*
- *SupervisorState*
- *SupportedFcs*
- *Swap*
- *SwapDword*
- *SwapLocalToIntel*
- *SwapLocalToMotorola*
- *SwapLword*
- *SwappedDirectAssigner*
- *SwapWord*
- *SWITCHBIT*

- *SwitchToActive*
- *SwitchToSimulation*
- *SwitchToStandalone*
- *SwitchToStandby*
- *SymbolicVarNodeAccessor*
- *SymbolicVarNodeFinder*
- *SymbolicVarsBaseHandleConverter*
- *SymbolInfo*
- *SymbolsBaseNode*
- *SymbolsBranchNode*
- *SymVarAccess*
- *SYNC_INFO*
- *SyncDefineVarList*
- *SyncDeleteVarList*
- *Synchronize*
- *SyncReadVarList*
- *SyncReadVarListFromPlc*
- *SyncReadVars*
- *SyncReadVarsRelease*
- *SyncSendService*
- *SyncWriteVarListToPlc*
- *SyncWriteVars*
- *SYS_COM_BAUDRATE*
- *SYS_COM_DTR_CONTROL*
- *SYS_COM_PARITY*
- *SYS_COM_PORTS*
- *SYS_COM_RTS_CONTROL*
- *SYS_COM_STOPBITS*
- *SYS_COM_TIMEOUT*
- *SYS_FILE_STATUS*
- *SYS_FILETIME*
- *SYS_INT_DESCRIPTION*
- *SYS_TASK_INFO*
- *SYS_TASK_PARAM*
- *SYS_TIME*
- *SysComAsyncFB*
- *SysComClose*
- *SysComGetSettings*
- *SysComGetSettings2*
- *SysComOpen*
- *SysComOpen2*
- *SysComOpen3*
- *SysComPurge*
- *SysComRead*
- *SysComSetSettings*
- *SysComSetSettings2*
- *SysComSetTimeout*
- *SysComSettings*
- *SysComSettingsEx*
- *SysComSettingsEx2*
- *SysComWrite*
- *SysCpuAtomicAdd*
- *SysCpuAtomicAdd64*
- *SysCpuAtomicCompareAndSwap*
- *SysCpuCallIecFuncWithParams*

- *SysCpuResetBit*
- *SysCpuResetBit2*
- *SysCpuSetBit*
- *SysCpuSetBit2*
- *SysCpuTestAndReset*
- *SysCpuTestAndSet*
- *SysCpuTestAndSetBit*
- *SysDirAsyncFB*
- *SysDirClose*
- *SysDirCopy*
- *SysDirCreate*
- *SysDirCreate2*
- *SysDirDelete*
- *SysDirDelete2*
- *SysDirGetCurrent*
- *SysDirOpen*
- *SysDirRead*
- *SysDirRename*
- *SysDirSetCurrent*
- *SysEthernetAdapterClose*
- *SysEthernetAdapterOpen*
- *SysEthernetCapabilities*
- *SysEthernetEthFrameReceive*
- *SysEthernetEthFrameSend*
- *SysEthernetFrame*
- *SysEthernetFrameRelease*
- *SysEthernetGetCapabilities*
- *SysEthernetGetInterfaceCounters*
- *SysEthernetGetMediaCounters*
- *SysEthernetGetPortConfigAndStatus*
- *SysEthernetInterfaceCounters*
- *SysEthernetIpFrameReceive*
- *SysEthernetIpFrameSend*
- *SysEthernetMediaCounters*
- *SysEthernetPortConfigAndStatus*
- *SysEthernetSetAutoNegAdvertisedCap*
- *SysEthernetSetAutoNegMode*
- *SysEthernetSetMauType*
- *SysEventCreate*
- *SysEventDelete*
- *SysEventSet*
- *SysEventWait*
- *SysExceptGenerateException*
- *SysFileAsyncFB*
- *SysFileClose*
- *SysFileCopy*
- *SysFileDelete*
- *SysFileDeleteByHandle*
- *SysFileEOF*
- *SysFileFlush*
- *SysFileGetName*
- *SysFileGetName2*
- *SysFileGetPath*
- *SysFileGetPos*
- *SysFileGetSize*

- *SysFileGetSizeByHandle*
- *SysFileGetStatus*
- *SysFileGetStatus2*
- *SysFileGetTime*
- *SysFileIoctl*
- *SysFileOpen*
- *SysFileRead*
- *SysFileRename*
- *SysFileSetPos*
- *SysFileTruncate*
- *SysFileWrite*
- *SysGraphicLightBeginPaint*
- *SysGraphicLightDrawBitmap*
- *SysGraphicLightDrawPolygon*
- *SysGraphicLightDrawRect*
- *SysGraphicLightDrawText*
- *SysGraphicLightEndPaint*
- *SysGraphicLightGetDisplayDeviceContext*
- *SysGraphicLightRegisterFont*
- *SysGraphicLightReleaseDisplayDeviceContext*
- *SysGraphicLightSetFill*
- *SysGraphicLightSetFont*
- *SysGraphicLightSetLine*
- *SysIntClose*
- *SysIntDisable*
- *SysIntDisableAll*
- *SysIntEnable*
- *SysIntEnableAll*
- *SysIntLevel*
- *SysIntOpen*
- *SysIntOpenByName*
- *SysIntRegister*
- *SysIntUnregister*
- *SysMCBDAlloc*
- *SysMCBDCount*
- *SysMCBDFree*
- *SysMCBDGetFirstID*
- *SysMCBDGetNextID*
- *SysMCBDIsSet*
- *SysMCGetLoad*
- *SysMCGetNumOfCores*
- *SysMCGetProcessBinding*
- *SysMCGetTaskBinding*
- *SysMemAllocData*
- *SysMemCmp*
- *SysMemCpy*
- *SysMemForceSwap*
- *SysMemFreeData*
- *SysMemGetCurrentHeapSize*
- *SysMemIsValidPointer*
- *SysMemMove*
- *SysMemReallocData*
- *SysMemSet*
- *SysMemSwap*
- *SysPciGetCardInfo*

- *SysPciGetConfigEntry*
- *SysPciReadValue*
- *SysPciSetConfigEntry*
- *SysPciWriteValue*
- *SysPipeWindowsClose*
- *SysPipeWindowsOpen*
- *SysPipeWindowsPeek*
- *SysPipeWindowsRead*
- *SysPipeWindowsSetHandleState*
- *SysPipeWindowsWrite*
- *SysPortAsyncFB*
- *SysPortIn*
- *SysPortInD*
- *SysPortInW*
- *SysPortOut*
- *SysPortOutD*
- *SysPortOutW*
- *SysProcessCreate*
- *SysProcessCreate2*
- *SysProcessExecuteCommand*
- *SysProcessExecuteCommand2*
- *SysProcessFreeHandle*
- *SysProcessGetCurrentHandle*
- *SysProcessGetOSId*
- *SysProcessGetPriority*
- *SysProcessGetState*
- *SysProcessResume*
- *SysProcessSetPriority*
- *SysProcessTerminate*
- *SysRWLCreate*
- *SysRWLDelete*
- *SysRWLReadLock*
- *SysRWLReadLockTry*
- *SysRWLReadUnlock*
- *SysRWLWriteLock*
- *SysRWLWriteLockTry*
- *SysRWLWriteUnlock*
- *SysSafetyAfterWriteOutput*
- *SysSafetyBeforeReadInput*
- *SysSafetyIoCfgReady*
- *SysSafetyMapShm*
- *SysSafetyReadConfigIdFromSafety*
- *SysSafetyUnmapShm*
- *SysSafetyWriteConfigIdOfStandard*
- *SysSemCreate*
- *SysSemDelete*
- *SysSemEnter*
- *SysSemLeave*
- *SysSemProcessCreate*
- *SysSemProcessDelete*
- *SysSemProcessEnter*
- *SysSemProcessLeave*
- *SysSemTry*
- *SysSharedMemoryClose*
- *SysSharedMemoryCreate*

- *SysSharedMemoryDelete*
- *SysSharedMemoryGetPointer*
- *SysSharedMemoryOpen2*
- *SysSharedMemoryRead*
- *SysSharedMemoryReadByte*
- *SysSharedMemoryWrite*
- *SysSharedMemoryWriteByte*
- *SysShmAsyncFB*
- *SysSock2Accept*
- *SysSock2Bind*
- *SysSock2Close*
- *SysSock2Connect*
- *SysSock2Create*
- *SysSock2FdInit*
- *SysSock2FdIsset*
- *SysSock2FdZero*
- *SysSock2GetOption*
- *SysSock2GetPeerName*
- *SysSock2GetSockName*
- *SysSock2Htonl*
- *SysSock2Htons*
- *SysSock2InetAddr*
- *SysSock2InetNtoa*
- *SysSock2Ioctl*
- *SysSock2Listen*
- *SysSock2Ntohl*
- *SysSock2Ntohs*
- *SysSock2Recv*
- *SysSock2RecvFrom*
- *SysSock2Select*
- *SysSock2Send*
- *SysSock2SendTo*
- *SysSock2SetOption*
- *SysSock2Shutdown*
- *SysSockAccept*
- *SysSockAsyncFB*
- *SysSockBind*
- *SysSockClose*
- *SysSockCloseUdp*
- *SysSockConnect*
- *SysSockCreate*
- *SysSockCreateUdp*
- *SysSocket2_Parameter*
- *SysSocket2_SpecificParameter*
- *SysSocket2_StdSockets*
- *SysSocket2_TlsSockets*
- *SysSocket2_Type*
- *SysSocketPair*
- *SysSockFdInit*
- *SysSockFdIsset*
- *SysSockFdZero*
- *SysSockGetAdapterInfo*
- *SysSockGetFirstAdapterInfo*
- *SysSockGetHostByName*
- *SysSockGetHostName*

- *SysSockGetNextAdapterInfo*
- *SysSockGetOption*
- *SysSockGetOSHandle*
- *SysSockGetPeerName*
- *SysSockGetRecvSizeUdp*
- *SysSockGetSockName*
- *SysSockGetSubnetMask*
- *SysSockHtonl*
- *SysSockHtons*
- *SysSockInetAddr*
- *SysSockInetNtoa*
- *SysSockIoctl*
- *SysSockListen*
- *SysSockNtohl*
- *SysSockNtohs*
- *SysSockPing*
- *SysSockRecv*
- *SysSockRecvFrom*
- *SysSockRecvFromUdp*
- *SysSockRecvFromUdp2*
- *SysSockSelect*
- *SysSockSend*
- *SysSockSendTo*
- *SysSockSendToUdp*
- *SysSockSetDefaultGateway*
- *SysSockSetIPAddress*
- *SysSockSetIpAddressAndNetMask*
- *SysSockSetOption*
- *SysSockSetSubnetMask*
- *SysSockShutdown*
- *SysTargetGetDeviceName*
- *SysTargetGetId*
- *SysTargetGetNodeName*
- *SysTargetGetOperatingSystemId*
- *SysTargetGetProcessorId*
- *SysTargetGetSerialNumber*
- *SysTargetGetType*
- *SysTargetGetVendorName*
- *SysTargetGetVersion*
- *SysTargetOperationNumber*
- *SysTaskAutoReleaseOnExit*
- *SysTaskCheckStack*
- *SysTaskCreate*
- *SysTaskCreate2*
- *SysTaskDestroy*
- *SysTaskEnd*
- *SysTaskEnter*
- *SysTaskExit*
- *SysTaskGenerateException*
- *SysTaskGetContext*
- *SysTaskGetCurrent*
- *SysTaskGetCurrentOSHandle*
- *SysTaskGetInfo*
- *SysTaskGetInterval*
- *SysTaskGetName*

- *SysTaskGetOSHandle*
- *SysTaskGetOSPriority*
- *SysTaskGetPriority*
- *SysTaskJoin*
- *SysTaskLeave*
- *SysTaskResume*
- *SysTaskSetExit*
- *SysTaskSetInterval*
- *SysTaskSetPriority*
- *SysTaskSuspend*
- *SysTaskWaitInterval*
- *SysTaskWaitSleep*
- *SysTaskWaitSleepUs*
- *SYSTEM_MEMORY_INFORMATION*
- *SystemParameter*
- *SYSTEMTIME*
- *SYSTIME*
- *SYSTIMEDATE*
- *SysTimeDateToString*
- *SysTimeGetMs*
- *SysTimeGetNs*
- *SysTimeGetUs*
- *SysTimeLock*
- *SysTimerCreateCallback*
- *SysTimerCreateCallback2*
- *SysTimerCreateEvent*
- *SysTimerDelete*
- *SysTimerGetInterval*
- *SysTimerGetTimeStamp*
- *SysTimerMaxTimer*
- *SysTimerSetInterval*
- *SysTimerStart*
- *SysTimerStop*
- *SysTimeRtcControl*
- *SysTimeRtcConvertDateToHighRes*
- *SysTimeRtcConvertDateToUtc*
- *SysTimeRtcConvertHighResToDate*
- *SysTimeRtcConvertHighResToLocal*
- *SysTimeRtcConvertLocalToHighRes*
- *SysTimeRtcConvertLocalToUtc*
- *SysTimeRtcConvertUtcToDate*
- *SysTimeRtcConvertUtcToLocal*
- *SysTimeRtcGet*
- *SysTimeRtcGetTimezone*
- *SysTimeRtcHighResGet*
- *SysTimeRtcHighResSet*
- *SysTimeRtcSet*
- *SysTimeRtcSetTimezone*
- *SysTimeSet*
- *SysTimeUnlock*
- *SysTimeUnSet*
- *SYSTYPE*
- *TableDefinition*
- *TableDefinitions*
- *TableSection*

- *TargetVisuCyclic*
- *TargetVisuFindById*
- *TargetVisuNotify*
- *Task_Desc*
- *Task_Desc2*
- *TASK_GROUP*
- *Task_Info2*
- *TASK_NAME*
- *TASKINFOLIST*
- *TaskLock*
- *TASKPARAM*
- *TASKSTATE*
- *TaskUnlock*
- *tCmpLogAdd*
- *TCP_Client*
- *TCP_Connection*
- *TCP_Processor*
- *TCP_Read*
- *TCP_ReadBuffer*
- *TCP_Reader*
- *TCP_Server*
- *TCP_Write*
- *TCP_WRITE_STATE*
- *TCP_WriteBuffer*
- *TCP_Writer*
- *teClass*
- *teEcatDcControlState*
- *teEcatDevState*
- *teEcatExtSyncInfoFlags*
- *teEcatSlvDCInfoFlags*
- *teErrorCodesOB*
- *teEvent*
- *teHwId*
- *tError*
- *Test_State*
- *Testcases*
- *TEXT*
- *TextCopyToString*
- *TextCopyToWString*
- *TextFree*
- *TextHelper*
- *TextListForCombobox_CIPClass*
- *TICK*
- *TICK_TO_UDINT*
- *TICK_TO_UINT*
- *TICK_TO_ULINT*
- *TicketsSafe*
- *TicketType*
- *TIME_TO_DURATION*
- *TIME_TO_INT64*
- *TIME_TO_ISO8601*
- *TIME_TO_REAL8*
- *TIME_ZONE_INFO*
- *Time2BACnetDateTime*
- *Time2BACnetTimeStamp*

- *TimeElement*
- *TimeElementFactory*
- *TimerSwitch*
- *TIMESTAMP*
- *Timestamp_to_DT*
- *TimeSync_SvcAppHook*
- *TimeZone*
- *TimezoneInformation*
- *TimezoneInformationToString*
- *TimeZoneSegmentToString*
- *TimeZoneToString*
- *TimingControlledBehaviourModelBase*
- *TimingController*
- *TL_AlarmStatus*
- *TL_AlarmTableColumnTitles*
- *TL_DateTime*
- *TL_ElementProperties*
- *TL_RecipeManager*
- *TLR_PACKET_HEADER_T*
- *TLS_VERSION*
- *TLSContext*
- *TODConcat*
- *TODSplit*
- *TOF*
- *TokenTypeToString*
- *TON*
- *TP*
- *TraceAddress*
- *TraceFctGetPropertyValue*
- *TraceFctGetVariableName*
- *TraceFctGetVariableNameW*
- *TraceMgrGetConfigFromFile*
- *TraceMgrGetConfigFromFileRelease*
- *TraceMgrPacketCheckTrigger*
- *TraceMgrPacketClose*
- *TraceMgrPacketComplete*
- *TraceMgrPacketCreate*
- *TraceMgrPacketDelete*
- *TraceMgrPacketDisable*
- *TraceMgrPacketDisableTrigger*
- *TraceMgrPacketEnable*
- *TraceMgrPacketEnableTrigger*
- *TraceMgrPacketGetAbsoluteStartTime*
- *TraceMgrPacketGetChangeTimestamp*
- *TraceMgrPacketGetConfig*
- *TraceMgrPacketGetFirst*
- *TraceMgrPacketGetNext*
- *TraceMgrPacketGetStartTime*
- *TraceMgrPacketGetState*
- *TraceMgrPacketOpen*
- *TraceMgrPacketReadBegin*
- *TraceMgrPacketReadEnd*
- *TraceMgrPacketReadFirst*
- *TraceMgrPacketReadFirst2*
- *TraceMgrPacketReadNext*

- *TraceMgrPacketReadNext2*
- *TraceMgrPacketResetTrigger*
- *TraceMgrPacketRestart*
- *TraceMgrPacketRestore*
- *TraceMgrPacketStart*
- *TraceMgrPacketStop*
- *TraceMgrPacketStore*
- *TraceMgrRecordAdd*
- *TraceMgrRecordGetConfig*
- *TraceMgrRecordGetFirst*
- *TraceMgrRecordGetNext*
- *TraceMgrRecordRemove*
- *TraceMgrRecordUpdate*
- *TraceMgrRecordUpdate2*
- *TraceMgrRecordUpdate3*
- *TraceMgrRecordUpdate4*
- *TraceMgrRecordUpdate5*
- *TracePacketConfiguration*
- *TraceRecordConfiguration*
- *TraceRecordEntry*
- *TraceState*
- *TraceTrigger*
- *TraceVariable*
- *TraceVariableAddress*
- *TraceVarInfo*
- *TRANSITION_STATE*
- *TransmissionTrigger*
- *TransmitParameterGroup*
- *Tree*
- *TreeBase*
- *TreeNode*
- *TreeNodeFactory*
- *TreeNodeType*
- *TrendFbDatabaseAccessErrorHandler*
- *TrendFbTrendStorageWriterReader*
- *TrendFctCursorSearchFirstRow*
- *TrendFctGetTimestamp*
- *TrendFctSetComplexElementCallState*
- *TrendFctShowLossOfPrecisionWarning*
- *TrendFctShowUnsupportedFunctionWarning*
- *TrendLog*
- *TrendStorageConvertFromTimestamp*
- *TrendStorageConvertToTimestamp*
- *TrendStorageReader*
- *TrendStorageReaderValueConverter*
- *TrendStorageVariableDescription*
- *TriggerState*
- *TriggerValue*
- *TrimEnd*
- *TrimStart*
- *Truncate*
- *TruncateF*
- *tsEcmExtSyncInfo*
- *tsEcmMstrDcInfo*
- *tsEcmMstrFrameLossCnt*

- *tsEcmMstrFrameLossCntEntry*
- *tsEcmMstrInfo*
- *tsEcmMstrMemInfo*
- *tsEcmMstrThresholdCnt*
- *tsEcmMstrThresholdCntEntry*
- *tsEcmMstrTimingInfo*
- *tsEcmSlvConnInfo*
- *tsEcmSlvDcInfo*
- *tsEcmSlvEmergencies*
- *tsEcmSlvEmergency*
- *tsEcmSlvESCVersion*
- *tsEcmSlvInfo*
- *tsEcmSlvLostLinkCnt*
- *tsEcmSlvRxErrorCnt*
- *tsNetxEcatBusScanDeviceInfo*
- *tsNetxEcatHandle*
- *tsParameterStruct*
- *tSysComClose*
- *tSysComGetSettings*
- *tSysComOpen*
- *tSysComOpen2*
- *tSysComPurge*
- *tSysComRead*
- *tSysComSetSettings*
- *tSysComSetTimeout*
- *tSysComWrite*
- *tSysDirClose*
- *tSysDirCreate*
- *tSysDirDelete*
- *tSysDirGetCurrent*
- *tSysDirOpen*
- *tSysDirRead*
- *tSysDirRename*
- *tSysDirSetCurrent*
- *tSysFileClose*
- *tSysFileCopy*
- *tSysFileDelete*
- *tSysFileDeleteByHandle*
- *tSysFileEOF*
- *tSysFileGetName*
- *tSysFileGetPath*
- *tSysFileGetPos*
- *tSysFileGetSize*
- *tSysFileGetSizeByHandle*
- *tSysFileGetStatus*
- *tSysFileGetTime*
- *tSysFileOpen*
- *tSysFileRead*
- *tSysFileRename*
- *tSysFileSetPos*
- *tSysFileWrite*
- *tSysPortIn*
- *tSysPortInD*
- *tSysPortInW*
- *tSysPortOut*

- *tSysPortOutD*
- *tSysPortOutW*
- *tSysShmClose*
- *tSysShmGetPointer*
- *tSysShmOpen*
- *tSysShmRead*
- *tSysShmReadByte*
- *tSysShmWrite*
- *tSysShmWriteByte*
- *tSysSockAccept*
- *tSysSockBind*
- *tSysSockClose*
- *tSysSockCloseUdp*
- *tSysSockConnect*
- *tSysSockCreate*
- *tSysSockCreateUdp*
- *tSysSockGetHostByName*
- *tSysSockGetHostname*
- *tSysSockGetOption*
- *tSysSockGetOsHandle*
- *tSysSockGetRecvSizeUdp*
- *tSysSockGetSubnetMask*
- *tSysSockHtonl*
- *tSysSockHtons*
- *tSysSockInetAddr*
- *tSysSockInetNtoa*
- *tSysSockIoctl*
- *tSysSockListen*
- *tSysSockNtohl*
- *tSysSockNtohs*
- *tSysSockPing*
- *tSysSockRecv*
- *tSysSockRecvFrom*
- *tSysSockRecvFromUdp*
- *tSysSockSelect*
- *tSysSockSend*
- *tSysSockSendTo*
- *tSysSockSendToUdp*
- *tSysSockSetIpAddress*
- *tSysSockSetOption*
- *tSysSockSetSubnetMask*
- *tSysSockShutdown*
- *tTaskInfo*
- *tyIEC61850_ASN1_Header*
- *tyIEC61850_AT_AnalogueValue*
- *tyIEC61850_AT_AnalogueValue_Struct*
- *tyIEC61850_AT_APC*
- *tyIEC61850_AT_APC_Operate*
- *tyIEC61850_AT_APC_Operate_SP*
- *tyIEC61850_AT_APC1*
- *tyIEC61850_AT_BOOLEAN*
- *tyIEC61850_AT_BSC_Operate*
- *tyIEC61850_AT_Check*
- *tyIEC61850_AT_CODED_ENUM*
- *tyIEC61850_AT_DPC_Operate*

- *tyIEC61850_AT_DstAddress*
- *tyIEC61850_AT_EntryTime*
- *tyIEC61850_AT_ENUM_CtlModels*
- *tyIEC61850_AT_ENUM_MODE*
- *tyIEC61850_AT_ENUM_SboClass*
- *tyIEC61850_AT_ENUMERATED*
- *tyIEC61850_AT_FLOAT32*
- *tyIEC61850_AT_INC*
- *tyIEC61850_AT_INC_Operate*
- *tyIEC61850_AT_INC1*
- *tyIEC61850_AT_INS*
- *tyIEC61850_AT_INT128*
- *tyIEC61850_AT_INT16*
- *tyIEC61850_AT_INT16U*
- *tyIEC61850_AT_INT32*
- *tyIEC61850_AT_INT32U*
- *tyIEC61850_AT_INT8*
- *tyIEC61850_AT_INT8U*
- *tyIEC61850_AT_ISC_Operate*
- *tyIEC61850_AT_Octet255*
- *tyIEC61850_AT_Octet64*
- *tyIEC61850_AT_Origin*
- *tyIEC61850_AT_POINT*
- *tyIEC61850_AT_PulseConfig*
- *tyIEC61850_AT_Quality*
- *tyIEC61850_AT_RANGECONFIG*
- *tyIEC61850_AT_ScaledValConfig*
- *tyIEC61850_AT_SPC*
- *tyIEC61850_AT_SPC_Operate*
- *tyIEC61850_AT_StatusValue_Struct*
- *tyIEC61850_AT_TimeStamp*
- *tyIEC61850_AT_UCSTRING255*
- *tyIEC61850_AT_UINT32*
- *tyIEC61850_AT_UNIT*
- *tyIEC61850_AT_ValWithTrans*
- *tyIEC61850_AT_VECTOR*
- *tyIEC61850_AT_VisSTRING129*
- *tyIEC61850_AT_VisSTRING255*
- *tyIEC61850_AT_VisSTRING32*
- *tyIEC61850_AT_VisSTRING64*
- *tyIEC61850_AT_VisSTRING65*
- *tyIEC61850_DataPoint*
- *tyIEC61850_DataSetRef*
- *tyIEC61850_GOOSE_Check*
- *tyIEC61850_GOOSEMsg*
- *tyIEC61850_MMS_DataExchange*
- *tyIEC61850_MMS_Initiate*
- *tyIEC61850_SubsDataBlock*
- *tyIEC61850_SubsDataPoint*
- *tyISO_BlockHeader*
- *tyISO_SPDU*
- *tyISO8073_BlockHeader*
- *tyISO8073_ClientPara*
- *tyISO8073_PDU*
- *tyISO8327_BlockHeader*

- *tyISO8327_ClientData*
- *tyISO8327_Connect_AcceptItem*
- *tyISO8327_ConnectionIdent*
- *tyISO8650_UserInfoData*
- *tyISO8823_ContextList*
- *tyISO8823_ContextName*
- *tyISO8823_CP_Type*
- *tyISO8823_DataUser*
- *tyISO8823_NormalModePara*
- *TypeClass*
- *TypeClass3*
- *TypeClassToVariantId*
- *TypedElement*
- *TypeDesc*
- *TypeDesc_Alias*
- *TypeDesc_AliasWithAttributes*
- *TypeDesc_Array*
- *TypeDesc_Array_ByteAddressed*
- *TypeDesc_Array_Remote*
- *TypeDesc_Enum*
- *TypeDesc_EnumWithAttributes*
- *TypeDesc_Executable*
- *TypeDesc_Executable2*
- *TypeDesc_Property*
- *TypeDesc_Property_Remote*
- *TypeDesc_Reference*
- *TypeDesc_Simple*
- *TypeDesc_Simple_Bit*
- *TypeDesc_Struct*
- *TypeDesc_Struct_Derived_Remote*
- *TypeDesc_Struct_Remote*
- *TypeDesc_Struct2*
- *TypeDesc_Struct2_WithBaseType*
- *TypeDesc_Struct2_WithBaseTypeAndAttributes*
- *TypeDesc_Subrange*
- *TypeDesc_VarLenArray*
- *TypeDescArrayAsStruct*
- *TypeDescAsUnion*
- *TypeDescSimpleAsStruct*
- *TypeDescStructAsStruct*
- *TypeDescUnion*
- *TypeDescVarArrayAsStruct*
- *TypedList*
- *TypedTree*
- *TypeHasCompleteBlittableLayout*
- *TZ_NAME*
- *UDINT_IN_BYTES*
- *UDINT_IN_WORDS*
- *UDINT_TO_COUNT*
- *UDINT_TO_HEX*
- *UDINT_TO_IPARRAY*
- *UDINT_TO_IPSTRING*
- *UDINT_TO_SIZE*
- *UDINT_TO_TICK*
- *UDINT_TO_UNSIGNED*

- *UdintElement*
- *UdintElementFactory*
- *UDP_GetDataSize*
- *UDP_Peer*
- *UDP_Processor*
- *UDP_Receive*
- *UDP_ReceiveBuffer*
- *UDP_Receiver*
- *UDP_REPLY*
- *UDP_REPLY2*
- *UDP_Send*
- *UDP_SendBuffer*
- *UDP_Sender*
- *UDPDriver*
- *UdpGetReceiveDataSize*
- *UdpOpenReceiveSocket*
- *UdpOpenSendSocket*
- *UdpReceiveData*
- *UdpSendData*
- *UINT_TO_COUNT*
- *UINT_TO_HEX*
- *UINT_TO_SIZE*
- *UINT_TO_TICK*
- *UINT_TO_UNSIGNED*
- *UintElement*
- *UintElementFactory*
- *ULINT_TO_COUNT*
- *ULINT_TO_SIZE*
- *ULINT_TO_TICK*
- *ULINT_TO_UNSIGNED*
- *UlintElement*
- *UlintElementFactory*
- *unexport*
- *UNION_BACNET_ADDRESS*
- *UNION_BACNET_ADDRESS_TO_STRING*
- *UNION_BACNET_CALENDAR_ENTRY*
- *UNION_BACNET_CHANNEL_VALUE*
- *UNION_BACNET_EP_COV_PARAM*
- *UNION_BACNET_EPFP_E_PARAMETER*
- *UNION_BACNET_ERROR*
- *UNION_BACNET_EVENT_LOG_RECORD*
- *UNION_BACNET_EVENT_PARAMETER*
- *UNION_BACNET_FAULT_PARAMETER*
- *UNION_BACNET_LOG_RECORD*
- *UNION_BACNET_LOG_RECORD_M_ENTRY*
- *UNION_BACNET_LOG_RECORD_MULTIPLE*
- *UNION_BACNET_MESSAGE_CLASS*
- *UNION_BACNET_NETWORK_MANAGEMENT_MESSAGE*
- *UNION_BACNET_NMM_BVLL*
- *UNION_BACNET_NMM_NETWORK*
- *UNION_BACNET_NOTIFICATION_PARAMETERS*
- *UNION_BACNET_NP_CMD_FAIL_PARAM*
- *UNION_BACNET_NP_CMD_FAIL_PARAM1*
- *UNION_BACNET_NP_COV_PARAM*
- *UNION_BACNET_NP_E_PARAMETER*

- *UNION_BACNET_OBJECT_SPECIFIER*
- *UNION_BACNET_OS_TIME_PROVIDER_VALUE*
- *UNION_BACNET_PRIORITY_ARRAY_ITEM*
- *UNION_BACNET_PROPERTY_ACCESS_RESULT*
- *UNION_BACNET_PROPERTY_STATES*
- *UNION_BACNET_READ_FILE_RESULT*
- *UNION_BACNET_READ_RESULT_ITEM*
- *UNION_BACNET_RECIPIENT*
- *UNION_BACNET_SCALE*
- *UNION_BACNET_SHED_LEVEL*
- *UNION_BACNET_SPECIAL_EVENT*
- *UNION_BACNET_STACK_CONTROL*
- *UNION_BACNET_STACK_DATALINK*
- *UNION_BACNET_STACK_INTERNAL_ERROR*
- *UNION_BACNET_STRING*
- *UNION_BACNET_TIME_STAMP*
- *UNION_BACNET_WHO_HAS_INFO*
- *UNION_BACNET_WHO_HAS_PARAM*
- *UNION_BACNET_WRITE_FILE_RESULT*
- *UNPACK*
- *UnpackArrayOfByte*
- *UnpackByte*
- *UnpackDWord*
- *UnpackWord*
- *Unregister*
- *UnregisterCallback*
- *UnregisterIdArea*
- *UNSIGNED*
- *UNSIGNED_TO_UDINT*
- *UNSIGNED_TO_UINT*
- *UNSIGNED_TO_ULINT*
- *UpdateByDefaultInfo*
- *UpdateByDefaultItem*
- *UpdateDiagnosis_Status*
- *UpdateDiagnosisEntry*
- *UserAuthentification*
- *UserMgrChangeMyPassword*
- *UserMgrGetUserAccessRights*
- *UserMgrHasUserAccessRights*
- *UserMgrIsActive*
- *UserMgrLogin*
- *UserMgrLogout*
- *UserMgrObjectAdd*
- *UserMgrObjectRemove*
- *UserMgrObjectSetUsedRights*
- *UserMgrRelogin*
- *UTC_TO_DT*
- *UTCTimeSync_SvcAppHook*
- *UtilAddrEqual*
- *UtilBytesFromHexSubString*
- *UtilDateTimeToString*
- *UtilFillNodeAddress*
- *UtilGetLocalByteorder*
- *UtilGetLocalByteorderAtRuntime*
- *UtilIsGeneralErrorReply*

- *UtilIsIntelByteorder*
- *UtilIsToSwap*
- *UtilReadAddressFromRouter*
- *UtilsWriteBYTE*
- *UtilsWriteString*
- *UtilsWriteUDINT*
- *UtilsWriteUINT*
- *UtilsWriteULINT*
- *UtilTokenizer*
- *UtilValidateByteOrder*
- *UUID*
- *UUID_COMPARE*
- *UUIDGenerator*
- *ValueToString*
- *VarAccUaNamespaceFragment*
- *VariableInformation*
- *VariableInformationStruct*
- *VariableInformationStruct2*
- *VariableInformationStruct3*
- *VariableInformationStruct4*
- *VariableInformationStruct5*
- *VariableValue*
- *VARIANCE*
- *Variance*
- *VarListDefine*
- *VarListDelete*
- *VarListDisable*
- *VarListEnable*
- *VarListEnter*
- *VarListFlags*
- *VarListLeave*
- *VarListRead*
- *VECTOR3D*
- *Verifier*
- *VERSIONINFO*
- *Visu_Assert*
- *Visu_CheckPropertyInfo*
- *Visu_ClientTagData*
- *Visu_FbClearEventsAfterStart*
- *Visu_FbStringDintMap*
- *Visu_FbWebserver*
- *Visu_FctCheckForLongFormatSpecifier*
- *Visu_FctClosePAADialogIfNecessary*
- *Visu_FctGetDatasources*
- *Visu_FctHandleInputGesture*
- *Visu_FctHandleVisuInputDialogTarget*
- *Visu_FctHandleVisuInputMouseEvent*
- *Visu_FctHandleVisuInputOverlayMeasureString*
- *Visu_FctHandleVisuInputPAA*
- *Visu_FctInitMemSet*
- *Visu_FctIsEventToIgnoreWhileEditboxOpen*
- *Visu_FctIsGestureEvent*
- *Visu_FctIsModalDialogOpen*
- *Visu_FctIsOnStraightLine*
- *Visu_FctIsRelevantGestureEvent*

- *Visu_FctIsSelectionEmpty*
- *Visu_FctLegacyIDStackInfoFill*
- *Visu_FctLegacyIDStackInfoReadFromAdditionalData*
- *Visu_FctRaiseMouseLeave*
- *Visu_FctReleaseNonIECMemClientResources*
- *Visu_FctTransformSelectionIsotropicOverlay*
- *Visu_HelpDumpLibHierarchy*
- *Visu_OnlinechangeNotify*
- *Visu_ScalarTypesUnion*
- *Visu_ScalarTypesWithPtr*
- *Visu_SetCodegenFeatureSupport*
- *Visu_StructCommandData*
- *VisuAlarmScrollValueProvider*
- *VisuBenchmarkFBStatistics*
- *VisuBenchmarkNowInUs*
- *VisuClientAnimationData*
- *VisuClientObject*
- *VisuClientObjectClientSpecificData*
- *VisuClientObjectFlags*
- *VisuClientObjectIdStack*
- *VisuClientObjectIdStackOptimized*
- *VisuClientObjectIdStackWithParentSize*
- *VisuClientObjectInputRectangleMgr*
- *VisuClientObjectLayerInitFlags*
- *VisuClientObjectMgr*
- *VisuClientObjectReservedIds*
- *VisuClientObjectStateFlags*
- *VisuClientTag*
- *VisuClientType*
- *VisuDateTimeFormatPlaceholders*
- *VisuDialogOpenFlags*
- *VisuElemLayer*
- *VisuElemLayerAlignmentFlag*
- *VisuElemLayerClientSpecificData*
- *VisuElemLayerData*
- *VisuElemLayerFlag*
- *VisuElemMgrClientData*
- *VisuElemMgrClientSpecificData*
- *VisuElemMgrClientSpecificDataIndices*
- *VisuElemSelectionLayer*
- *VisuEnumActionType*
- *VisuEnumAfterTransformation*
- *VisuEnumAlarmDataType*
- *VisuEnumAnalogClockStyle*
- *VisuEnumBackgroundDrawingState*
- *VisuEnumClientTag*
- *VisuEnumCreateTemporaryRenderLocationFlags*
- *VisuEnumFileTransferDirection*
- *VisuEnumFileTransferError*
- *VisuEnumInputOnElementType*
- *VisuEnumLegendDisplayerLineType*
- *VisuEnumRectangleFlags*
- *VisuEnumRedundancyValueChanged*
- *VisuEnumXYChartActivityType*
- *VisuEnumXYChartAxisPosition*

- *VisuEnumXYChartAxType*
- *VisuEnumXYChartBarType*
- *VisuEnumXYChartBgType*
- *VisuEnumXYChartCommands*
- *VisuEnumXYChartCursorActive*
- *VisuEnumXYChartCursorType*
- *VisuEnumXYChartCursorVisible*
- *VisuEnumXYChartCvChartType*
- *VisuEnumXYChartCvFillType*
- *VisuEnumXYChartCvHeapCmd*
- *VisuEnumXYChartCvOverlapType*
- *VisuEnumXYChartFocusType*
- *VisuEnumXYChartGradientType*
- *VisuEnumXYChartGridType*
- *VisuEnumXYChartLineType*
- *VisuEnumXYChartLvlLineLbPos*
- *VisuEnumXYChartPointStyle*
- *VisuEnumXYChartProgType*
- *VisuEnumXYChartShadowStyle*
- *VisuEnumXYChartZeroLineType*
- *VisuEventOptimization*
- *VisuEventTarget*
- *VisuFbAlarmBannerDataBlock*
- *VisuFbAnalyzeDateTimeFormatExtractWithoutWeekdays*
- *VisuFbAnalyzeDateTimeFormatStringBase*
- *VisuFbAnalyzeDateTimeFormatStringMinSecOnly*
- *VisuFbAnalyzeStateVarsTapAware*
- *VisuFbAnalyzeTextVarsDateTimeOnly*
- *VisuFbBaseVector*
- *VisuFbCapturedTransformationProvider*
- *VisuFbClientLogger*
- *VisuFbClientStartVisuMgr*
- *VisuFbClientTagDataHelper*
- *VisuFbCommandVector*
- *VisuFbDatasourcesResourceEntries_MBM*
- *VisuFbDatasourcesResourceEntries_SysMem*
- *VisuFbDateTimeNamesLocalizer*
- *VisuFbDialogClientInfo*
- *VisuFbDialogInfoVector*
- *VisuFbDWORDVector*
- *VisuFbElemTextEditor*
- *VisuFbExecution*
- *VisuFbFileTransferManager*
- *VisuFbFrameRegistrationVector*
- *VisuFbGestureFromEvent*
- *VisuFbGroupOverlay*
- *VisuFbInputRectangle*
- *VisuFbLegacyCapturingTransformationProvider*
- *VisuFbLibHierarchy*
- *VisuFbMainClientMgmt*
- *VisuFbMouseTouchDragUtil*
- *VisuFbMoveAbsoluteTapAware*
- *VisuFbMoveAbsoluteTapAwareF*
- *VisuFbMoveRelativeTapAware*
- *VisuFbNamespaceTable*

- *VisuFbNamespaceTableHelper*
- *VisuFbNativeControlItemVector*
- *VisuFbPaintAfterAllDialog*
- *VisuFbPaintAfterAllElement*
- *VisuFbPaintRectF*
- *VisuFbPointF*
- *VisuFbPrintDateTimeFormatBase*
- *VisuFbPrintDateTimeFormatCurrentTime*
- *VisuFbPrintDateTimeFormatVariable*
- *VisuFbRectangleListManager*
- *VisuFbRectF*
- *VisuFbResourcesEntryVector*
- *VisuFbScalingInfo*
- *VisuFbSizeF*
- *VisuFbTabControlOverlayTabs*
- *VisuFbTemporaryPolygon*
- *VisuFbTickMarkDrawer2*
- *VisuFbTransformationCommon*
- *VisuFbTransformationScrolling*
- *VisuFbVisuVector*
- *VisuFbWriteDateTimeVariableFormatted*
- *VisuFbXYChartDataProvider*
- *VisuFbXYChartDataProviderAxis*
- *VisuFbXYChartDataProviderCurve*
- *VisuFbXYChartGenericVariable*
- *VisuFbXYChartGenericVariableArray*
- *VisuFct_IsBehindOverlayElement*
- *VisuFctAddChecksumBool*
- *VisuFctAddChecksumForConverted*
- *VisuFctAddClientToEventQueue*
- *VisuFctAssignValue*
- *VisuFctCalculateCompleteSurroundingSimpleRectOfElemArray*
- *VisuFctCalculateMaxTooltipLength*
- *VisuFctCalculateSurroundingSimpleRectOfElemArray*
- *VisuFctCheckClientSupportsTouch*
- *VisuFctClearElementEntries*
- *VisuFctClearEventIdStack*
- *VisuFctConfigureTextBufferSize*
- *VisuFctCreateEventMapIfNeeded*
- *VisuFctCreateIdStack*
- *VisuFctDatasourcesResourceEntryAllocatorGet*
- *VisuFctDatasourcesResourceEntryAllocatorGet_MBM*
- *VisuFctDatasourcesResourceEntryAllocatorGet_SysMem*
- *VisuFctDrawCircle*
- *VisuFctDrawDot*
- *VisuFctDrawDot2*
- *VisuFctDrawImage*
- *VisuFctDrawLine*
- *VisuFctDrawLineEx*
- *VisuFctDrawLineExUntransformed*
- *VisuFctDrawLineUntransformed*
- *VisuFctDrawPolygon*
- *VisuFctDrawPolyline*
- *VisuFctDrawPolyline2*
- *VisuFctDrawText*

- *VisuFctEvaluatePanGesture*
- *VisuFctEventIdStackGetValue*
- *VisuFctEventIdStackGetValuePtr*
- *VisuFctEventIdStackGetValuePtrFromEvent2*
- *VisuFctEventIdStackGetValuePtrFromEventLegacy*
- *VisuFctEventIdStackHas*
- *VisuFctEventIdStackPopHelp*
- *VisuFctEventIdStackPopId*
- *VisuFctEventIdStackPopTarget*
- *VisuFctEventIdStackPopVisuVersion*
- *VisuFctEventIdStackPushId*
- *VisuFctEventIdStackSetValue*
- *VisuFctExitVisuClientObject*
- *VisuFctFillPolygon*
- *VisuFctFillPolygon2*
- *VisuFctFillPolygon3*
- *VisuFctFillRectangle*
- *VisuFctFreeClientTagData*
- *VisuFctGetAbsolutePosition*
- *VisuFctGetClientName*
- *VisuFctGetEffectiveTextProperties*
- *VisuFctGetElementClientData*
- *VisuFctGetElementEntry*
- *VisuFctGetElementState*
- *VisuFctGetGradient*
- *VisuFctGetLineJoinMiterLimit*
- *VisuFctGetMeasureString2Result*
- *VisuFctGetMeasureStringApprox*
- *VisuFctGetMeasureStringResult*
- *VisuFctGetMultitouchActive*
- *VisuFctGetMultitouchScrollbarsActive*
- *VisuFctGetPaintRectFromSimpleRect*
- *VisuFctGetRectangleFromPaintRect*
- *VisuFctGetRectangleFromSimpleRect*
- *VisuFctGetRectHeight*
- *VisuFctGetRectWidth*
- *VisuFctGetShadowColor*
- *VisuFctGetTargetVisuTouchFlags*
- *VisuFctGetTransparentValue*
- *VisuFctHandleInputOnElementEvent*
- *VisuFctHandleInputVisuClientObject*
- *VisuFctHandleInputWithoutInputHandler*
- *VisuFctIncreaseSimpleRectIfRotated*
- *VisuFctInitFlagsVisuClientObject*
- *VisuFctInitVisuClientObject*
- *VisuFctIsDegenerateRectangle*
- *VisuFctIsMultitouchClient*
- *VisuFctIsRectangleRotated*
- *VisuFctIsToPaintSelection*
- *VisuFctIsTransparentBackground*
- *VisuFctLimitSimpleRectangleSize*
- *VisuFctMainClientsCheck*
- *VisuFctMainClientsCheckOld*
- *VisuFctPaintSelection*
- *VisuFctPaintVisuClientObject*

- *VisuFctRectSize*
- *VisuFctRemoveClientFromEventQueue*
- *VisuFctSelectElement*
- *VisuFctSetClientDataVisuClientObject*
- *VisuFctSetMaxElementCountPaintAfterAll*
- *VisuFctSetNumericValue*
- *VisuFctSetRectangleUpdateNecessaryOnAllClients*
- *VisuFctSetSelectionChanged*
- *VisuFctSetSimpleRect*
- *VisuFctSimpleRectangleFToSimpleRectangle*
- *VisuFctSplitColor*
- *VisuFctTestLReal*
- *VisuFctTestReal*
- *VisuFctTextEditorGetErrorText*
- *VisuFctTryAtomicAssignValueBySize*
- *VisuFctTryAtomicAssignValueByType*
- *VisuFctWriteValueIfValid*
- *VisuGestureInfo*
- *VisuInput_CheckUpdateElementStatePossible_DependingOnCurrentInput*
- *VisuRegistrationHelpDuringDecl*
- *VisuScrollValueData*
- *VisuScrollValueProvider*
- *VisuStructAllDialogInfo*
- *VisuStructAllModalDialogInfo*
- *VisuStructAllNonModalDialogInfo*
- *VisuStructBackgroundAndStaticElementDrawing*
- *VisuStructButtonClientSpecificData*
- *VisuStructClientTagData*
- *VisuStructCompleteSurroundingRectInfo*
- *VisuStructComplexFrameClientSpecificData*
- *VisuStructElementClientData*
- *VisuStructElementClientDataExtended*
- *VisuStructFindElementEvent*
- *VisuStructFlickInfo*
- *VisuStructIECTouchInfo*
- *VisuStructInputInfo*
- *VisuStructInputOnElementEvent*
- *VisuStructLegendDisplayerCheckBoxPos*
- *VisuStructLegendDisplayerCheckBoxStatus*
- *VisuStructNamespace*
- *VisuStructNamespaceProjectIdent*
- *VisuStructPAADialogClientSpecificData*
- *VisuStructPanInfo*
- *VisuStructPoint*
- *VisuStructPointD*
- *VisuStructPolygonClientSpecificData*
- *VisuStructRadius*
- *VisuStructRectangularElementUtilBaseClientSpecificData*
- *VisuStructScaleScrollInfo*
- *VisuStructSimpleRectangleD*
- *VisuStructSimpleRectWithBorder*
- *VisuStructSingleIECTouchInfo*
- *VisuStructSpreadPinchInfo*
- *VisuStructTopMostDialogInfo*
- *VisuStructTraceGradientColor*

- *VisuStructUpdateRectangle*
- *VisuStructWaitingCubeClientSpecificData*
- *VisuStructWaitingFlowerClientSpecificData*
- *VisuStructXYChart*
- *VisuStructXYChartAxis*
- *VisuStructXYChartCurve*
- *VisuStructXYChartGradientColor*
- *VisuStructXYChartLevelLine*
- *VisuStyleFct_GetImageAccordingMapping*
- *VisuTaskOpClientBase*
- *VisuTouchState*
- *VUM_EditType*
- *VUM_ReturnValues*
- *VUM_User*
- *WARNING_ID*
- *WCharToUpper*
- *WCONCAT*
- *WDELETE*
- *WEEK*
- *WEEKDAY*
- *WeekOfYear*
- *WFIND*
- *WINSERT*
- *WLEFT*
- *WLEN*
- *WMID*
- *WORD_AS_BIT*
- *WORD_AS_STRING*
- *WORD_TO_BCD*
- *WORD_TO_GRAY*
- *WORD_TO_HANDLE*
- *WORD_TO_IDENT*
- *WORD_TO_PVOID*
- *WorkerRegister*
- *WorkerUnregister*
- *WRAP_FB_INIT_STRUCT*
- *WRAP_INITIALIZE_STRUCT*
- *WREPLACE*
- *WRIGHT*
- *Write*
- *writeBit*
- *WriteBootProject*
- *WriteCfgThumb*
- *WriteMemory*
- *WriteRequest*
- *WRREC*
- *WStringElement*
- *WStringElementFactory*
- *WStringsEqual*
- *X509CertCheckHost*
- *X509CertCheckIP*
- *X509CertClose*
- *X509CertCmsDecrypt*
- *X509CertCmsVerify*
- *X509CertCreateCSR*

- *X509CertCreateSelfSigned*
- *X509CertGetBinary*
- *X509CertGetContent*
- *X509CertGetPrivateKey*
- *X509CertGetPublicKey*
- *X509CertGetThumbprint*
- *X509CertHasExtendedKeyUsage*
- *X509CertInfoExit*
- *X509CertInfoInit*
- *X509CertIsDateValid*
- *X509CertIsSelfSigned*
- *X509CertKeyClose*
- *X509CertStoreAddCert*
- *X509CertStoreClose*
- *X509CertStoreGetFirstCert*
- *X509CertStoreGetNextCert*
- *X509CertStoreGetRegisteredCert*
- *X509CertStoreOpen*
- *X509CertStoreRegister*
- *X509CertStoreRemoveCert*
- *X509CertStoreSearchGetFirst*
- *X509CertStoreSearchGetNext*
- *X509CertStoreUnregister*
- *X509CertVerify*
- *X509ParseCertificate*
- *XChgClass*
- *XChgCreateH*
- *XChgCreateP*
- *XChgDelete*
- *XChgExtendH*
- *XChgGetSize*
- *XChgIsEmpty*
- *XChgMsgLeft*
- *XWORD*
- *XwordVector*
- *YEAR*

1.11 Contact ABB

ABB AG

Eppelheimer Str. 82

69123 Heidelberg, Germany

PLC support: +49 (0)6221 701 1444, plc.support@de.abb.com

abb.com/plc

abb.com/automationbuilder

abb.com/contacts

2 Index

1, 2, 3 ...

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

—
ABB AG
Eppelheimer Str. 82
69123 Heidelberg, Germany
Telephone: +49 (0)6221 701 1444
E-mail: plc.support@de.abb.com
abb.com/plc
abb.com/automationbuilder
abb.com/contacts

3ADR010583, 3, en_US